Querying a Web of Linked Data

Foundations and Query Execution

D I S S E R T A T I O N

zur Erlangung des akademischen Grades

Dr. Rer. Nat.
im Fach Informatik

eingereicht an der
Mathematisch-Wissenschaftlichen Fakultät II
Humboldt-Universität zu Berlin

von

Dipl.-Inf. Olaf Hartig

eingereicht am: 19. März 2014
Tag der mündlichen Prüfung: 10. Juli 2014

Abstract

During recent years a set of best practices for publishing and connecting structured data on the World Wide Web (WWW) has emerged. These best practices are referred to as the Linked Data principles and the resulting form of Web data is called *Linked Data*. The increasing adoption of these principles has lead to the creation of a globally distributed space of Linked Data that covers various domains such as government, libraries, life sciences, and media. Approaches that conceive this data space as a huge distributed database and enable an execution of declarative queries over this database hold an enormous potential; they allow users to benefit from a virtually unbounded set of up-to-date data. As a consequence, several research groups have started to study such approaches. However, the main focus of existing work is to address practical challenges that arise in this context. Research on the foundations of such approaches is largely missing. This dissertation closes this gap.

This dissertation first establishes a well-defined framework for defining and studying queries over Linked Data on the WWW. In particular, we introduce a data model that enables us to formally conceive Linked Data on the WWW as a (distributed) database and a computation model that captures the capabilities of a query execution system for this database. Based on these models, we adapt the declarative query language SPARQL to the given scenario. More precisely, we define a full-Web query semantics and a family of reachability-based query semantics such that each of these query semantics presents a well-defined basis for using SPARQL to query Linked Data on the WWW. Thereafter, we show theoretical properties of queries under these query semantics. Perhaps the most important result of this study formally verifies the common assumption that a computation of query results that are complete w.r.t. all Linked Data on the WWW, is not feasible. However, we also identify classes of queries for which the computational feasibility is less limited.

After analyzing queries over Linked Data independent of specific approaches for executing such queries, this dissertation focuses on a particular execution approach and studies fundamental aspects thereof. The studied approach presents a general strategy for executing queries by integrating traversal-based data retrieval into the result construction process. To analyze this notion of traversal-based query execution formally, we define it in the form of an abstract query execution model. In addition, we discuss a concrete implementation approach for our execution model; this approach is based on the well-known concept of iterators. Our analysis of both the execution model and the iterator-based implementation shows that (i) for one of our reachability-based query semantics, the given notion of traversal-based query execution, in general, is sound and complete, whereas (ii) for the same query semantics, the specific, iterator-based implementation approach cannot guarantee completeness of query results. Based on an experimental analysis we verify that the latter limitation has a significant impact in practice.

Zusammenfassung

In den letzten Jahren haben sich spezielle Prinzipien zur Veröffentlichung strukturierter Daten im World Wide Web (WWW) etabliert. Diese Prinzipien erlauben es, von den jeweils angebotenen Daten auf weitere, nach den selben Prinzipien veröffentlichten Daten zu verweisen. Die daraus resultierende Form von Web-Daten wird entsprechend als *Linked Data* bezeichnet. Mit der Veröffentlichung von Linked Data im WWW entsteht ein sehr großer Datenraum, welcher Daten verschiedenster Anbieter miteinander verbindet und neuartige Möglichkeiten für Web-basierte Anwendungen bietet. Als Basis für die Entwicklung solcher Anwendungen haben mehrere Forschungsgruppen begonnen, Ansätze zu untersuchen, welche diesen Datenraum als eine Art verteilte Datenbank auffassen und die Ausführung deklarativer Anfragen über dieser Datenbank ermöglichen. Forschungsarbeit zu theoretischen Grundlagen der untersuchten Ansätze fehlt jedoch nahezu vollständig. Das Ziel der vorliegenden Dissertation ist es mitzuhelfen, diese Lücke zu schließen.

Die Basis der Dissertation bilden ein Datenmodell und ein Berechnungsmodell. Während das Datenmodell das Konzept eines *Web of Linked Data* als (verteilte) Datenbank formalisiert und eine exakte Definition von Anfragesemantiken ermöglicht, formalisiert das Berechnungsmodell das Leistungsvermögen von Systemen, welche Anfragen über dieser Datenbank ausführen können. Auf Basis dieser Modelle wird die, vom WWW-Consortium (W3C) spezifizierte Anfragesprache SPARQL um verschiedene Anfragesemantiken für eine Nutzung im Linked-Data-Kontext erweitert und die, sich unter diesen Semantiken ergebenden Anfragen bezüglich ihrer Berechenbarkeit entsprechend unseres Berechnungsmodells untersucht. Insbesondere führt die Arbeit eine Menge von erreichbarkeitsbasierten Anfragesemantiken und eine unbegrenzte Anfragesemantik (engl.: full-Web query semantics) ein. Als Hauptergebnis der Untersuchung dieser Anfragesemantiken liefert die Arbeit einen formalen Beweis für die weit verbreitete Annahme, dass eine Berechnung vollständiger Anfrageergebnisse in Bezug auf die komplette Menge an Linked Data im WWW nicht möglich ist. Weitere Ergebnisse der Untersuchung identifizieren Klassen von Anfragen, deren vollständige Berechnung unter gewissen Bedingungen möglich ist.

Neben der Analyse theoretischer Eigenschaften von SPARQL-basierten Anfragen über Linked Data im WWW, beschäftigt sich die Dissertation mit einem verweisbasierten Ansatz zur Ausführung solcher Anfragen. Hierbei stehen insbesondere grundsätzliche Eigenschaften wie Terminierung der Anfrageausführung, sowie Korrektheit und Vollständigkeit im Zentrum der Untersuchung. Die Kernidee des untersuchten Ansatzes besteht darin, die Bestimmung des Ergebnisses einer gegebenen Anfrage mit dem Verfolgen von Verweisen im angefragten Web zu kombinieren und somit potentiell relevante Daten während der Anfrageausführung zu entdecken. Um diese Idee formal untersuchen zu können, wird ein abstraktes Anfrageausführungsmodell eingeführt. Zusätzlich wird eine konkrete, auf dem Konzept eines Iterators basierende Möglichkeit zur Umsetzung dieses abstrakten Modells besprochen.

Die formale Analyse zeigt, dass (i) der generelle Ansatz einer verweisbasierten Anfrageausführung korrekt und vollständig bezüglich einer, der eingeführten erreichbarkeitsbasierten Anfragesemantiken ist, während (ii) die konkrete, iteratorbasierte Umsetzung die Vollständigkeit von Anfrageergebnissen bezüglich derselben Anfragesemantik nicht garantieren kann. Eine experimentelle Analyse untersucht die iteratorbasierte Umsetzung eingehender und zeigt, dass die theoretische Möglichkeit von unvollständigen Anfrageergebnissen auch in der Praxis eine maßgebliche Rolle spielt.

Contents

List of Figures

List of Tables

1. Introduction

Since its emergence, the World Wide Web (WWW) has attracted research interest in adopting database techniques for retrieving information from the WWW. The main motivation for such an attraction was—and still is—*"the popularity of the [WWW as] a prime vehicle for disseminating information"* [49].

However, approaches traditionally used for disseminating information on the WWW focus on human users as (direct) consumers of WWW content; as a result, there exist major practical hurdles for automated query processing over data available in (traditional) Web pages. In particular, the data in such Web pages is at best semi-structured [2] and a large percentage of these pages is "hidden" behind form-based interfaces designed for human users [82]. This situation has quickly led to a divergence from research on *"database-like access to the WWW"* [94] to research necessary to overcome the aforementioned hurdles. Most notable in this context is a plethora of work on extracting structured data from semi-structured or unstructured Web pages [32, 101, 167] and on getting access to such pages by automated completion of Web forms [44, 91, 134]. However, we observe a shortage of more recent work that comes back to the original vision of conceiving the whole WWW (and not just single Web sites) *"as a gigantic database"* [94].

On the other hand, a new set of best practices for publishing and connecting *structured* data on the WWW has emerged [14, 83]. The resulting form of Web data is commonly referred to as *Linked Data* and has gained tremendous momentum in recent years. That is, more and more content providers make their data openly available as Linked Data [23, 24, 114, 118]. This development presents an exciting opportunity to reconsider viewing the WWW as a database. In particular, because the aforementioned problems caused by traditional data publishing approaches do not exist for Linked Data (nonetheless, other characteristics of the WWW present further challenges as we shall discuss shortly).

Consequently, the aim of this dissertation is to study foundations for a database-like access to Linked Data on the WWW. To introduce the particular problems addressed in this dissertation, we first outline the principles for publishing Linked Data and discuss query processing paradigms applicable to the scenario of querying Linked Data.

1.1. Linked Data on the WWW

The publication of Linked Data on the WWW is based on the following four principles, which have become known as the *"Linked Data principles"* [23].

"1. Use URIs as names for things

2. Use HTTP URIs so that people can look up those names.

3. *When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)*

4. *Include links to other URIs, so that they can discover more things."* [14]

Thus, the publication of Linked Data is based on standard Web technologies such as Uniform Resource Identifiers (URIs) [15], the Hypertext Transfer Protocol (HTTP) [47], and the Resource Description Framework (RDF) [92]. In the following, we informally describe how the Linked Data principles propose to use these technologies for publishing and linking structured data on the WWW (a formal definition of concepts relevant to this dissertation follows in Chapter 2).

The Linked Data principles require data providers to identify entities via HTTP scheme based URIs (i.e., URIs that begin with http://), hereafter, simply referred to as *HTTP URIs*. Such a URI does not only serve as a globally unique identifier, it also provides access to a structured data representation of the identified entity. Hence, looking up such a URI via the HTTP protocol yields data about the entity identified by the URI. According to the third principle, this data should be represented using RDF. RDF is a data model that represents information based on triples of the form (*subject*, *predicate*, *object*). By definition, each element of such an RDF triple may be a URI; objects may also be literal values (e.g., a string or a number), and subjects and objects may also be local identifiers for unnamed entities (called *"blank nodes"* [92]). The predicate in an RDF triple specifies the relationship between the subject and the object of that triple.

Example 1.1. The RDF triple

(http://example.org/foaf.rdf#me , http://xmlns.com/foaf/0.1/name , "John Scott")

states that the person identified by the given subject URI is called John Scott, and

(http://example.org/foaf.rdf#me , http://xmlns.com/foaf/0.1/knows , http://acme.com/empl/Jeff)

states that John Scott knows another person identified by the given object URI. □

The semantics of predicate URIs as well as classes of entities are defined in *vocabularies*. The RDF Vocabulary Definition Language [28] and the Web Ontology Language [87] allow users to define such vocabularies. Since such a definition may be represented as a set of RDF triples, the terms introduced in a vocabulary should also be identified with HTTP URIs and vocabularies should also be published as Linked Data [18]. This practice enables a Linked-Data-aware software system to retrieve and utilize automatically the definition of terms used in the currently processed data.

The third Linked Data principle requires responding to a URI look-up request with a representation of RDF data that contains triples about the entity identified by the requested URI. However, the principles do not determine how exactly such RDF data should look like, what triples are necessary, or what vocabularies should be used. Nonetheless, a common practice is to provide RDF triples that contain the requested URI.

Even if the Linked Data principles do not prescribe what data should be provided in response to a URI look-up request, the fourth principle requires that the data includes links pointing to Linked Data from other data sources on the WWW. Such a *data link* is established by an RDF triple whose subject is a URI in the namespace of one data provider and whose object is another URI in the namespace of another provider. For instance, the second RDF triple in Example 1.1 establishes such a data link. These links are the most important feature of Linked Data because they form the basis for connecting structured data of different sources in a way similar to how human-readable Web documents have been interlinked for more than 25 years. Hence, based on these data links, the WWW evolves into a platform where self-describing data of any type can be posted, discovered, and integrated in an automated and standardized manner.

The particular set of RDF triples that a Web server returns in response to a URI look-up request, may exist as a precomputed RDF document stored on the server. Another typical approach is a Linked Data server that returns subsets of a larger set of RDF triples (which is usually stored in a database management system for RDF data). Other Linked Data servers may be implemented as wrappers over a relational database or over a Web API. There exist even Web servers that generate Linked Data on the fly as the following example illustrates.

Example 1.2. Assume a URI pattern that can be used to construct an HTTP URI http://example.org/number/i for every natural number $i \in \mathbb{N}$. The WWW server that is responsible for these URIs may be set up to return a specific set of RDF triples for *each* of these URIs; these sets may be generated *upon request*. For instance, the data generated in response to a request for URI http://example.org/number/42 may include the RDF triple

$$\big(\text{http://example.org/number/42}\,,\ \text{http://example.org/vocab\#next}\,,\ \text{http://example.org/number/43}\big)$$

that associates the requested natural number 42 with its successor 43. An example for such a server is provided by the Linked Open Numbers project [161]. The URI pattern for natural numbers as used by this server is: http://km.aifb.kit.edu/projects/numbers/web/ni. □

In addition to publishing data using the Linked Data principles, several publishers also provide a Web service for executing queries over their Linked Data. Usually, such a service supports the query language SPARQL [63] and may be accessed using the corresponding SPARQL protocol [46]. Therefore, such a service is called a *SPARQL endpoint*.

For the sake of conciseness, we have left out a number of technical details in this introduction. Most of these details relate to how exactly the HTTP protocol is used for publishing and consuming Linked Data on the WWW; those details are not important for this dissertation. For a comprehensive introduction to publishing Linked Data we refer to Heath and Bizer's recent book on the topic [83].

After the Linked Data principles had been proposed in 2006, a grass-roots movement started to publish and interlink multiple open databases on the WWW following these principles [22]. Since then, community initiatives and research groups, as well as enterprises and government initiatives, adopted the Linked Data principles, and publishing Linked Data has become a non-negligible trend on today's WWW [23, 114, 118].

Prominent publishers include the BBC [93, 155], the New York Times [103], the UK government [35], the Library of Congress [50], Best Buy [17], and Renault [144]. Available data covers diverse topics such as books [21], movies [80], music [132], radio and television programs [93], reviews [84], scientific publications [165], genes, proteins, medicine, clinical trials [136], geographic locations [10], people, companies, census data, etc.

The emergence of this global, interlinked data space—often referred to as the *"Web of Data"* [23, 83]—presents an interesting development; the possibility to query the Web of Data as if it were a huge distributed database holds an enormous potential: Data from a virtually unbounded set of data sources can be aggregated in a standardized manner; fragmentary information from a multitude of sources can be integrated to achieve a more complete view or to answer complex information needs in an automated fashion. In the following, we discuss options for implementing such a query processing functionality.

1.2. Approaches to Query Linked Data

Several general options for querying Linked Data exist. In the simplest case, an application may access the SPARQL endpoint provided by a particular data publisher. While such an access may already provide the application with valuable data, this approach ignores the great potential of the Web of Data; it does not exploit the possibilities of this huge data space that integrates a large number of interlinked datasets. The following example illustrates this limitation.

Example 1.3. Consider a query that asks for the phone number of people who authored a data integration related paper at the European Semantic Web Conference 2009 (ESWC'09). Figure 1.1 provides a SPARQL representation of this query (for a definition of the SPARQL query language we refer to Chapter 3). For instance, the URI http://data.semanticweb.org/conference/eswc/2009/proceedings, as used in line 9 in Figure 1.1, denotes the proceedings of ESWC'09.

This query cannot be answered from a single dataset but requires data from a diverse set of data sources on the WWW. For instance, the list of papers and their topics (as asked for in lines 9 to 11) are published as part of the Semantic Web Conference Corpus (online at `http://data.semanticweb.org`); the names of the paper topics (line 12) are provided by the data sources responsible for the URIs used to denote the topics; the phone numbers (line 18) are provided by the authors (e.g., in a FOAF document on their personal Web site [41]). □

The example introduces a query that can only be answered by executing queries over a (potentially virtual) union of Linked Data from multiple sources. The database literature focuses on two paradigms for querying distributed data provided by autonomous sources: data warehousing [33] and federated query processing [145]. Both of these paradigms can be used to query Linked Data provided by multiple publishers [78].

Data warehouse approaches are based on copying data into a centralized repository similar to collecting Web documents managed by search engines for the WWW. By using such a repository, it is possible to provide almost instant query results. This

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX swc:  <http://data.semanticweb.org/ns/swc/ontology#>
PREFIX swrc: <http://swrc.ontoware.org/ontology#>

SELECT DISTINCT ?author ?phone
WHERE {
    <http://data.semanticweb.org/conference/eswc/2009/proceedings>
                                                            swc:hasPart ?pub .
    ?pub   swc:hasTopic ?topic .
    ?topic rdfs:label   ?topicLabel .
    FILTER regex( str(?topicLabel), "Data␣integration", "i" ) .

    ?pub swrc:author ?author .
    {?author owl:sameAs ?authAlt} UNION {?authAlt owl:sameAs ?author}

    ?authAlt foaf:phone ?phone .
}
```

Figure 1.1.: SPARQL query that asks for phone numbers of people who authored a data integration related paper at the European Semantic Web Conference 2009.

capability comes at the cost of setting up and maintaining a centralized repository. Thus, query results may not reflect the most recent data and users may only benefit from the portion of the Web of Data that has been copied into the repository. For instance, if we aim to answer the query in Example 1.3 by using a repository that lacks, e.g., some authors' personal data (or the most recent version thereof), we may get an answer that is incomplete (or outdated) w.r.t. all Linked Data available on the WWW.

Federated query processing approaches distribute query execution over the SPARQL endpoints that publishers provide for their Linked Data. Building a federation system for a given set of SPARQL endpoints differs not much from work on relational federation systems [55]; a number of SPARQL federation systems have been presented in the literature (e.g., ANAPSID [5], Avalanche [11], DARQ [130], FedX [143], and SPLENDID [54]). The advantage of using such a system is that no copied data needs to be synchronized; instead, queries are always answered based on the original, up-to-date data. With version 1.1 of SPARQL, query federation even becomes a feature of the query language: The keyword SERVICE enables users to identify subqueries that have to be processed by remote SPARQL endpoints [129]. However, a particular downside of all SPARQL federation approaches is their limited coverage: We cannot assume that all publishers provide a SPARQL endpoint for their Linked Data. Providing and maintaining a *reliable* SPARQL endpoint presents a significant additional effort that not all publishers are willing or able to make [29]. In contrast, the Linked Data principles present a simple publishing method that can be easily added to existing workflows for generating HTML documents. Using the RDFa standard, Linked Data can even be embedded into HTML documents [6], allowing publishers to serve a single type of document for human and machine consumption. Therefore, it is more likely that people expose Linked Data on their personal Web site via dedicated RDF documents or embedded in HTML documents, rather than setting up a SPARQL endpoint (which renders a query execution approach that relies on such endpoints unsuitable for the query in Example 1.3). Consequently, querying the Web of Data as a federation of SPARQL endpoints may result in ignoring a large portion of Linked Data available.

Given the limitations that data warehousing and federated query processing have in exploiting the Web of Data to its full potential, special approaches for *Linked Data query processing* have been studied [74, 99, 115, 139, 159]. The goal of Linked Data query processing is an online execution of declarative queries over the Web of Data, *by relying only on the Linked Data principles*. That is, Linked Data query processing systems obtain Linked Data for answering a given query by looking up URIs during the query execution process itself. As a result, Linked Data query processing is likely to contribute to bringing the Web of Data to its full potential. However, we notice a shortage of work on fundamental aspects of this new paradigm. That is, existing work on Linked Data query processing focuses primarily on various, system-related aspects of (query-local) data management, query execution, and optimization. To fill this gap, this dissertation studies the foundations of Linked Data query processing.

1.3. Problem Statement

Querying the Web of Data is fundamentally different from querying a more traditional database whose elements (such as tables and indexes) are registered in a catalog and can be accessed without any limitations. The Web of Data is a virtually unbounded space; that is, looking up any randomly generated HTTP URI may result in retrieving Linked Data. Although these URIs are countable, there exists an infinite number of them. Therefore, we cannot assume the existence of a complete catalog of all URIs for retrieving all Linked Data (even if the Web of Data would be static). Consequently, we also cannot assume that any system ever has access to all Linked Data that is—or was—(openly) available on the WWW at a certain point in time.

While existing work on Linked Data query processing has studied practical approaches for dealing with these restrictions, a more fundamental discussion of the queries that may be executed based on such approaches is largely missing. More precisely, there do not exist provable criteria for identifying queries whose execution (over the Web of Data) may not terminate or is not feasible at all. Furthermore, for most of the Linked Data query processing approaches it is not even clear what exactly the expected result of executing a query is (because the corresponding publications lack a precise definition of supported queries and query semantics).

While the declarative query language SPARQL is the standard for querying a-priory defined sets of (RDF-based) Linked Data copied from the WWW, there does not exist a similarly declarative language for querying the Web of Data itself. Nonetheless, given that Linked Data is based on the RDF data model and SPARQL is a query language for RDF, it seems natural to assume that SPARQL could also be used for such a purpose (as we have done in Example 1.3). In fact, existing Linked Data query processing approaches use (a particular fragment of) SPARQL to denote the queries that they focus on. However, this approach is insufficient because the existing SPARQL semantics defines queries as functions over an *"RDF dataset"* [63], that is, a fixed, a-priory defined collection of sets of RDF triples; therefore, given that the Web of Data is not such an RDF dataset, the expected result for evaluating a SPARQL expression over the Web of Data remains undefined in most of the literature on Linked Data query processing.

We note that some publications exist that propose such a missing query semantics to use SPARQL for the Web of Data [26, 66, 160] (for a detailed discussion of these proposals refer to Section 3.1.3, page 35ff). However, these proposals are limited to conjunctive queries and they lack an analysis of the computational feasibility (or other properties) of queries under the proposed semantics (let alone a proof showing that a Linked Data query processing approach correctly supports such a query semantics).

1.4. Contributions

The primary goal of this dissertation is to formally establish fundamental properties and limitations of Linked Data query processing. To this end, we address the aforementioned problems by making the following main contributions:

1. We **establish well-defined foundations** for Linked Data query processing (assuming SPARQL as our query language). In particular, these foundations include:
   a) a **data model** that formally represents the Web of Data,
   b) a **computation model** that captures the capabilities of systems whose access to data sources relies only the Linked Data principles, and
   c) multiple, well-defined **query semantics** for using the complete core fragment of SPARQL as a query language for Linked Data on the WWW.
2. We **study computational feasibility** and related properties of queries under the proposed query semantics.
3. We provide a comprehensive overview and **classification of query execution techniques** that are used in existing query processing approaches for Linked Data.
4. We **show soundness and completeness** of a *"traversal-based query execution"* approach [75], which is among the most prevalent query processing approaches for Linked Data.
5. We **investigate the suitability of iterators** for implementing traversal-based query execution.

In the following we provide a more detailed description of our contributions.

Formal Framework

The basis of this dissertation is a formal framework that enables us to define query semantics of queries over Linked Data and to study the computational feasibility of queries under such query semantics. Consequently, our formal framework consists of a data model and a computation model.

The data model formalizes the idea of Linked Data by introducing an abstract structure called the *Web of Linked Data*. We emphasize that such a Web may be infinitely large in our data model; allowing for an infinite Web of Linked Data enables us to capture the existence of Web servers that are able to respond to an infinite number of URIs by generating Linked Data on the fly (as illustrated in Example 1.2, page 3). Our data model also introduces the concept of a *Linked Data query*. This concept formalizes the notion of queries over Linked Data.

In addition to the data model, our formal framework comprises a computation model. This model allows us to formally identify those Linked Data queries for which a complete computation is feasible. To this end, the model captures the capabilities of systems that aim to make use of data available as per the Linked Data principles (e.g., for query computation or for answering decision problems about Linked Data on the WWW).

We emphasize that our data model and our computation model are independent of any particular query language (such as SPARQL) or query processing approach (such as traversal-based query execution). Hence, these models present a basis not only for the work in this dissertation but also for future work related to the foundations of query processing for Linked Data.

Query Semantics

This dissertation focuses on Linked Data queries that are represented using the complete core fragment of the RDF query language SPARQL (which includes conjunctions, disjunctions, optional parts, and filter constraints over values bound to query variables). To use SPARQL in our context, we have to adjust the semantics of SPARQL expressions. More precisely, we redefine the scope for evaluating SPARQL expressions. In this dissertation we propose (and study) two approaches for such an adjustment. The first approach uses a query semantics where the scope of a query is the complete set of Linked Data on the Web. We call this semantics *full-Web semantics*. The second approach introduces a family of *reachability-based semantics* which restrict the scope to data that is reachable by traversing a well-defined set of data links.

Formal Analysis of SPARQL-Based Linked Data Queries

By introducing the aforementioned query semantics, we contribute a well-defined foundation for Linked Data query processing. However, instead of merely defining query semantics, we aim to understand the consequences of using these semantics. Therefore, we formally analyze several theoretical properties of queries under these semantics: Most importantly, we study the computational feasibility of such queries and the feasibility of deciding whether a query execution terminates. For this analysis we use our computation model. The perhaps most important result of this analysis is a *formal verification* of the common, as yet unverified assumption that *"processing queries against Linked Data where sources have to be discovered online might not yield all results"* [99]. However, we also identify the cases in which—at least in theory—an expected query result may be computed completely by an execution that is guaranteed to terminate.

Furthermore, we study basic properties such as satisfiability and monotonicity, and we discuss the implications of querying an infinite Web of Linked Data. For the reachability-based query semantics, we also discuss the relationships between different notions of reachability and the impact of these notions on queries and their properties. Finally, we identify commonalities and differences between the different query semantics.

Classification of Possible Query Execution Techniques

Multiple approaches to process Linked Data queries have been presented in the literature. Each of these approaches introduces a number of (complementary) query execution techniques. Some of the techniques proposed for different approaches implement the same abstract idea and, thus, are conceptually similar; other techniques are very different from each other (or serve different purposes).

There does not exist a systematic overview on the state of the art in executing Linked Data queries that reviews all of these techniques separate from discussing the particular approaches in whose context they are introduced. To fill this gap, we introduce a classification that categorizes possible query execution techniques along three orthogonal dimensions: (i) data source selection, (ii) data source ranking, and (iii) integration of data retrieval and result construction. For each of the dimensions, we provide a comprehensive conceptual comparison of the techniques in that dimension.

Formal Analysis of Traversal-Based Execution

One of the most prevalent approaches to execute Linked Data queries is *"traversal-based query execution"* [75], which takes advantage of the characteristics of the Web of Data. The fundamental approach is to intertwine the traversal of data links with the construction of the query result thus integrating the discovery of data into the query execution process. Hence, in contrast to more traditional query execution approaches, this approach *does not assume a fixed set of relevant data sources beforehand*; instead, it uses data from initially unknown data sources for answering queries and, therefore, enables applications to tap the full potential of the WWW.

While different implementations of the idea of traversal-based query execution have been published [79, 99, 115], we are interested in whether the general approach is sound and complete w.r.t. the query semantics that we introduce. Therefore, we define an abstract query execution model that formalizes the idea of traversal-based query execution; i.e., this model captures the approach of intertwining link traversal and result construction independent from particular implementation techniques. Based on this model we prove the soundness and completeness of the new query execution paradigm.

Analysis of Link Traversing Iterators

As mentioned before, the idea of traversal-based query execution may be implemented using different techniques. In this dissertation we aim to achieve an understanding of the suitability of classical database techniques for such a purpose. In particular, we focus on the well-known iterator model [56].

We define a *link traversing iterator*. The main feature of this iterator is that calling its `GetNext` function (to obtain intermediate results for computing a query) has the desired side effect of a dynamic, traversal-based retrieval of Linked Data. Hence, a pipeline of such iterators continuously augments the query-local dataset over which it operates.

We prove that such a pipeline presents a sound implementation of our abstract model of traversal-based query execution. However, this implementation approach cannot guarantee completeness of computed query results. In a formal and experimental analysis we study this limitation, as well as other properties of the implementation approach.

Technical Contributions

In addition to the aforementioned research contributions, we developed a complete query processing system for Linked Data queries during our work on this dissertation. This system, called *SQUIN*, is implemented in Java, and it consists of more than 10K lines of native source code (which is available as Free Software on the SQUIN project homepage at `http://squin.org`). The query engine in SQUIN performs a traversal-based execution of Linked Data queries and has been implemented based on the iterator approach as studied in this dissertation.

By using SQUIN, the aforementioned example query (cf. Example 1.3 on page 4) can be executed *live* on the WWW. On September 16, 2013, such an execution resulted in obtaining the phone numbers of two relevant authors. During this execution, SQUIN discovered (and used) Linked Data from 14 different Web sites.

1.5. Thesis Outline

This dissertation consists of two main parts. The first part focuses on the theoretical foundations and fundamental properties of SPARQL-based Linked Data queries. This part is divided into three chapters:

- Chapter 2 introduces the formal framework for this dissertation, that is, our data model and our computation model.
- Chapter 3 defines SPARQL and the full-Web semantics for SPARQL; thereafter, the chapter presents our analysis of queries under this full-Web semantics. Moreover, the chapter also reviews related work on query languages and query semantics for Linked Data queries and for accessing the WWW in general.
- Chapter 4 defines the family of reachability-based query semantics for SPARQL and provides an analysis of queries under these semantics (analogous to the analysis

of full-Web semantics in the previous chapter). Furthermore, the chapter discusses different notions of reachability, and it compares the full-Web semantics with the reachability-based query semantics.

The second part of this dissertation focuses on topics related to the execution of Linked Data queries. This part also consists of three chapters:

- Chapter 5 provides a comprehensive, systematic review of query execution techniques for Linked Data queries.
- Chapter 6 defines an implementation-independent execution model of traversal-based query execution and shows the soundness and completeness of this model.
- Chapter 7 introduces the iterator approach for implementing traversal-based query execution and analyzes this implementation approach formally and experimentally.

Finally, Chapter 8 summarizes the results of this dissertation and outlines directions for future research.

The formalizations in this dissertation introduce a number of symbols. We emphasize that, for the convenience of the reader, Appendix A lists these symbols and refers to the corresponding definitions.

Part I.

Foundations of Queries over a Web of Linked Data

2. Models for Linked Data Queries

This chapter introduces formal models for defining and analyzing queries over data that is accessible on the WWW based on the Linked Data principles. In particular, these models are a data model and a computation model.

The data model formalizes the idea of a Web of Linked Data and a notion of queries over such a Web. The main purpose of this model is to provide a basis for introducing well-defined query semantics in Chapters 3 and 4. However, we also use the data model as a basis for developing a query execution model that formally captures a particular approach to execute queries over a Web of Linked Data (cf. Chapter 6).

The concept of a Web of Linked Data, as formalized by our data model, presents the notion of a database in our work. We assume that this database is distributed over the WWW, data access is limited to URI lookups, and there does not exist a complete catalog of all URIs whose lookup results in the retrieval of some data. These limitations have an impact on the capabilities of systems that aim to compute queries over Linked Data on the WWW. The computation model that we introduce in this chapter captures these capabilities formally and, thus, allows us to classify queries w.r.t. whether Linked Data query execution systems for the WWW can compute them.

For the models presented in this chapter, we assume a static view of the Web. More precisely, we assume that no changes are made to the data on the Web during the computation of a query (or any other type of computation). Such a static view also ignores temporarily unavailable data and the possibility of timeouts during the retrieval of data.

This chapter is organized as follows: First, Section 2.1 introduces our data model. Second, Section 2.2 specifies our computation model.

2.1. Data Model

In this section we first define the structural elements of our data model. Thereafter, we augment the model by introducing the notion of a Linked Data query. Finally, we review related work on modeling the WWW and Linked Data on the WWW.

2.1.1. Structural Elements

Berners-Lee's Linked Data principles prescribe RDF as a common data model for representing Linked Data published on the WWW [14]. Therefore, we use the RDF data model [92] as a basis for our model of a Web of Linked Data. That is, we assume three pairwise disjoint, countably infinite sets $\mathcal{U}$ (URIs), $\mathcal{B}$ (blank nodes), $\mathcal{L}$ (literals). An *RDF triple* is a tuple $(s, p, o) \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$, where s, p, and o are called the *subject*, *predicate*, and *object* of that triple, respectively. For any RDF triple $t = (s, p, o)$

we define $\mathrm{terms}(t) := \{s, p, o\}$ and $\mathrm{uris}(t) := \mathrm{terms}(t) \cap \mathcal{U}$. Overloading function terms, we define $\mathrm{terms}(G) := \bigcup_{t \in G} \mathrm{terms}(t)$ for any set G of RDF triples.

Given these preliminaries we are ready to define a *Web of Linked Data*: Assume a countably infinite set $\mathcal{D}$ that is disjoint from $\mathcal{U}$, $\mathcal{B}$, and $\mathcal{L}$, respectively. We refer to elements in this set as Linked Data documents, or *LD document*s for short, and use them to represent the concept of Web documents from which Linked Data can be extracted. Then, a Web of Linked Data is a potentially infinite structure of interlinked LD documents. In such a Web, LD documents are accessed via URIs and contain a set of RDF triples. The following definition captures our approach:

Definition 2.1 (Web of Linked Data). Let $\mathcal{T} = (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ be the infinite set of all possible RDF triples and let $\perp$ be a special symbol that denotes the *nonexistent LD document* ($\perp \notin \mathcal{D}$). A *Web of Linked Data* is a tuple $W = (D, data, adoc)$ with the following three elements:

- D is a finite or countably infinite set of LD documents; i.e., $D \subset \mathcal{D}$.
- $data$ is a total mapping $data \colon D \to 2^{\mathcal{T}}$ such that (i) $data(d)$ is finite for all LD documents $d \in D$, and (ii) for each pair $(d_1, d_2) \in D \times D$ of distinct LD documents $d_1 \neq d_2$, $\mathrm{terms}\big(data(d_1)\big) \cap \mathrm{terms}\big(data(d_2)\big) \cap \mathcal{B} = \emptyset$ holds.
- $adoc$ is a total mapping $adoc \colon \mathcal{U} \to D \cup \{\perp\}$ such that for each LD document $d \in D$ there exists a URI $u \in \mathcal{U}$ for which $adoc(u) = d$. □

In the following, we discuss the rationale and properties of the three elements that define a Web of Linked Data in our model (i.e., D, $data$, and $adoc$), and we introduce additional, related concepts that we need in this dissertation.

Elements of a Web of Linked Data

We say that a Web of Linked Data $W = (D, data, adoc)$ is *infinite* if its set of LD documents D is infinite; otherwise, we say W is *finite*. Our model allows for infinite Webs to cover the possibility that Linked Data about an infinite number of identifiable entities is generated on the fly. As an example for such a case recall the Web server that provides Linked Data for all natural numbers as discussed in Example 1.2 (cf. page 3). Another example for an infinite number of entities is the LinkedGeoData project; this project provides Linked Data about any circular and rectangular area on Earth [10]. These two examples illustrate that—even if we assume a static view—an infinite Web of Linked Data is possible in practice because of the existence of data generating servers. Covering these cases in our model enables us to define queries over such data and analyze the effects of executing those queries.

Even if a Web of Linked Data is infinite, we require countability for its set of LD documents. We shall see that this requirement has nontrivial consequences: It limits the potential size of Webs of Linked Data in our model and, thus, allows us to encode such a Web on the tape of a Turing machine (cf. Section 2.2.2, page 27ff). We emphasize that the requirement of countability does not restrict us in modeling the WWW as a

Web of Linked Data: In the WWW, HTTP-scheme-based URIs [47] are used to locate documents that contain Linked Data. Even if those URIs are not limited in length, they are words over a finite alphabet. Thus, the infinite set of all possible HTTP URIs is countable, and so is the set of all documents that may be retrieved using such URIs.

The mapping *data* associates each LD document $d \in D$ in a Web of Linked Data $W = (D, data, adoc)$ with a finite set of RDF triples. In practice, these triples are obtained by parsing a Web document after it has been retrieved from the WWW. The actual syntax for representing RDF triples in Web documents is not relevant for our model. However, as prescribed by the RDF data model [92], Definition 2.1 requires that the data of each LD document uses a unique set of blank nodes.

To denote the (potentially infinite but countable) set of *all RDF triples* in a Web of Linked Data $W = (D, data, adoc)$ we write $\mathsf{AllData}(W)$. More precisely, we define:

$$\mathsf{AllData}(W) := \bigcup_{d \in D} data(d).$$

According to the Linked Data principles [14], a URI does not only identify an entity, but it also serves as a reference to a particular document that contains data about that entity. Our model captures this relationship between URIs and documents by defining the mapping *adoc*. LD document $adoc(u) \in D$ may be considered as an authoritative source of data for URI u (hence the name *adoc*). To allow for LD documents that are authoritative for multiple URIs, we do not require injectivity for mapping *adoc*.

However, we require totality for mapping *adoc*, which shall allow us to model a notion of partial knowledge about a queried Web of Linked Data when we discuss our query execution model in Chapter 6 (see in particular Definition 6.4, page 118). Given that mapping *adoc* is total, we need the concept of a *nonexistent LD document*, denoted by $\perp$, in order to accommodate for URIs for which no authoritative LD document exists (in a given Web of Linked Data). Hence, *adoc* maps these URIs to $\perp$.

Graph Structure of a Web of Linked Data

Our data model does not make any assumption about the relationship between a URI u for which there exists an authoritative LD document $d = adoc(u) \in D$ in a Web of Linked Data $W = (D, data, adoc)$ and the data in this document. Nonetheless, as encouraged by the Linked Data principles, it is common practice that the URI occurs in this data; i.e., there exists an RDF triple $t \in data(d)$ such that $u \in \mathrm{uris}(t)$. Clearly, other URIs may occur in this data as well. Then, the occurrence of a URI u with $adoc(u) \neq \perp$ in the data of some LD document establishes a *data link* from that document to the authoritative LD document for the URI. These data links form the following graph structure.

Definition 2.2 (Link Graph). Given a Web of Linked Data $W = (D, data, adoc)$, the *link graph* of W is a directed graph (D, E) whose vertices are all LD documents in W, and which has an edge from LD document d_i to LD document d_j if there exists a data link from d_i to d_j; i.e., E is defined as follows:

$$E := \Big\{ (d_i, d_j) \in D \times D \,\Big|\, t \in data(d_i) \text{ and } u \in \mathrm{uris}(t) \text{ and } adoc(u) = d_j \Big\} \qquad \square$$

We note that the link graph of an *infinite* Web of Linked Data is infinite (i.e., it has an infinite number of vertices and it may also have an infinite number of edges); the link graph of a finite Web of Linked Data is finite. Furthermore, link graphs are not necessarily strongly connected; they do not even have to be weakly connected. Moreover, given that URIs usually occur in the data of their authoritative LD document (see above), link graphs may have loops (i.e., edges that connect vertices to themselves).

Example 2.1. Let $W_{\mathsf{ex}} = (D_{\mathsf{ex}}, data_{\mathsf{ex}}, adoc_{\mathsf{ex}})$ be a Web of Linked Data in which two producers and two vendors publish (and interlink) data about themselves, their products, and their offers. We use this Web as a running example throughout this dissertation.

W_{ex} consists of ten LD documents: $D_{\mathsf{ex}} = \{d_{\mathsf{Pr1}}, d_{\mathsf{Pr2}}, d_{\mathsf{V1}}, d_{\mathsf{V2}}, d_{\mathsf{p1}}, d_{\mathsf{p2}}, d_{\mathsf{p3}}, d_{\mathsf{p4}}, d_{\mathsf{off1.1}}, d_{\mathsf{off1.2}}, d_{\mathsf{off2.1}}\}$. The sets of RDF triples in these documents are given as shown in Figure 2.1 in which double quote delimited strings denote literals (e.g., $\mathsf{"Producer\ 1"} \in \mathcal{L}$) and any other element in these triples is a URI. For any URI $u \in \mathcal{U}$, mapping $adoc_{\mathsf{ex}}$ is given as follows:

$$adoc_{\mathsf{ex}}(u) = \begin{cases} d_{\mathsf{Pr1}} & \text{if } u = \mathsf{producer1}, \\ d_{\mathsf{Pr2}} & \text{if } u = \mathsf{producer2}, \\ d_{\mathsf{V1}} & \text{if } u = \mathsf{vendor1}, \\ d_{\mathsf{V2}} & \text{if } u = \mathsf{vendor2}, \\ d_{\mathsf{p1}} & \text{if } u = \mathsf{product1}, \\ d_{\mathsf{p2}} & \text{if } u = \mathsf{product2}, \\ d_{\mathsf{p3}} & \text{if } u = \mathsf{product3}, \\ d_{\mathsf{p4}} & \text{if } u = \mathsf{product4}, \\ d_{\mathsf{off1.1}} & \text{if } u = \mathsf{offer1.1}, \\ d_{\mathsf{off1.2}} & \text{if } u = \mathsf{offer1.2}, \\ d_{\mathsf{off2.1}} & \text{if } u = \mathsf{offer2.1}, \\ \perp & \text{else.} \end{cases}$$

Then, Figure 2.2 illustrates the link graph of W_{ex} (cf. page 20). □

Subwebs of a Web of Linked Data

To study the monotonicity of queries over a Web of Linked Data we require a notion of containment for such Webs. For this purpose, we define the concept of a subweb.

Definition 2.3 (Subweb). Let $W = (D, data, adoc)$ and $W' = (D', data', adoc')$ be Webs of Linked Data. W' is a *subweb* of W if the following four properties hold:

1. $D' \subseteq D$,
2. For each LD document $d \in D'$, $data'(d) = data(d)$.
3. For each URI $u \in \mathcal{U}$, if $adoc(u) \in D'$, then $adoc'(u) = adoc(u)$ or $adoc'(u) = \perp$.
4. For each URI $u \in \mathcal{U}$, if $adoc(u) \notin D'$, then $adoc'(u) = \perp$. □

$$\begin{aligned} data_{\mathsf{ex}}(d_{\mathsf{Pr1}}) = \{ & (\mathsf{producer1}, \mathsf{name}, \text{"Producer 1"}), \\ & (\mathsf{product2}, \mathsf{producedBy}, \mathsf{producer1}), \\ & (\mathsf{product3}, \mathsf{producedBy}, \mathsf{producer1}) \} \end{aligned}$$

$$\begin{aligned} data_{\mathsf{ex}}(d_{\mathsf{p1}}) = \{ & (\mathsf{product1}, \mathsf{name}, \text{"Product 1"}), \\ & (\mathsf{product1}, \mathsf{oldVersionOf}, \mathsf{product2}), \\ & (\mathsf{product1}, \mathsf{oldVersionOf}, \mathsf{product3}) \} \end{aligned}$$

$$\begin{aligned} data_{\mathsf{ex}}(d_{\mathsf{p2}}) = \{ & (\mathsf{product2}, \mathsf{name}, \text{"Product 2"}), \\ & (\mathsf{product2}, \mathsf{producedBy}, \mathsf{producer1}), \\ & (\mathsf{product1}, \mathsf{oldVersionOf}, \mathsf{product2}) \} \end{aligned}$$

$$\begin{aligned} data_{\mathsf{ex}}(d_{\mathsf{p3}}) = \{ & (\mathsf{product3}, \mathsf{name}, \text{"Product 3"}), \\ & (\mathsf{product3}, \mathsf{producedBy}, \mathsf{producer1}) \} \end{aligned}$$

$$\begin{aligned} data_{\mathsf{ex}}(d_{\mathsf{Pr2}}) = \{ & (\mathsf{producer2}, \mathsf{name}, \text{"Producer 2"}), \\ & (\mathsf{product4}, \mathsf{producedBy}, \mathsf{producer2}) \} \end{aligned}$$

$$\begin{aligned} data_{\mathsf{ex}}(d_{\mathsf{p4}}) = \{ & (\mathsf{product4}, \mathsf{name}, \text{"Product 4"}), \\ & (\mathsf{product4}, \mathsf{producedBy}, \mathsf{producer2}) \} \end{aligned}$$

$$\begin{aligned} data_{\mathsf{ex}}(d_{\mathsf{V1}}) = \{ & (\mathsf{vendor1}, \mathsf{name}, \text{"Vendor 1"}), \\ & (\mathsf{offer1.1}, \mathsf{offeredBy}, \mathsf{vendor1}), \\ & (\mathsf{offer1.2}, \mathsf{offeredBy}, \mathsf{vendor1}) \} \end{aligned}$$

$$\begin{aligned} data_{\mathsf{ex}}(d_{\mathsf{off1.1}}) = \{ & (\mathsf{offer1.1}, \mathsf{offeredBy}, \mathsf{vendor1}), \\ & (\mathsf{offer1.1}, \mathsf{price}, 10), \\ & (\mathsf{offer1.1}, \mathsf{offeredProduct}, \mathsf{product2}) \} \end{aligned}$$

$$\begin{aligned} data_{\mathsf{ex}}(d_{\mathsf{off1.2}}) = \{ & (\mathsf{offer1.2}, \mathsf{offeredBy}, \mathsf{vendor1}), \\ & (\mathsf{offer1.2}, \mathsf{price}, 6), \\ & (\mathsf{offer1.2}, \mathsf{offeredProduct}, \mathsf{product3}) \} \end{aligned}$$

$$\begin{aligned} data_{\mathsf{ex}}(d_{\mathsf{V2}}) = \{ & (\mathsf{vendor2}, \mathsf{name}, \text{"Vendor 2"}), \\ & (\mathsf{offer2.1}, \mathsf{offeredBy}, \mathsf{vendor2}) \} \end{aligned}$$

$$\begin{aligned} data_{\mathsf{ex}}(d_{\mathsf{off2.1}}) = \{ & (\mathsf{offer2.1}, \mathsf{offeredBy}, \mathsf{vendor2}), \\ & (\mathsf{offer2.1}, \mathsf{price}, 11), \\ & (\mathsf{offer2.1}, \mathsf{offeredProduct}, \mathsf{product2}) \} \end{aligned}$$

Figure 2.1.: The data in our example Web W_{ex} (cf. Example 2.1, page 18).

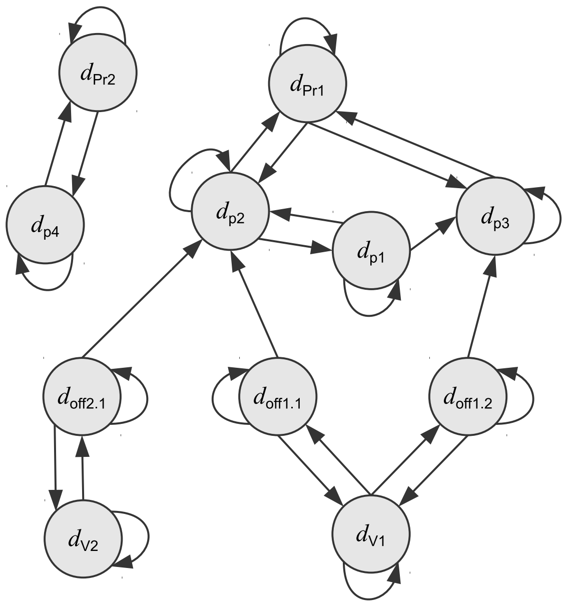


Figure 2.2.: Link graph of the example Web W_{ex}.

As can be seen from Definition 2.3, we require that any LD document in subweb W' is also contained in the parent Web W and has the same data as in W (Properties 1 and 2). Furthermore, for URIs whose authoritative LD document is contained in the parent Web and in the subweb, the relationship between URI and authoritative LD document may also be available in the subweb; however, the latter is not a must (Property 3). Finally, if a URI has an authoritative LD document in the parent Web that is not contained in the subweb, the URI must not have any authoritative LD document in the subweb (Property 4). Due to this definition our notion of a subweb resembles the well-known concept of a subgraph in graph theory. In fact, any subweb relation becomes a subgraph relation when we consider the link graphs for a Web of Linked Data and its subweb. That is, it is easily verified that the link graph for a subweb is a subgraph of the link graph for the corresponding parent Web.

Example 2.2. A possible subweb of the Web of Linked Data $W_{\mathsf{ex}} = (D_{\mathsf{ex}}, data_{\mathsf{ex}}, adoc_{\mathsf{ex}})$ given in Example 2.1 (cf. page 18) is the Web of Linked Data $W'_{\mathsf{ex}} = (D'_{\mathsf{ex}}, data'_{\mathsf{ex}}, adoc'_{\mathsf{ex}})$ with (i) $D'_{\mathsf{ex}} = \{d_{\mathsf{p2}}, d_{\mathsf{p3}}, d_{\mathsf{off1.1}}, d_{\mathsf{off1.2}}, \mathsf{offer2.1}\} \subset D_{\mathsf{ex}}$, (ii) $data'_{\mathsf{ex}}(d') = data_{\mathsf{ex}}(d')$ for all LD documents $d' \in D'_{\mathsf{ex}}$, and (iii) for any URI $u \in \mathcal{U}$, mapping $adoc'_{\mathsf{ex}}$ is given as follows:

$$adoc'_{\mathsf{ex}}(u) = \begin{cases} d_{\mathsf{p2}} & \text{if } u = \mathsf{product2}, \\ d_{\mathsf{p3}} & \text{if } u = \mathsf{product3}, \\ d_{\mathsf{off1.1}} & \text{if } u = \mathsf{offer1.1}, \\ d_{\mathsf{off1.2}} & \text{if } u = \mathsf{offer1.2}, \\ d_{\mathsf{off2.1}} & \text{if } u = \mathsf{offer2.1}, \\ \perp & \text{else.} \end{cases}$$

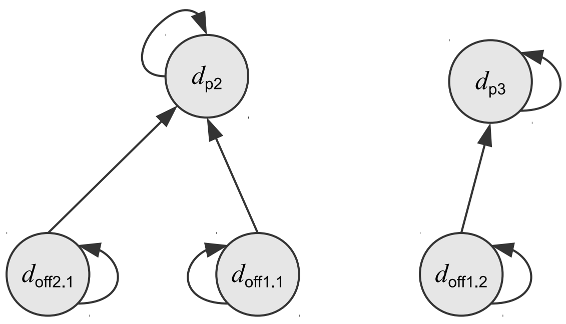


Figure 2.3.: Link graph of the subweb W'_{ex} given in Example 2.2.

Figure 2.3 illustrates the link graph of this subweb. □

In the context of defining reachability-based query semantics in Chapter 4, we shall define the concept of a reachable subweb. This definition is based on a particular class of subwebs that resemble the graph theoretic notion of an induced subgraph and, thus, are called *induced subwebs*.

Definition 2.4 (Induced Subweb). Let $W = (D, data, adoc)$ be a Web of Linked Data and let $W' = (D', data', adoc')$ be a subweb of W. W' is an *induced subweb* of W if, for each URI $u \in \mathcal{U}$ with $adoc(u) \in D'$, $adoc'(u) = adoc(u)$ holds. □

The condition in Definition 2.4 is a more strict version of Property 3 in Definition 2.3. We remark that the link graph of an induced subweb is an *induced* subgraph of the link graph of the corresponding parent Web (we omit a formal proof of this statement because this statement is not required for the results in this dissertation).

The following proposition establishes several properties of subwebs and induced subwebs that we shall use throughout this dissertation:

Proposition 2.1. *Let $W = (D, data, adoc)$ be a Web of Linked Data.*

1. *For any subweb W' of W, $\mathsf{AllData}(W') \subseteq \mathsf{AllData}(W)$.*
2. *Any subweb $W' = (D', data', adoc')$ of W is specified unambiguously by defining the set of LD documents D' and mapping $adoc'$.*
3. *Any induced subweb $W' = (D', data', adoc')$ of W is specified unambiguously by defining the set of LD documents D'.*

Proof. *1.:* For any subweb $W' = (D', data', adoc')$ of W, $D' \subseteq D$ holds by Definition 2.3 (cf. page 18). Then, we show $\mathsf{AllData}(W') \subseteq \mathsf{AllData}(W)$ by using:

$$\mathsf{AllData}(W') = \bigcup_{d \in D'} data(d) \qquad \text{and} \qquad \mathsf{AllData}(W) = \bigcup_{d \in D} data(d).$$

2.: Let $W' = (D', data', adoc')$ be an arbitrary subweb of W. To prove that W' is specified unambiguously by defining D' and $adoc'$ we have to show that there does not exist another subweb of W that has both the same D' and the same *adoc*′. We show this by contradiction: Assume there exists another subweb $W'' = (D'', \mathit{data}'', \mathit{adoc}'')$ of W such that $D'' = D'$, $\mathit{adoc}'' = \mathit{adoc}'$, and $\mathit{data}'' \neq \mathit{data}'$. Since both W' and W'' are a Web of Linked Data, by Definition 2.1 (cf. page 16), data' and data'' are total mappings from the set D' and D'', respectively; i.e., $\mathrm{dom}(\mathit{data}') = D'$ and $\mathrm{dom}(\mathit{data}'') = D''$. Since both subwebs have the same set of LD documents ($D'' = D'$), it holds that $\mathrm{dom}(\mathit{data}') = \mathrm{dom}(\mathit{data}'')$. Furthermore, from Property 2 in Definition 2.3, we have:

$$\forall\, d \in D' : \mathit{data}'(d) = \mathit{data}(d) \qquad \text{and} \qquad \forall\, d \in D'' : \mathit{data}''(d) = \mathit{data}(d).$$

Thus, given $D'' = D'$, it follows that mappings data'' and data' are equivalent. Since this equivalence contradicts our assumption, we conclude that defining D' and adoc' is sufficient to specify subweb W' unambiguously (given its parent Web W).

3.: Let $W' = (D', \mathit{data}', \mathit{adoc}')$ be an arbitrary induced subweb of W. To prove that W' is specified unambiguously by defining D' we, again, use a proof by contradiction. That is, we assume there exists another induced subweb $W'' = (D'', \mathit{data}'', \mathit{adoc}'')$ of W such that $D'' = D'$ and $\mathit{data}'' \neq \mathit{data}'$ or $\mathit{adoc}'' \neq \mathit{adoc}'$. By Definition 2.4 and by Property 4 in Definition 2.3, we have:

$$\begin{aligned}\forall\, u \in \mathcal{U} : \mathit{adoc}(u) \in D' &\Rightarrow \mathit{adoc}'(u) = \mathit{adoc}(u),\\ \forall\, u \in \mathcal{U} : \mathit{adoc}(u) \notin D' &\Rightarrow \mathit{adoc}'(u) = \bot,\end{aligned}$$

as well as:

$$\begin{aligned}\forall\, u \in \mathcal{U} : \mathit{adoc}(u) \in D'' &\Rightarrow \mathit{adoc}''(u) = \mathit{adoc}(u),\\ \forall\, u \in \mathcal{U} : \mathit{adoc}(u) \notin D'' &\Rightarrow \mathit{adoc}''(u) = \bot\,.\end{aligned}$$

Thus, given $D'' = D'$, it follows that mappings adoc'' and adoc' are equivalent. Therefore, based on our assumption, mappings data'' and data' cannot be equivalent. However, by using the same argument as in the more general case of subwebs, we may show that mappings data'' and data' are equivalent. Thus, our assumption cannot hold. ■

Example 2.3. Example 2.2 introduces a subweb $W'_{\mathsf{ex}} = (D'_{\mathsf{ex}}, \mathit{data}'_{\mathsf{ex}}, \mathit{adoc}'_{\mathsf{ex}})$ of our example Web of Linked Data W_{ex} (cf. page 20). This subweb is even an induced subweb of W_{ex}. Thus, its set of LD documents D'_{ex} specifies W'_{ex} unambiguously. □

2.1.2. Queries

In addition to the structural part, our data model introduces an abstract notion of queries over a Web of Linked Data. We aim to define this notion without imposing any particular query formalism (including query languages, query semantics, etc.) because we understand our data model as a general framework for defining and studying queries over Webs of Linked Data. Therefore, in the following definition we deliberately leave open query formalism specific aspects.

Definition 2.5 (Linked Data Query). Let $\mathcal{W}_{\mathsf{All}}$ be the infinite set of all possible Webs of Linked Data (that is, all 3-tuples that satisfy Definition 2.1) and let $\mathcal{R}(F)$ be a possibly infinite but countable set of all possible elements of query results specific to some query formalism F. A *Linked Data query* Q, specified using query formalism F, is a total function $Q\colon \mathcal{W}_{\mathsf{All}} \rightarrow 2^{\mathcal{R}(F)}$. □

As can be seen from Definition 2.5, every Linked Data query maps a queried Web of Linked Data to some subset of the set of all possible (query formalism specific) result elements. For instance, for the Linked Data queries discussed in the following chapters, these result elements are valuations that provide bindings for query variables (a formal definition of these valuations follows in Section 3.2.1, page 38ff). However, for the following, general discussion of Linked Data queries (that is, before we introduce concrete notions of Linked Data queries), we assume an arbitrary set $\mathcal{R}(F)$ (as per Definition 2.5). While such a (query formalism specific) set of possible result elements may be infinite, we assume countability. This limitation is necessary to allow us to analyze computability of Linked Data queries using an abstract machine model (for which encoding result elements requires countability; cf. Section 2.2.2, page 27ff).

Hereafter, for any Linked Data query Q and any Web of Linked Data W, we refer to the particular subset $Q(W) \subseteq \mathcal{R}(F)$ that is expected according to the query semantics used for Q as the *expected query result* for Q in W. Each element $\mu \in Q(W)$ in this expected query result is a *query solution* (or simply a *solution*) for Q in W. Furthermore, we use the term *computed query result* to refer to the set of result elements that a particular query execution process reports as a result of executing a Linked Data query over a Web of Linked Data. For query execution approaches that are sound and complete (w.r.t. the query semantics used) it holds that every computed query result is equivalent to the corresponding expected query result; approaches that are sound but not complete guarantee computed query results that are subsets of the expected query results.

While our definition of Linked Data queries assumes that all query results are sets, we emphasize that this definition does not rule out boolean queries or queries under some bag semantics: Boolean queries may be simulated by interpreting an empty (expected) query result as false and a nonempty result as true. To accommodate bag semantics, solutions may be augmented with some notion of identity.

Given the concept of subwebs, we define monotonicity of Linked Data queries.

Definition 2.6 (Monotonicity). A Linked Data query Q is *monotonic* if the following statement holds for any pair of Webs of Linked Data W_1 and W_2: If W_1 is a subweb of W_2, then $Q(W_1) \subseteq Q(W_2)$. A Linked Data query is *non-monotonic* if it is not monotonic. □

Similarly, the satisfiability property carries over naturally to Linked Data queries.

Definition 2.7 (Satisfiability). A Linked Data query Q is *satisfiable* if there exists a Web of Linked Data W such that $Q(W)$ is not empty. A Linked Data query is *unsatisfiable* if it is not satisfiable. □

As usual, non-monotonicity entails satisfiability:

Proposition 2.2. *Any non-monotonic Linked Data query is satisfiable.*

Proof. W.l.o.g., let Q be an arbitrary non-monotonic Linked Data query. Then, by Definition 2.6, there exists a pair of Webs of Linked Data W_1 and W_2 such that W_1 is a subweb of W_2 and $Q(W_1) \nsubseteq Q(W_2)$. Given such a pair, from $Q(W_1) \nsubseteq Q(W_2)$ it follows that there exists a solution for query Q in Web W_1 that is not a solution for Q in Web W_2. Let μ be such a solution. Hence, $\mu \in Q(W_1)$. Therefore, $Q(W_1)$ is not empty and, thus, Linked Data query Q is satisfiable. ■

For some of the discussions in this dissertation we need the following, more restrictive notion of satisfiability.

Definition 2.8 (Bounded and Unbounded Satisfiability). A Linked Data query Q is *unboundedly satisfiable* if, for any natural number $k \in \{0, 1, 2, ...\}$, there exists a Web of Linked Data W such that the cardinality of query result $Q(W)$ is greater than k; i.e., $|Q(W)| > k$. A Linked Data query is *boundedly satisfiable* if it is satisfiable but *not* unboundedly satisfiable. □

Remark 2.1. From Definition 2.8 we see that every unboundedly satisfiable Linked Data query is also satisfiable. Hence, the two disjoint classes—boundedly satisfiable Linked Data queries and unboundedly satisfiable Linked Data queries—cover the more general class of satisfiable Linked Data queries completely.

2.1.3. Related Work

We conclude the discussion of our data model with an overview on related work. In particular, we review general models of the WWW briefly and, thereafter, discuss approaches for modeling Linked Data on the WWW.

Models of the World Wide Web

Early works that introduce models of the WWW, are based on an understanding of the WWW as a distributed hypertext system consisting of Web pages that are interconnected by *a single type of* hypertext links. Therefore, most of these works model the WWW as a directed graph where each vertex represents a Web page, and edges represent the links between those pages [3, 58, 86, 95, 96, 97, 106, 112, 113]. In most cases, the edges are labeled with the anchor text of the corresponding hypertext link. In some models, vertices have additional attributes (e.g., title, modification date, size) [112, 113] or an internal structure [58, 95, 106, 152]. Furthermore, some authors represent their conceptual, graph-based model as a relational database schema [3, 98, 112, 113, 152].

For instance, Abiteboul and Vianu introduce the notion of a *"Web instance"* [3]. Such a Web instance is an *infinite* structure over the relational database schema

$$\{\text{Obj}(\mathit{oid}),\ \text{Ref}\ (\mathit{source},\ \mathit{label},\ \mathit{destination})\},$$

where the domain of attributes *oid*, *source*, and *destination* is a *"set [...] of object identifiers"* and the domain of attribute *label* is a *"set [...] of labels"* [3]. Then, for each Web instance, *"relation Obj specifies the set of objects [in that instance, where,]* *intuitively, an object corresponds to a Web page"* and *"relation Ref specifies, for each of the objects, a finite set of links to other objects, each of which has a label"* [3]. Note that, although Web pages have only a finite number of links to other pages, any Web instance (in Abiteboul and Vianu's model) consists of infinitely many of these pages.

For a more comprehensive discussion of the challenges and different approaches for *"modeling the Web from a data perspective"* we refer to Gutierrez' overview [60].

Models of Linked Data on the World Wide Web

With the emergence of Linked Data on the WWW, several Linked Data specific query formalisms (cf. Section 3.1, page 34ff) and query execution techniques (cf. Chapter 5, page 99ff) have been proposed in the literature. Along with these works come a number of more or less formal approaches for modeling Linked Data on the WWW.

Surprisingly, none of these approaches presents a graph-based model. Instead, the common method of modeling Linked Data is to, first, introduce the notion of an *"RDF graph"* [26, 66, 88, 139, 160, 162], which, essentially, is a set of RDF triples (other authors call this concept *"source"* [99] or refer to it simply as a *"set of triples"* [48]). Then, most authors define some structure that consists of (i) a set of such RDF graphs and (ii) specific mappings that capture the possibility for obtaining these RDF graphs based on Web protocols [48, 66, 88, 160] (in other models such a structure is not made explicit [26, 99, 139, 162]). While the mappings in such a structure are similar to the mappings *adoc* and *data* in our data model, the set of RDF graphs in such a structure corresponds to the image of mapping *data*.

The primary conceptual difference between the models introduced by different authors is whether every URI is directly mapped to a (potentially empty) set of RDF triples [48, 99], or whether there exists a level of indirection between URIs and sets of RDF triples [26, 66, 88, 139, 160]. Authors who introduce such a level of indirection aim to distinguish between (i) *"URIs [for which] HTTP lookups [...] directly return [HTTP] status code `200 OK` and data in a suitable RDF format"* and (ii) URIs for which Web servers answer with an *"HTTP redirect to another URI"* [160]. Based on such a distinction it is possible to make explicit that (i) the lookup of a particular URI involves multiple HTTP requests and (ii) looking up different URIs may result in retrieving the same RDF graph (from the same location on the Web).

While our data model does not focus on such protocol-specific details, our notion of an LD document represents a similar level of indirection. Thus, our model also allows us to express that looking up different URIs may result in retrieving the same set of RDF triples from the same location. Furthermore, we may express that the same set of RDF triples is available from different locations. As an additional feature, by introducing the concept of a link graph, our model makes the graph-based nature of the Web explicit.

2.2. Computation Model

Our data model provides the foundation to define precisely the queries that Linked Data query execution systems for the WWW aim to compute. To analyze the feasibility of computing such queries we need a model of computation that captures the capabilities of such systems. This section introduces such a model.

The commonly used model of computation, a Turing machine [158], is unsuitable in our context because of the assumption that functions are computed over finite structures that are directly and completely accessible by the system that performs the computation. This assumption does not hold for a Web of Linked Data $W = (D, data, adoc)$ that is distributed over the WWW: First, W may be infinitely large. Second, and more importantly, we cannot assume that a system for executing Linked Data queries over this Web has direct access to any of the three elements D, $data$, and $adoc$. Instead, such a system can only access the queried Web by looking up URIs. Since the set $\mathcal{U}$ of all URIs is infinite, the system cannot have complete information about W (in particular about mapping $adoc$). Finally, if we assume that there does not exist a *complete* catalog of all URIs based on which the system may retrieve an LD document $d \in D$ (i.e., a catalog of all URIs $u \in \mathcal{U}$ for which $adoc(u) \neq \bot$), the system can never be certain that it has seen all LD documents in D.

More suitable models of computation on the WWW have been introduced by Abiteboul and Vianu [3] and by Mendelzon and Milo [112]. These models are based on the hypertext-centric view of the WWW as outlined in Section 2.1.3 (cf. page 24f). For our computation model we adopt the idea of these existing models to our context (that is, a Web of Linked Data distributed over the WWW).

In the following, we briefly describe the work of Abiteboul and Vianu and of Mendelzon and Milo. Afterwards, we introduce an abstract machine that captures the capabilities of Linked Data query execution systems for the WWW. Based on this machine we define (i) a classification for the computability of Linked Data queries and (ii) a notion of decidability for decision problems whose input is a Web of Linked Data.

2.2.1. Existing Models of Computation on the WWW

In earlier work on computing queries over a hypertext-centric view of the WWW (cf. Section 2.1.3, page 24f), Abiteboul and Vianu model computation using specific adaptations of the Turing machine which they call *"Web machine"*, *"browser machine"*, and *"browse/search machine"* [3]. The latter two machine models capture the limited data access capabilities on the WWW. That is, the *"Web instance"* (Abiteboul and Vianu's abstraction of a snapshot of the WWW; cf. Section 2.1.3) is not available directly to programs run by these machines. Instead, such a program needs to write an encoding of a valid object identifier (that exists in the Web instance) to a special *"browsing tape"* and then enter a special state, called *"expand"* [3]; in this state the machine replaces the object identifier on the browsing tape by an encoding of the description of the corresponding object (that is, all those *Ref* tuples whose *source* attribute is the given object identifier; cf. Section 2.1.3). Hence, this procedure resembles the process of retrieving a Web page by looking up the URL of that page. The browse/search machine augments the browser machine with a nondeterministic search feature that captures the possibility of querying a search engine for the WWW.

Based on their machines, Abiteboul and Vianu introduce particular notions of computability for queries over a Web instance. These notions are: *finitely computable queries*, which correspond to the traditional notion of computability; and *eventually computable queries* whose computation may not terminate but each element of the query result will eventually be reported during the computation [3].

Mendelzon and Milo introduce a similar machine model and similar notions of computability [112]. They also call their machine *"Web machine."* However, this machine is analogous to Abiteboul and Vianu's browser machine. The only significant difference between Abiteboul and Vianu's browser machine and Mendelzon and Milo's Web machine is that the latter allows for programs that guess URLs (or enumerate all possible URLs) in order to discover Web pages by chance; by contrast, the browser machine terminates when presented with an object identifier that does not exist in the queried Web instance. Another fundamental difference between the authors' works exists in the underlying data models: While Abiteboul and Vianu's Web instances *must be* infinite [3], Mendelzon and Milo require finiteness for every instance of their *"Web database schema"* [112] (which is a relational database schema similar to Abiteboul and Vianu's schema that we describe in Section 2.1.3). This divergence leads to subtle differences in both authors' notions of (finitely) computable queries and eventually computable queries (however, these differences are not relevant in this dissertation).

We adopt the ideas of Abiteboul and Vianu and of Mendelzon and Milo for our work. More precisely, we adapt the idea of a browser machine (respectively Mendelzon and Milo's Web machine) to our scenario of a Web of Linked Data. We call our machine a *Linked Data machine* (or *LD machine*, for short). Furthermore, similar to Abiteboul and Vianu's and Mendelzon and Milo's notions of computability, we use our LD machine to define finite computability and eventual computability for Linked Data queries.

2.2.2. The LD Machine

As a preliminary for defining our LD machine we have to specify how to encode (fragments of) a Web of Linked Data $W = (D, data, adoc)$ on the tapes of such a machine. Such an encoding is straightforward because all relevant structures—such as the set D of all LD documents in W or the set $\mathcal{U}$ of all URIs—are countable (even if infinite). Hereafter, we write $\text{enc}(x)$ to denote the encoding of some element x, where x may be a single RDF triple, a set of RDF triples, a full Web of Linked Data, etc. For a detailed definition of the encoding that we use in this dissertation, we refer to Appendix B.

We now define the LD machine as a multi-tape Turing machine with *limited access* to an input Web of Linked Data. We emphasize that using multiple tapes is a technical detail; an analogous definition with a single tape is possible but it would make the presentation unnecessarily involved.

Definition 2.9 (LD Machine). An *LD machine* is a multi-tape Turing machine with five tapes and a finite set of states, including a special state called *expand*. The five tapes consist of two, read-only input tapes: (i) an ordinary input tape and (ii) a right-infinite *Web input tape* which can only be accessed in the expand state; two work tapes: (iii) an ordinary, two-way infinite work tape and (iv) a right-infinite *lookup tape*; and (v) a right-infinite, append-only output tape. Initially, the work tapes and the output tape are empty, the Web input tape contains a (potentially infinite) word that encodes a

Web of Linked Data, and the ordinary input tape contains an encoding of further input (if any). An LD machine operates like an ordinary multi-tape Turing machine except when it reaches the expand state. In this case the machine performs the following *expand procedure*: The machine inspects the word on the lookup tape. If the suffix of this word is the encoding $\mathrm{enc}(u)$ of some URI $u \in \mathcal{U}$ and the word on the Web input tape contains $\sharp\,\mathrm{enc}(u)\,\mathrm{enc}(adoc(u))\,\sharp$, then the machine appends $\mathrm{enc}(adoc(u))\,\sharp$ to the (right) end of the word on the lookup tape by copying from the Web input tape; otherwise, the machine appends $\sharp$ to the word on the lookup tape. □

Notice how an LD machine is limited in the way it may access a Web of Linked Data $W = (D, data, adoc)$ that is encoded on its Web input tape: The machine may use the data of any particular LD document $d \in D$ only after performing the expand procedure using a URI $u \in \mathcal{U}$ for which $adoc(u) = d$. Hence, the expand procedure simulates a URI lookup which conforms to the typical method for accessing Linked Data on the WWW.

In comparison to the aforementioned machine models (cf. Section 2.2.1), the expand procedure of our LD machine corresponds to entering the expand state in Abiteboul and Vianu's browser machine [3] and to calling the oracle in Mendelzon and Milo's Web machine [112]. Furthermore, similar to Mendelzon and Milo's Web machine (and in contrast to Abiteboul and Vianu's browser machine), an LD machine supports programs that guess or enumerate URIs for lookup.

We note that the word on the Web input tape of an LD machine is infinitely large if (and only if) the encoded Web of Linked Data $W = (D, data, adoc)$ is infinite. Such an infinitely large input is impossible with an ordinary Turing machine. We also note that the words on the other tapes of an LD machine are always finite at any step in any possible computation of such a machine.

In this dissertation we are particularly interested in LD machines that (aim to) compute a Linked Data query Q over a Web of Linked Data W encoded on the Web input tape. The general process of such a computation is to use the expand procedure to access W as needed for the computation and, for any solution $\mu \in Q(W)$ computed during the process, write an encoding of μ to the output tape. We emphasize that no particular order is required for the output unless query Q explicitly asks for an ordered query result (assuming the query language used for Q provides such a feature).

Since our abstract notion of a Linked Data query allows for arbitrary types of solutions (cf. Section 2.1.2, page 22ff), the particular convention for encoding solutions (on the output tape of an LD machine) depends on the particular type of result elements that is specific to the query formalism used for query Q. For instance, our encoding in Appendix B covers the type of result elements that are relevant in this dissertation (that is, valuations, which we introduce formally in Section 3.2.1, page 38ff). Since we require countability for any (query formalism specific) set of possible result elements (cf. Definition 2.5, page 23), it is straightforward to augment our encoding to cover other types of result elements. Therefore, for the definitions in the remainder of this chapter, we assume that any possible query formalism is associated with such an augmented encoding and write $\mathrm{enc}(\mu)$ to denote the encoding of a result element μ that is possible in this formalism. Furthermore, if a query result $Q(W)$ may be reported in an arbitrary order (see above), there exist different *possible encodings* of $Q(W)$, each of which is a word that concatenates the encodings of all $\mu \in Q(W)$ in a different order.

2.2.3. LD Machine Computability

We now use LD machines to adapt Abiteboul and Vianu's notions of finite computability and eventual computability [3] for Linked Data queries.

Definition 2.10 (Finitely Computable Linked Data Query). A Linked Data query Q is *finitely computable* by an LD machine if there exists an LD machine M whose computation, for any Web of Linked Data W encoded on the Web input tape of M, halts after a finite number of computation steps and produces a possible encoding of query result $Q(W)$ on the output tape of M. □

Definition 2.11 (Eventually Computable Linked Data Query). A Linked Data query Q is *eventually computable* by an LD machine if there exists an LD machine M whose computation on any Web of Linked Data W encoded on the Web input tape of M has the following two properties:

1. The word on the output tape of M at each step of the computation is a prefix of a possible encoding of query result $Q(W)$.
2. The encoding $\mathrm{enc}(\mu)$ of each solution $\mu \in Q(W)$ becomes part of the word on the output tape of M after a finite number of computation steps. □

Informally, finite computability of a Linked Data query requires the existence of an LD machine that guarantees a terminating, complete computation of the query over any possible Web of Linked Data. Hence, finite computability resembles the traditional notion of computability. In contrast, eventual computability does not require a termination guarantee. However, as finite computability, eventual computability requires that every element of the corresponding query result is reported eventually. Due to these definitions, all finitely computable Linked Data queries are also eventually computable. The converse of this statement does not hold in general.

We note that computing any unsatisfiable query is trivial: Any machine for such a query may immediately report the empty result. We may show the following proposition.

Proposition 2.3. *Any Linked Data query that is unsatisfiable (as per Definition 2.7, page 23), is finitely computable by an LD machine.*

Proof. W.l.o.g., let Q be an arbitrary unsatisfiable Linked Data query; that is, the expected query result $Q(W)$ is empty for any possible Web of Linked Data W. Assume an LD machine for Q that immediately terminates its computation without writing anything to its (initially empty) output tape. Clearly, for any Web of Linked Data W encoded on the Web input tape of this machine, the machine produces the expected query result, $Q(W) = \emptyset$, on its output tape and halts (as required by Definition 2.10). Hence, Linked Data query Q is finitely computable by an LD machine. ■

2.2.4. LD Machine Decidability

Our analysis of Linked Data queries in the following chapters includes a discussion of decision problems that have a Web of Linked Data as input (e.g., problems such as: is the expected query result for a given Linked Data query over a given Web of Linked Data finite?). These problems depart from the usual notion of decision problems defined based on the Turing machine (that cannot have infinitely large input). Therefore, we call them *Web of Linked Data decision problems* (or simply *LD decision problems*).

Definition 2.12 (LD Decision Problem). Let $\mathcal{W}$ be a (potentially infinite) set of Webs of Linked Data (each of which may be infinite itself); let $\mathcal{X}$ be an arbitrary (potentially infinite) set of finite structures; and let $DP \subseteq \mathcal{W} \times \mathcal{X}$ be a subset of pairs from $\mathcal{W}$ and $\mathcal{X}$. The *LD decision problem* for DP is the problem to decide for any possible pair $(W, X) \in \mathcal{W} \times \mathcal{X}$ whether $(W, X) \in DP$. □

Example 2.4. A trivially simple LD decision problem is to decide whether a given Web of Linked Data contains an authoritative LD document for a given URI. Let us call this problem AuthDocExistence. Given that $\mathcal{W}_{\mathsf{All}}$ denotes the (infinite) set of *all* possible Webs of Linked Data (cf. Definition 2.5, page 23) and $\mathcal{U}$ is the (infinite) set of all URIs, we formalize AuthDocExistence as the LD decision problem for a subset $DP_{\text{AuthDocExistence}} \subseteq \mathcal{W}_{\mathsf{All}} \times \mathcal{U}$ that is given as follows:

$$DP_{\text{AuthDocExistence}} = \Big\{ \big((D, data, adoc), u \big) \in \mathcal{W}_{\mathsf{All}} \times \mathcal{U} \,\Big|\, adoc(u) \neq \perp \Big\}.$$

Notice, $\mathcal{W}_{\mathsf{All}}$ corresponds to $\mathcal{W}$ in Definition 2.12, and $\mathcal{U}$ corresponds to $\mathcal{X}$. □

As for the computation of Linked Data queries, we assume that a Web of Linked Data that is input to an LD decision problem, is distributed over the WWW. Then, we are interested in the feasibility of computing answers to such a problem if we restrict the computation to the (limited) capabilities of systems that access Linked Data on the WWW. That is, we aim to analyze whether such a problem can be answered by an LD machine (with an encoding of the Web of Linked Data on its Web input tape). Consequently, we introduce a notion of *LD machine decidability*.

Definition 2.13 (LD Machine Decidability). Let $\mathcal{W}$ be a (potentially infinite) set of Webs of Linked Data and let $\mathcal{X}$ be an arbitrary (potentially infinite) set of finite structures. The LD decision problem for a subset $DP \subseteq \mathcal{W} \times \mathcal{X}$ is *LD machine decidable* if there exist an LD machine M_{DP} whose computation on any Web of Linked Data $W \in \mathcal{W}$ (encoded on the Web input tape of M_{DP}) and any $X \in \mathcal{X}$ encoded on the ordinary input tape of M_{DP}, has the following property: M_{DP} halts with a nonempty output tape if $(W, X) \in DP$; otherwise the machine halts with an empty output tape. □

Example 2.5. It is trivial to show that LD decision problem AuthDocExistence, as introduced in Example 2.4, is LD machine decidable. We only have to construct an LD machine M that performs the following program (for any possible Web of Linked Data $W = (D, data, adoc) \in \mathcal{W}_{\mathsf{All}}$ encoded on the Web input tape and any possible URI $u \in \mathcal{U}$ encoded on the ordinary input tape): First, M copies $\text{enc}(u)$ from its ordinary input tape to its lookup tape. Next, M enters its expand state to perform the expand procedure using URI u. Finally, M checks the resulting word on the lookup tape. By Definition 2.9 (cf. page 27), this word may either be $\mathrm{enc}(u)\,\mathrm{enc}(adoc(u))\,\sharp$ or $\mathrm{enc}(u)\,\sharp$.

In the first case, $adoc(u) \neq \perp$ holds (i.e., the given Web of Linked Data W contains an authoritative LD document $d = adoc(u)$ for the given URI u). In this case, to indicate that $(W, u) \in DP_{\textsc{AuthDocExistence}}$ holds, machine M adds an arbitrary word to its output tape (e.g., the symbol $\sharp$) and halts.

In the second case, $adoc(u) = \perp$ holds and, thus, $(W, u) \notin DP_{\textsc{AuthDocExistence}}$. In this case, machine M simply halts (without adding anything to the empty output tape). $\square$

Hereafter, instead of specifying their subset $DP \subseteq \mathcal{W} \times \mathcal{X}$ (as per Definition 2.12), we represent the definition of LD decision problems as shown for AuthDocExistence in the following:

LD Problem:	AuthDocExistence
Web Input:	a Web of Linked Data $W = (D, data, adoc)$
Ordin. Input:	a URI u
Question:	Does $adoc(u) \neq \perp$ hold?

We emphasize that an LD machine may also be used to (try to) compute any ordinary decision problem. The input to such a problem may be encoded on the ordinary input tape of the LD machine. Then, by ignoring its Web input tape (and its expand procedure), the machine may behave like a standard (multi-tape) Turing machine and simulate a potentially existing decider (that is, an ordinary Turing machine that can be used for deciding the given ordinary decision problem). More precisely, for any ordinary decision problem, if and only if this problem is (Turing) decidable, an LD machine may simulate the corresponding decider (and, thus, be a decider itself).

The definition of LD machine decidability completes the mathematical framework that we need to define and to analyze concrete types of Linked Data queries in the following chapters.

3. Full-Web Query Semantics

In the previous chapter we introduce a Web of Linked Data as a model of data published and interlinked on a (distributed) platform such as the WWW. To query such a Web of Linked Data we require a language for expressing queries and a well-defined semantics for this query language. In particular, such a semantics must define the expected query result for any possible query expression over any possible Web of Linked Data. In this dissertation we focus on using the SPARQL query language [63] for such a purpose.

SPARQL is the de facto, declarative query language for the RDF data model. That is, using its standard query semantics, SPARQL allows for querying fixed, a-priory defined collections of RDF data. Such a collection may be stored in a DBMS for RDF data. Storing RDF data and querying it efficiently has attracted much research during recent years (e.g., [1, 65, 76, 89, 104, 122, 123, 140, 149, 153, 157, 164, 169]).

Although the SPARQL query language is defined for expressing queries over fixed, a-priory defined collections of RDF data, it seems natural to use SPARQL also for Linked Data queries (that is, queries over a Web of Linked Data), because Linked Data is represented based on RDF triples. In fact, all published approaches to process Linked Data queries that we are aware of (and review in Chapter 5), assume that those queries are expressed using SPARQL (or a fragment thereof). However, a precise definition of the query semantics assumed for the supported queries is missing in most of these works.

To use SPARQL as a language for Linked Data queries, we have to adjust the query semantics of SPARQL. More precisely, we have to redefine the scope for evaluating SPARQL expressions. In this chapter we focus on a first approach for such an adjustment. Informally, the scope of a query under the semantics introduced by this approach, is the complete set of all data on the queried Web (a formal definition follows in Section 3.2.2). Consequently, we call this query semantics *full-Web semantics*.

The main contribution of this chapter is a formal analysis of theoretical properties of queries under full-Web semantics. It is a common assumption that such queries have limited computability; for instance, Ladwig and Tran assume that *"processing queries against Linked Data where sources have to be discovered online might not yield all results"* [99]. Our analysis provides a formal verification of these assumptions. In particular, we study basic properties (such as satisfiability and monotonicity) and computation related properties (such as LD machine based computability and the decidability of termination for query computations). Furthermore, we discuss the implications of querying an infinite Web of Linked Data.

We begin the chapter with a discussion of related work on query languages and query semantics for Linked Data queries and for accessing the WWW in general (cf. Section 3.1). Thereafter, we provide a formal definition of SPARQL and of the full-Web semantics that allows us to query a Web of Linked Data using SPARQL (cf. Section 3.2).

Given these definitions, we provide an analysis of SPARQL-based Linked Data queries under full-Web semantics in Section 3.3. In particular, Section 3.3.1 focuses on basic properties (satisfiability, bounded and unbounded satisfiability, and monotonicity). Sections 3.3.2, 3.3.3, and 3.3.4 discuss the termination problem, LD machine based computability, and finiteness of (expected) query results, respectively. Finally, Section 3.4 summarizes our results.

3.1. Related Work

Since its emergence the WWW has spawned research on declarative query languages for a retrieval of information from the WWW. In this section we briefly review general (i.e., Linked Data independent) query languages for the WWW and, afterwards, discuss query languages for expressing queries over Webs of Linked Data. In this context we particularly look at proposals for using SPARQL as a language for Linked Data queries.

3.1.1. Web Query Languages

Initial work on querying the WWW emerged in the late 1990s. For an overview on early work in this area we refer to Florescu et al.'s survey [49]. Most of this work is based on the hypertext-centric view that led to modeling the WWW as a graph of Web pages and links between them (as mentioned in Section 2.1.3, page 24f). Query languages proposed and studied in this context allow a user to either ask for:

- specific Web pages (e.g., W3QL [94, 95]),
- particular attributes of specific Web pages (e.g., WebSQL [9, 113], F-logic [86], Web Calculus [112]), or
- particular content within specific Web pages (e.g., WebOQL [8], NetQL [58], WebLog [102], WQL [106], HTML-QL [109], NALG [111], Squeal [152]).

Common to these languages is the navigational nature of the queries. That is, each of these Web queries is based on some form of path expression that specifies navigation paths to relevant Web pages. The specific form of path expressions supported by each Web query language is tailored to the particular graph structure used for modeling the WWW, respectively.

3.1.2. Navigational Query Languages for Linked Data

In recent years, some research groups have started to work on navigational query languages tailored to query Linked Data on the WWW [48, 138]. These languages are similar in nature to the aforementioned, more general Web query languages. That is, they introduce some form of path expressions based on which a user may specify navigation paths over the link graph of a queried Web of Linked Data. To the best of our knowledge, two such languages have been proposed in the literature: LDPath [138] and NautiLOD [48]. In the following we briefly describe both languages.

In LDPath, the basic type of path expressions is a *"property selection"* that refers to a particular URI. Such an expression selects the objects of those RDF triples 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 [138]. To our knowledge, there does not exist a formally defined semantics for LDPath. However, according to Schaffert et al. [138], *"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"* [138]. W.r.t. the computability of LDPath, Schaffert et al. claim that *"in LDPath, only queries can be expressed that can be evaluated properly over Linked Data"* [138]. However, without a well-defined query semantics the justification of this claim remains unclear.

NautiLOD expressions, in contrast, come with a formal semantics [48]. In terms of our data model, the result of evaluating such an expression is a set of URIs whose lookup yields an LD 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 [48].

The expressive power of NautiLOD is not comparable to the power of SPARQL for Linked Data queries (as studied in this dissertation). The latter does not provide navigational features (that is, expressions for describing navigation over the link graph of a Web of Linked Data). On the other hand, NautiLOD (and LDPath) cannot be used to express conditions across data from multiple LD documents. As mentioned before, elements of the result of evaluating a NautiLOD expression are simply URIs, whereas, for SPARQL-based Linked Data queries, any result element is a set of variable bindings whose computation may require (the discovery and) combination of multiple LD documents.

3.1.3. SPARQL as a Query Language for Linked Data

Instead of developing a new language for Linked Data queries, in this dissertation we focus on using the RDF query language SPARQL for such a purpose. We shall see that this approach allows users who are not interested in prescribing particular navigation paths, to express queries over Linked Data without knowing anything about the link graph of the queried Web of Linked Data. Another motivation for studying SPARQL-based Linked Data queries is the focus on such queries in existing works on processing Linked Data queries. While we postpone a discussion of these works to Chapter 5, we emphasize that most of them lack a precise definition of the query semantics assumed for the supported queries.

The theoretical properties of SPARQL as a query language for fixed, a-priory defined collections of RDF data are well understood today [7, 127, 128, 140]. Particularly interesting in our context are semantical equivalences between SPARQL expressions [140]; these equivalences may also be used for optimizing SPARQL-based Linked Data queries.

Bouquet et al. were the first to provide a formalization for using (a fragment of) SPARQL as a language for Linked Data queries [26]. Other proposals have been published by Harth and Speiser [66] and by Umbrich et al. [160]; a first version of one of the reachability-based query semantics presented in this dissertation can be found in our earlier work on Linked Data query processing [72]. The remainder of this section describes these proposals in detail and informally compares the respectively introduced query semantics to the query semantics that we study in both this and the following chapter.

Bouquet et al. formalize three *"query methods"* for conjunctive queries [26]. In terms of our data model these methods can be characterized as follows:

"Bounded method": This method assumes that queries include a specification of a set of LD documents. The evaluation of such a query is then restricted to the data in these documents. This approach corresponds to the most restrictive version of our reachability-based query semantics, namely, our notion of c_{None}-semantics (cf. Section 4.1.2, page 63f).

"Navigational method": This 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 LD document (specified as part of the query). This navigational method prescribes a query semantics that is equivalent to, what we call, c_{All}-semantics (cf. Section 4.1.2); it is the most general of our reachability-based semantics. Bouquet et al.'s navigational method does not support other, more restrictive notions of reachability, as is possible with our model.

"Direct access method": For this method, Bouquet et al. assume an oracle that, for a given query, provides a set of *"relevant"* LD documents (from a queried Web of Linked Data). Without discussing their understanding of relevance any further, the authors define an expected query result based on such a set of relevant documents. Due to the undefined basis of this definition, it is unclear how Bouquet et al.'s direct access method is related to the query semantics discussed in this dissertation.

After introducing these query methods, Bouquet et al. use model theory to establish a formalism for interpreting query results [26]. However, an analysis of the properties or the feasibility of the three query methods is missing from Bouquet et al.'s work.

Harth and Speiser also propose several query semantics for conjunctive Linked Data queries [66]. These semantics use authoritativeness of data sources to restrict the evaluation of queries to particular subsets of all data in a queried Web. In terms of our data model, the authors call an LD document d *"subject-authoritative"* for an RDF triple $t = (s, p, o)$ in a Web of Linked Data $W = (D, data, adoc)$, if s is a URI and $adoc(s) = d$; analogously, LD documents may be predicate-authoritative and object-authoritative for a given RDF triple. Based on these notions of authoritativeness, the authors introduce a formalism that allows users to express what data they consider relevant for a query. More precisely, users may specify that only those RDF triples are relevant for evaluating a query, that are available in their authoritative documents (or in particular subsets thereof); any other triple is considered irrelevant and must be ignored. These restrictions may be specified separately for each predicate of a given conjunctive query.

In addition to these data-specific *"authority restrictions"* [66], Harth and Speiser introduce three *"completeness classes."* For any query, such a class designates particular documents from a queried Web such that, by definition, these documents are considered *"completely sufficient"* for the query [66]. Hence, these completeness classes may be understood as document-specific restrictions on the relevance of data. The authors' formalism allows users to combine any of the three completeness classes with any possible set of authority restrictions. Then, an RDF triple needs to be considered for a predicate of a given (conjunctive) query if and only if (i) the triple is available in its authoritative documents (or in a specified subset thereof) and (ii) these documents qualify according to the completeness class used. Thus, depending on which completeness class and authority restrictions are used, a different query semantics ensues.

Unfortunately, Harth and Speiser's work lacks a proper formal definition of one of the key concepts for specifying authority restrictions (that is, the concept of an *"authoritative lookup"*—represented by a function called derefa [66, Definition 10]). Therefore, it is impossible to discuss Harth and Speiser's query semantics in detail or to provide an informed comparison with the query semantics discussed in this dissertation.

Umbrich et al. define five different query semantics for conjunctive Linked Data queries and analyze them empirically [160]. The first of these semantics is the reachability-based query semantics presented in our earlier work [72]. In this dissertation we shall refer to this semantics as c_{Match}-semantics (cf. Section 4.1.2). In comparison to the reachability-based c_{All}-semantics that corresponds to the aforementioned navigational method of Bouquet et al. [26], we shall see that c_{Match}-semantics is more restrictive (w.r.t. what data is considered relevant for evaluating queries).

However, the main contribution of the work by Umbrich et al. are several query semantics that extend c_{Match}-semantics in order to *"benefit [from] inferable knowledge"* [160]. 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 [119] (restricted to a fixed, a-priori defined set of vocabularies), and (ii) inference rules for RDF triples with the predicate owl:sameAs [142]. The latter allows for making use of information about coreferenced entities because owl:sameAs is commonly used to indicate coreferencing URIs in Linked Data [42].

In an empirical analysis, Umbrich et al. compare their extended, inference-based query semantics to c_{Match}-semantics (which does not integrate inference rules) [160]. This analysis shows that the number of solutions in a query result under any of the inference-based semantics is usually greater than the result for the corresponding query under c_{Match}-semantics. The price for such an increase in *"recall"* [160] is an increase in average query execution times because, for a complete execution of queries under the inference-based semantics, it becomes necessary to look up more URIs than under c_{Match}-semantics.

We consider extending Linked Data queries with features for leveraging inferable knowledge a very interesting topic for future research. However, the aim of this dissertation is to establish a comprehensively studied foundation for the base case (that is, Linked Data queries without inference rules).

In summary, a common limitation of the query semantics that have been proposed to use SPARQL as a language for Linked Data queries [26, 66, 72, 160], is their focus on a very basic type of SPARQL expressions, namely, *"basic graph patterns"* [63], which allow users to express some form of conjunctive queries (cf. Section 6.1, page 111ff). By contrast, this dissertation covers the complete core fragment of SPARQL; in addition to conjunctions, this fragment includes disjunctions, constraints on variable bindings, and optional parts. Furthermore, the aforementioned proposals merely define some query semantics without properly analyzing what a sound and complete support of these semantics entails. That is, a formal analysis of queries under a given semantics—our primary contribution in both this and the following chapter—is missing for any of these proposals.

3.2. Definition

This section provides a formal definition of the full-Web query semantics that allows us to query a Web of Linked Data using the SPARQL query language. As a preliminary for this definition we need to introduce SPARQL and the standard SPARQL semantics (which focuses on queries over a-priori defined sets of RDF triples). Afterwards, based on the standard semantics, we define the full-Web semantics for SPARQL.

3.2.1. SPARQL

This dissertation focuses on the core fragment of SPARQL discussed by Pérez et al. [128]. We adopt their formalization approach by using the algebraic syntax and the compositional set semantics as introduced in [128]. We emphasize that the official W3C specification introduces bag semantics for SPARQL [63]; the definition of this bag semantics is based on a formalism that comprises compositional and operational elements [7]. For a detailed comparison of both formalization approaches we refer to the work of Angles and Gutierrez [7], who show that both approaches have the same expressive power (under bag semantics). Pérez et al. provide a brief historical survey of the different approaches [128].

In the following, we introduce the syntax and the (standard) semantics of SPARQL expressions as used in this dissertation. Furthermore, we define basic theoretical properties (such as satisfiability and monotonicity) for SPARQL expressions.

SPARQL Syntax

The basic elements of the algebraic syntax as introduced by Pérez et al. [128], are triple patterns and filter conditions:

- A *triple pattern* is a tupel $(s, p, o) \in (\mathcal{V} \cup \mathcal{U}) \times (\mathcal{V} \cup \mathcal{U}) \times (\mathcal{V} \cup \mathcal{U} \cup \mathcal{L})$, where $\mathcal{V}$ is an infinite set of query variables that is disjoint from $\mathcal{U}$, $\mathcal{B}$, and $\mathcal{L}$, respectively (we recall that $\mathcal{U}$ are all URIs, $\mathcal{B}$ are all blank nodes, and $\mathcal{L}$ are all literals; cf. Section 2.1, page 15ff). We denote variables by a leading question mark symbol (e.g., $?v \in \mathcal{V}$). Function vars maps triple patterns to the (finite) set of all variables mentioned in such a pattern; that is, for each triple pattern $tp = (s, p, o)$, $\mathrm{vars}(tp) := \{s, p, o\} \cap \mathcal{V}$.
- A *filter condition* is defined recursively as follows:
  1. If $?x, ?y \in \mathcal{V}$ and $c \in (\mathcal{U} \cup \mathcal{L})$, then $?x = c$, $?x = ?y$, and $\mathrm{bound}(?x)$ are filter conditions.
  2. If R_1 and R_2 are filter conditions, then $(\neg R_1)$, $(R_1 \wedge R_2)$, and $(R_1 \vee R_2)$ are filter conditions.

Given these basic elements, a *SPARQL expression* is defined recursively as follows:

1. A triple pattern is a SPARQL expression.
2. If P_1 and P_2 are SPARQL expressions and R is a filter condition, then $(P_1 \text{ AND } P_2)$, $(P_1 \text{ UNION } P_2)$, $(P_1 \text{ OPT } P_2)$, and $(P_1 \text{ FILTER } R)$ are SPARQL expressions.

Overloading function vars, we write $\mathrm{vars}(P)$ to denote the (finite) set of all variables in all triple patterns of a SPARQL expression P.

In contrast to the official SPARQL syntax introduced by the W3C specification [63], the syntax that we use here avoids the use of blank nodes in SPARQL expressions (for the sake of more straightforward definitions). However, omitting blank nodes is not a problem because *"each SPARQL query [with blank nodes] can be simulated by a SPARQL query [...] without blank nodes"* [7].

SPARQL Semantics

As a preliminary for defining the semantics of SPARQL we introduce the notion of valuations:[1] A *valuation* μ is a partial mapping $\mu : \mathcal{V} \to \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$ that is defined for a finite subset of $\mathcal{V}$ (the set of all variables). For valuations we overload function terms, that is, for each valuation μ, we define $\mathrm{terms}(\mu) := \{\mu(?v) \mid ?v \in \mathrm{dom}(\mu)\}$, where $\mathrm{dom}(\mu)$ denotes the domain of μ (i.e., the subset of $\mathcal{V}$ for which μ is defined). Furthermore, we let $\mathrm{uris}(\mu) := \mathrm{terms}(\mu) \cap \mathcal{U}$. A specific valuation is the *empty valuation*, denoted by $\mu_\emptyset$, for which it holds that $\mathrm{dom}(\mu_\emptyset) = \emptyset$.

[1] We use the term *valuation* instead of the term *"solution mapping"* as used in the W3C specification of SPARQL [63]; valuation is the standard term used in database theory [4].

The standard semantics of SPARQL is defined based on an evaluation function that maps a SPARQL expression and a set of RDF triples to a set of valuations. Our formalization explicitly allows for infinitely large sets of RDF triples; as a consequence, resulting sets of valuations may be infinite as well. Supporting infiniteness is necessary to study SPARQL in the context of a potentially infinite Web of Linked Data.

To define the evaluation function we need some additional terminology. In particular, we introduce the application of a valuation to a triple pattern, matching triples, compatibility of valuations, and certain binary operations over sets of valuations:

- Given a valuation μ and a triple pattern tp, we write $\mu[tp]$ to denote the triple pattern that we obtain by replacing the variables in tp according to μ (any variable for which μ is not defined, is not replaced). Notice that $\mu[tp]$ is an RDF triple if $\mathrm{vars}(tp) \subseteq \mathrm{dom}(\mu)$.
- Given an RDF triple t and a triple pattern tp, we say t is a *matching triple* for tp if there exists a valuation μ such that $t = \mu[tp]$. Applying the aforementioned empty valuation $\mu_\emptyset$ to a triple pattern has no effect, that is, $\mu_\emptyset[tp] = tp$ for any possible triple pattern tp.
- Two valuations μ and μ' are *compatible*, denoted by $\mu \sim \mu'$, if $\mu(?v) = \mu'(?v)$ for all variables $?v \in \mathrm{dom}(\mu) \cap \mathrm{dom}(\mu')$. The empty valuation $\mu_\emptyset$ is compatible with any other valuation. For two compatible valuations μ and μ', we write $\mu \cup \mu'$ to denote a valuation μ^* for which the following three properties hold: (i) $\mu^* \sim \mu$, (ii) $\mu^* \sim \mu'$, and (iii) $\mathrm{dom}(\mu^*) := \mathrm{dom}(\mu) \cup \mathrm{dom}(\mu')$.
- Let Ω_1 and Ω_2 be sets of valuations, each of which may be finite or countably infinite. The binary operations *join*, *union*, *difference*, and *left outer-join* over Ω_1 and Ω_2 are defined as follows:

$$\begin{aligned}
\Omega_1 \bowtie \Omega_2 &:= \{\mu_1 \cup \mu_2 \,|\, \mu_1 \in \Omega_1 \text{ and } \mu_2 \in \Omega_2 \text{ and } \mu_1 \sim \mu_2\} \\
\Omega_1 \cup \Omega_2 &:= \{\mu \,|\, \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\} \\
\Omega_1 \setminus \Omega_2 &:= \{\mu_1 \in \Omega_1 \,|\, \forall \mu_2 \in \Omega_2 : \mu_1 \not\sim \mu_2\} \\
\Omega_1 ⟕ \Omega_2 &:= (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)
\end{aligned}$$

We are now ready to define the evaluation function that specifies the (standard) semantics of SPARQL expressions: Let P be a SPARQL expression and let G be a (potentially infinite but countable) set of RDF triples. The *evaluation of P over G*, denoted by $[\![P]\!]_G$, is defined recursively as follows:

1. If P is a triple pattern tp, then $[\![P]\!]_G := \{\mu \,|\, \mathrm{dom}(\mu) = \mathrm{vars}(tp) \text{ and } \mu[tp] \in G\}$.
2. If P is $(P_1 \;\mathsf{AND}\; P_2)$, then $[\![P]\!]_G := [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$.
3. If P is $(P_1 \;\mathsf{UNION}\; P_2)$, then $[\![P]\!]_G := [\![P_1]\!]_G \cup [\![P_2]\!]_G$.
4. If P is $(P_1 \;\mathsf{OPT}\; P_2)$, then $[\![P]\!]_G := [\![P_1]\!]_G ⟕ [\![P_2]\!]_G$.

5. If P is $(P_1 \text{ FILTER } R)$, then $[\![P]\!]_G := \{\mu \in [\![P_1]\!]_G \mid \mu \text{ satisfies } R\}$, where a valuation μ satisfies a filter condition R if any of the following conditions holds:
   a) R is $?x = c$, $?x \in \text{dom}(\mu)$, and $\mu(?x) = c$;
   b) R is $?x = ?y$, $?x \in \text{dom}(\mu)$, $?y \in \text{dom}(\mu)$, and $\mu(?x) = \mu(?y)$;
   c) R is $\text{bound}(?x)$ and $?x \in \text{dom}(\mu)$;
   d) R is $(\neg R_1)$ and μ does not satisfy R_1;
   e) R is $(R_1 \wedge R_2)$ and μ satisfies R_1 and R_2; or
   f) R is $(R_1 \vee R_2)$ and μ satisfies R_1 or R_2.

Each valuation $\mu \in [\![P]\!]_G$ is called a *solution* for P in G.

Basic Properties

We conclude the definition of SPARQL by introducing basic properties for SPARQL expressions. For a detailed discussion of these properties we refer to Appendix C in which we identify fragments of SPARQL for which these properties hold (cf. page 195ff).

We first specify how the standard notions of monotonicity and satisfiability carry over to SPARQL: A SPARQL expression P is *monotonic* if, for any pair G_1, G_2 of (potentially infinite) sets of RDF triples with $G_1 \subseteq G_2$, $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$ holds. A SPARQL expression P is *satisfiable* if there exists a (potentially infinite) set of RDF triples G such that the evaluation of P over G is not empty; i.e., $[\![P]\!]_G \neq \emptyset$.

In addition to the traditional notion of satisfiability we introduce more restrictive types of satisfiability for Linked Data queries in Section 2.1.2 (cf. page 22ff). For SPARQL expressions we define these notions of satisfiability as follows: A SPARQL expression P is *unboundedly satisfiable* if for any natural number $k \in \{0, 1, 2, ...\}$ there exists a set of RDF triples G such that $\big|[\![P]\!]_G\big| > k$. A SPARQL expression P is *boundedly satisfiable* if it is satisfiable but *not* unboundedly satisfiable.

Example 3.1. Consider three arbitrary URIs $u_1^*, u_2^*, u_3^* \in \mathcal{U}$ and a variable $?v \in \mathcal{V}$.

The triple pattern $tp_1 = (u_1^*, u_2^*, ?v)$ is unboundedly satisfiable: The set of all matching triples for tp_1 is $\mathcal{T}_{tp_1} = \{u_1^*\} \times \{u_2^*\} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$. This set is infinite because $\mathcal{U}$, $\mathcal{B}$, and $\mathcal{L}$ are infinite, respectively. Then, for any $k \in \{0, 1, 2, ...\}$ we may select a subset $G_k \subset \mathcal{T}_{tp_1}$ of size $k{+}1$; i.e., $|G_k| = k{+}1$. It is easy to see that $\big|[\![tp_1]\!]_{G_k}\big| = k{+}1 > k$. Due to the infiniteness of $\mathcal{T}_{tp_1}$, such a subset exists for all $k \in \{0, 1, 2, ...\}$.

In contrast, triple pattern $tp_2 = (u_1^*, u_2^*, u_3^*)$ is a trivial example of a boundedly satisfiable SPARQL expression: Since $\text{vars}(tp_2) = \emptyset$, tp_2 is an RDF triple. Hence, any possible set of RDF triples may contain at most a single matching triple for tp_2, namely (u_1^*, u_2^*, u_3^*). For any such set G (for which $(u_1^*, u_2^*, u_3^*) \in G$), $[\![tp_2]\!]_G = \{\mu_\emptyset\}$ holds with $\mu_\emptyset$ being the empty valuation. For any other set of RDF triples the expected query result contains no solution at all, that is, $[\![tp_2]\!]_{G'} = \emptyset$ for any set of RDF triples G' for which $(u_1^*, u_2^*, u_3^*) \notin G'$. Therefore, $k = 1$ is an upper bound for the number of possible solutions that may be computed for tp_2; more precisely, there does not exist a set of RDF triples G such that $\big|[\![tp_2]\!]_G\big| > 1$.

Another, less trivial example for boundedly satisfiable SPARQL expressions is the expression $((u_1^*, u_2^*, ?v) \text{ FILTER } (?v = u_1^* \vee ?v = u_3^*))$. This expression contains the aforementioned triple pattern tp_1 as a subexpression. Although the aforementioned set $\mathcal{T}_{tp_1}$ of matching triples for tp_1 is infinite (see above), there exist only two valuations in $[\![tp_1]\!]_{\mathcal{T}_{tp_1}}$ that satisfy the filter condition, namely $\mu = \{?v \to u_1^*\} \in [\![tp_1]\!]_{\mathcal{T}_{tp_1}}$ and $\mu' = \{?v \to u_3^*\} \in [\![tp_1]\!]_{\mathcal{T}_{tp_1}}$. Hence, μ and μ' are the only solutions for the given SPARQL expression in $\mathcal{T}_{tp_1}$. Similarly, there exist at most these two solutions in any other set of RDF triples. □

3.2.2. SPARQL$_{\mathsf{LD}}$

SPARQL expressions are used for queries over sets of RDF triples. The evaluation function introduced before defines the semantics of these queries. A Linked Data query, in contrast, is a function over a Web of Linked Data (cf. Definition 2.5 on page 23). To interpret SPARQL expressions as Linked Data queries we may assume a full-Web query semantics: Informally, the scope of evaluating SPARQL expressions under this semantics is the complete set of all data on the queried Web of Linked Data. Hereafter, we refer to SPARQL-based Linked Data queries under full-Web semantics as *SPARQL$_{\mathsf{LD}}$ queries*. The definition of these queries is unsurprisingly straightforward and makes use of SPARQL expressions and their evaluation function:

Definition 3.1 (SPARQL$_{\mathsf{LD}}$ Query). Given a SPARQL expression P, the *SPARQL$_{\mathsf{LD}}$ query* that uses P, denoted by $\mathcal{Q}^P$, is a Linked Data query that, for any Web of Linked Data W, is defined by $\mathcal{Q}^P(W) := [\![P]\!]_{\mathsf{AllData}(W)}$. Each valuation $\mu \in \mathcal{Q}^P(W)$ is a *solution* for $\mathcal{Q}^P$ in W. □

Example 3.2. Consider a SPARQL expression P_{ex} that is given as follows:

$$\Big((?product, \mathsf{producedBy}, \mathsf{producer1}) \;\; \mathsf{AND} \;\; (?product, \mathsf{name}, ?productName)\Big)$$

If $\mathcal{Q}^{P_{\mathsf{ex}}}$ is the SPARQL$_{\mathsf{LD}}$ query that uses SPARQL expression P_{ex}, the (expected) query result of $\mathcal{Q}^{P_{\mathsf{ex}}}$ over our example Web of Linked Data W_{ex} (cf. Example 2.1, page 18) is $\mathcal{Q}^{P_{\mathsf{ex}}}(W_{\mathsf{ex}}) = \{\mu_1, \mu_2, \mu_3\}$ where $\mu_1 = \{?product \to \mathsf{product1}, ?productName \to \mathsf{"Product\ 1"}\}$, $\mu_2 = \{?product \to \mathsf{product2}, ?productName \to \mathsf{"Product\ 2"}\}$, and $\mu_3 = \{?product \to \mathsf{product3}, ?productName \to \mathsf{"Product\ 3"}\}$. □

3.3. Theoretical Properties

In this section we analyze theoretical properties of SPARQL$_{\mathsf{LD}}$ queries. For this analysis we assume that the data access capabilities in a queried Web of Linked Data are limited as they are in the WWW. Consequently, we use the computation model introduced in the previous chapter (cf. Section 2.2, page 25ff) to analyze SPARQL$_{\mathsf{LD}}$ as follows: We identify cases in which an LD machine based computation of SPARQL$_{\mathsf{LD}}$ queries may (not) terminate with a complete query result. Based on these cases we discuss the more general question of whether an LD machine can decide, for any given SPARQL$_{\mathsf{LD}}$ query, if such a computation is possible. Furthermore, we classify SPARQL$_{\mathsf{LD}}$ queries w.r.t. the notions of finite computability and eventual computability (as introduced in Section 2.2.3, page 29). Our results shall show that both the possibility for terminating computations and the computability depend on the more basic properties satisfiability and monotonicity. Therefore, we first focus on these properties. Afterwards, we study the termination problem for SPARQL$_{\mathsf{LD}}$ and LD machine based computability. Finally, we discuss the finiteness of expected query results.

3.3.1. Satisfiability, (Un)bounded Satisfiability, and Monotonicity

We show that computation related properties of SPARQL$_{\mathsf{LD}}$ queries depend on basic properties such as satisfiability and monotonicity. Thus, to classify any particular SPARQL$_{\mathsf{LD}}$ query w.r.t. computation related properties it is important to identify the basic properties of such a query. For this purpose, we may use results on basic properties of SPARQL expressions (such as our results in Appendix C). Therefore, in the following we show relationships between basic properties of SPARQL expressions and their SPARQL$_{\mathsf{LD}}$ counterparts. Based on these relationships we then carry over our SPARQL specific results to SPARQL$_{\mathsf{LD}}$.

For any SPARQL$_{\mathsf{LD}}$ query the basic properties, satisfiability, bounded and unbounded satisfiability, and monotonicity, are directly correlated with the corresponding property of the SPARQL expression used:

Proposition 3.1. *Let $\mathcal{Q}^P$ be a SPARQL$_{\mathsf{LD}}$ query that uses SPARQL expression P.*

1. *$\mathcal{Q}^P$ is satisfiable if and only if P is satisfiable.*
2. *$\mathcal{Q}^P$ is unboundedly satisfiable if and only if P is unboundedly satisfiable.*
3. *$\mathcal{Q}^P$ is boundedly satisfiable if and only if P is boundedly satisfiable.*
4. *$\mathcal{Q}^P$ is monotonic if and only if P is monotonic.*

While Proposition 3.1 seems trivial, proving it requires some attention because we are concerned with structures that may be infinite. More precisely, our definition of standard SPARQL semantics in Section 3.2.1 (cf. page 38ff) allows for infinitely large sets of RDF triples. For a (hypothetical) Web of Linked Data that we may construct from such a set in a proof of Proposition 3.1, we must ensure that each constructed LD document contains a finite number of triples only (as required by our data model; cf. Definition 2.1, page 16). While it is possible to split up a set of RDF triples—in order to distribute it over multiple (potentially infinitely many) LD documents—dealing with blank nodes in such a case needs additional care: The data of each LD document in a Web of Linked Data must use a unique set of blank nodes (cf. Definition 2.1). Although we could naively rename blank nodes when we distribute the RDF triples from a set G over multiple LD documents, such an approach is insufficient for our proof. If we compute a SPARQL$_{\mathsf{LD}}$

query $\mathcal{Q}^P$ over such a naively constructed Web W, we may lose some solutions from $[\![P]\!]_G$, that is, we may have $\mathcal{Q}^P(W) \subset [\![P]\!]_G$. To avoid this issue we introduce an isomorphism that replaces blank nodes by URIs not used in the corresponding set of RDF triples:

Definition 3.2 (Grounding Isomorphism). Let G be a (potentially infinite) set of RDF triples, let $B_G = \mathrm{terms}(G) \cap \mathcal{B}$ (i.e., the set of all blank nodes mentioned in G), let $U_B \subseteq \mathcal{U}$ be a set of new URIs not mentioned in G (i.e., $U_B \cap \mathrm{terms}(G) = \emptyset$) such that $|U_B| = |B_G|$, and let ϱ_B be a bijection $\varrho_B : B_G \to U_B$ that maps each blank node in G to a new, unique URI $u \in U_B$. Then, a *grounding isomorphism* for G is a bijective mapping $\varrho : \mathrm{terms}(G) \to (U_B \cup (\mathrm{terms}(G) \setminus B_G))$ such that, for each $x \in \mathrm{terms}(G)$,

$$\varrho(x) := \begin{cases} \varrho_B(x) & \text{if } x \in B_G, \\ x & \text{else .} \end{cases} \qquad \square$$

We use the term *grounding* isomorphism because the RDF specification calls a set of RDF triples *grounded*, if this set is free of blank nodes [81].

The *application* of such a grounding isomorphism ϱ (for an arbitrary set of RDF triples) to an arbitrary valuation μ, denoted by $\varrho[\mu]$, results in a valuation μ' such that (i) $\mathrm{dom}(\mu') = \mathrm{dom}(\mu)$ and (ii) $\mu'(?v) = \varrho(\mu(?v))$ for all $?v \in \mathrm{dom}(\mu)$. Furthermore, the application of ϱ to an arbitrary RDF triple $t = (x_1, x_2, x_3)$, denoted by $\varrho[t]$, results in an RDF triple (x'_1, x'_2, x'_3) such that $x'_i = \varrho(x_i)$ for all $i \in \{1, 2, 3\}$.

The following properties are easily verified:

Property 3.1. *Let G be a set of RDF triples, ϱ be a grounding isomorphism for G, and $G' = \{\varrho[t] \,|\, t \in G\}$, then $|G| = |G'|$.*

Property 3.2. *Let G be a set of RDF triples, ϱ be a grounding isomorphism for G, and $G' = \{\varrho[t] \,|\, t \in G\}$. Furthermore, let P be a SPARQL expression and let μ be a valuation. Then, valuation $\mu' = \varrho[\mu]$ is a solution for P in G' if and only if μ is a solution for P in G. More precisely, if we let ϱ^{-1} denote the inverse of the bijective mapping ϱ, then:*

$$\forall\, \mu \in [\![P]\!]_G : \varrho[\mu] \in [\![P]\!]_{G'} \qquad \textit{and} \qquad \forall\, \mu' \in [\![P]\!]_{G'} : \varrho^{-1}[\mu'] \in [\![P]\!]_G \,.$$

To prove Proposition 3.1 we use the concept of a grounding isomorphism to construct the following type of a Web of Linked Data for a given set of RDF triples:

Definition 3.3 (ϱ-Web). Let G be a set of RDF triples, ϱ be a grounding isomorphism for G, and $G' = \{\varrho[t] \,|\, t \in G\}$. A Web of Linked Data $(D, \mathit{data}, \mathit{adoc})$ is a *ϱ-Web* for G if there exist (i) a set $U \subseteq (\mathcal{U} \cap \mathrm{uris}(G'))$ of URIs not mentioned in G', (ii) a bijective mapping $t_U : U \to G'$, and (iii) a bijective mapping $t_D : D \to G'$, such that

$$\forall\, d \in D : \mathit{data}(d) = \{t_D(d)\} \qquad \text{and} \qquad \forall\, u \in \mathcal{U} : \mathit{adoc}(u) = \begin{cases} t_D^{-1}(t_U(u)) & \text{if } u \in U, \\ \perp & \text{else,} \end{cases}$$

where t_D^{-1} denotes the inverse of the bijective mapping t_D. $\square$

Based on our definitions it is trivial to verify the following properties:

Property 3.3. *Let G be a set of RDF triples; let ϱ be a grounding isomorphism for G; and let $W = (D, data, adoc)$ be an arbitrary ϱ-Web for G. W has the following properties:*

1. *W is infinite if and only if G is infinite.*
2. *W contains no data links, that is, $adoc(u) = \bot$ for all $u \in \mathrm{uris}(\mathsf{AllData}(W))$.*
3. *For any SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ it holds that $\mathcal{Q}^P(W) = [\![P]\!]_{G'}$ with $G' = \{\varrho[t] \mid t \in G\}$.*

We now prove Proposition 3.1 by showing its four claims one after another:

Proof of Proposition 3.1, Claim 1 (Satisfiability). Let $\mathcal{Q}^P$ be a SPARQL$_{\mathsf{LD}}$ query.

If: Suppose the SPARQL expression P (used by $\mathcal{Q}^P$) is satisfiable. Then, there exists a set of RDF triples G such that $[\![P]\!]_G \neq \emptyset$. Let ϱ be a grounding isomorphism for G, let $G' = \{\varrho[t] \mid t \in G\}$, and let W be a ϱ-Web for G. Based on Property 3.2 and on the fact that $[\![P]\!]_G \neq \emptyset$, we have $[\![P]\!]_{G'} \neq \emptyset$. Then, by Property 3.3, $\mathcal{Q}^P(W) \neq \emptyset$ and, thus, SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ is satisfiable.

Only if: Suppose SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ is satisfiable. In this case there exists a Web of Linked Data W such that $\mathcal{Q}^P(W) \neq \emptyset$. Since $\mathcal{Q}^P(W) = [\![P]\!]_{\mathsf{AllData}(W)}$ (cf. Definition 3.1, page 42), we conclude that SPARQL expression P is satisfiable. ■

Proof of Proposition 3.1, Claim 2 (Unbounded satisfiability). Let $\mathcal{Q}^P$ be a SPARQL$_{\mathsf{LD}}$ query. We prove Claim 2 using a similar argumentation as for Claim 1.

If: Suppose SPARQL expression P (used by $\mathcal{Q}^P$) is unboundedly satisfiable. W.l.o.g., let $k \in \{0, 1, 2, ...\}$ be an arbitrary natural number. To prove that SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ is unboundedly satisfiable it suffices to show that there exists a Web of Linked Data W such that $|\mathcal{Q}^P(W)| > k$. Since SPARQL expression P is unboundedly satisfiable, there exists a set of RDF triples G such that $|[\![P]\!]_G| > k$. Let ϱ be a grounding isomorphism for G, let $G' = \{\varrho[t] \mid t \in G\}$, and let W be a ϱ-Web for G. Based on Property 3.2 and on the fact that $|[\![P]\!]_G| > k$, we have $|[\![P]\!]_{G'}| > k$. Then, by Property 3.3, $|\mathcal{Q}^P(W)| > k$. Hence, SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ is unboundedly satisfiable.

Only if: Suppose SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ is unboundedly satisfiable. W.l.o.g., we let $k \in \{0, 1, 2, ...\}$ be an arbitrary natural number. To prove that SPARQL expression P (used by $\mathcal{Q}^P$) is unboundedly satisfiable it suffices to show that there exists a set of RDF triples G such that $|[\![P]\!]_G| > k$. Since SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ is unboundedly satisfiable, there exists a Web of Linked Data W such that $|\mathcal{Q}^P(W)| > k$. Using $\mathcal{Q}^P(W) = [\![P]\!]_{\mathsf{AllData}(W)}$ (cf. Definition 3.1), we note that $\mathsf{AllData}(W)$ is such a set of RDF triples that we need to find for P. Hence, P is unboundedly satisfiable. ■

Proof of Proposition 3.1, Claim 3 (Bounded satisfiability). Claim 3 follows trivially from Claims 1 and 2: Suppose SPARQL expression P is boundedly satisfiable. In this case, P is satisfiable and not unboundedly satisfiable (cf. Section 3.2.1, page 38ff). By Claims 1 and 2, SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ (which uses P) is also satisfiable and not unboundedly satisfiable. Therefore, $\mathcal{Q}^P$ is boundedly satisfiable (cf. Definition 2.8 on page 24). The same argument applies for the other direction of Claim 3. ■

Proof of Proposition 3.1, Claim 4 (Monotonicity). Let $\mathcal{Q}^P$ be a SPARQL$_{\mathsf{LD}}$ query that uses SPARQL expression P.

If: Suppose SPARQL expression P is monotonic. Let W_1, W_2 be an arbitrary pair of Webs of Linked Data such that W_1 is a subweb of W_2. To prove that SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ is monotonic it suffices to show $\mathcal{Q}^P(W_1) \subseteq \mathcal{Q}^P(W_2)$. By Definition 3.1 (cf. page 42), $\mathcal{Q}^P(W_1) = [\![P]\!]_{\mathsf{AllData}(W_1)}$ and $\mathcal{Q}^P(W_2) = [\![P]\!]_{\mathsf{AllData}(W_2)}$. Since W_1 is a subweb of W_2, by Property 1 of Proposition 2.1 (cf. page 21), $\mathsf{AllData}(W_1) \subseteq \mathsf{AllData}(W_2)$. Then, due to the monotonicity of SPARQL expression P, $[\![P]\!]_{\mathsf{AllData}(W_1)} \subseteq [\![P]\!]_{\mathsf{AllData}(W_2)}$. Hence, $\mathcal{Q}^P(W_1) \subseteq \mathcal{Q}^P(W_2)$ and, thus, SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ is monotonic.

Only if: Suppose SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ is monotonic. We distinguish two cases: SPARQL expression P (used by $\mathcal{Q}^P$) is satisfiable or P is not satisfiable. In the latter case, P trivially is monotonic (cf. Property C.1, page 201). Hence, we only have to discuss the first case.

Let G_1, G_2 be an arbitrary pair of sets of RDF triples such that $G_1 \subseteq G_2$. To prove that (the satisfiable) SPARQL expression P is monotonic it suffices to show $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$. Similar to the proof for the other direction, we aim to use G_1 and G_2 for constructing two Webs of Linked Data W_1 and W_2 (where W_1 is a subweb of W_2) and then use the monotonicity of $\mathcal{Q}^P$ for showing the monotonicity of P. However, since G_1 and G_2 may be (countably) infinite we cannot simply construct Webs of Linked Data that consist of single LD documents which contain all RDF triples of G_1 and G_2, respectively. Instead, we have to use the same approach as we use for proving the other claims of Proposition 3.1 (see above). That is, we let $W_2 = (D_2, data_2, adoc_2)$ be a ϱ-Web for G_2 where ϱ is a grounding isomorphism for G_2.

Since $G_1 \subseteq G_2$ we may use ϱ not only for G_2 but also for G_1. In particular, we let $G'_1 = \{\varrho[t] \,|\, t \in G_1\}$. Then, let $W_1 = (D_1, data_1, adoc_1)$ be the induced subweb of W_2 that is defined by $D_1 = \{d \in D_2 \,|\, data_2(d) \subseteq G'_1\}$ (we recall that any induced subweb is unambiguously defined by specifying its set of LD documents; cf. Proposition 2.1, page 21). It can be easily seen that $\mathsf{AllData}(W_1) = G'_1$ and $\mathsf{AllData}(W_2) = G'_2$.

In the following we use both Webs, W_1 and W_2, and the monotonicity of $\mathcal{Q}^P$, to show $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$ (which proves that P is monotonic). W.l.o.g., let μ be an arbitrary solution for P in G_1, that is, $\mu \in [\![P]\!]_{G_1}$. Such a solution μ exists because we assume that P is satisfiable (see above). To prove $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$ it suffices to show $\mu \in [\![P]\!]_{G_2}$.

By Property 3.2 it holds that $\varrho[\mu] \in [\![P]\!]_{G'_1}$. With $\mathsf{AllData}(W_1) = G'_1$ and Definition 3.1 we have $[\![P]\!]_{G'_1} = [\![P]\!]_{\mathsf{AllData}(W_1)} = \mathcal{Q}^P(W_1)$ and, thus, $\varrho[\mu] \in \mathcal{Q}^P(W_1)$. Since W_1 is an (induced) subweb of W_2 and $\mathcal{Q}^P$ is monotonic, we show $\mathcal{Q}^P(W_1) \subseteq \mathcal{Q}^P(W_2)$ and, thus, $\varrho[\mu] \in \mathcal{Q}^P(W_2)$. We now use $\mathsf{AllData}(W_2) = G'_2$ and show $\varrho[\mu] \in [\![P]\!]_{G'_2}$. Finally, we again use Property 3.2 and find $\varrho^{-1}[\varrho[\mu]] \in [\![P]\!]_{G_2}$ and, thus, $\mu \in [\![P]\!]_{G_2}$. ■

The relationships in Proposition 3.1 enable us to carry over our results on basic properties of SPARQL expressions (cf. Appendix C, page 195ff) to SPARQL$_{\mathsf{LD}}$ queries. First, we focus on the (un)decidability of basic properties: For SPARQL expressions the ordinary (Turing-machine-based) decision problems that are related to satisfiability, bounded satisfiability, and monotonicity, are undecidable (see Proposition C.1, C.5, and C.8 on page 195, 201, and 206, respectively). To show the same for SPARQL$_{\mathsf{LD}}$ we introduce corresponding (ordinary) decision problems as follows:

Problem:	SATISFIABILITY(SPARQL$_{\text{LD}}$)
Input:	a SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$
Question:	Is $\mathcal{Q}^P$ satisfiable?

Problem:	MONOTONICITY(SPARQL$_{\text{LD}}$)
Input:	a SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$
Question:	Is $\mathcal{Q}^P$ monotonic?

Problem:	BOUNDEDSATISFIABILITY(SPARQL$_{\text{LD}}$)
Input:	a SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$
Question:	Is $\mathcal{Q}^P$ boundedly satisfiable?

The undecidability of these problems follows readily from the undecidability of their SPARQL counterparts:

Corollary 3.1. SATISFIABILITY(SPARQL$_{\text{LD}}$)*,* MONOTONICITY(SPARQL$_{\text{LD}}$)*, and* BOUNDEDSATISFIABILITY(SPARQL$_{\text{LD}}$) *are undecidable.*

Proof. Based on Proposition 3.1 it is trivial to use the undecidability of SATISFIABILITY(SPARQL) (shown in Proposition C.1 on page 195) to show by reduction that SATISFIABILITY(SPARQL$_{\text{LD}}$) is undecidable. Similarly, the undecidability of MONOTONICITY(SPARQL$_{\text{LD}}$) follows immediately from the undecidability of MONOTONICITY(SPARQL) (shown in Proposition C.5 on page 201), and the undecidability of BOUNDEDSATISFIABILITY(SPARQL$_{\text{LD}}$) follows from the undecidability of BOUNDEDSATISFIABILITY(SPARQL) (shown in Proposition C.8 on page 206). ■

Although we cannot decide the basic properties for SPARQL$_{\mathsf{LD}}$ in general, we may identify certain fragments of SPARQL$_{\mathsf{LD}}$ for which the properties can be shown. However, the relationships in Proposition 3.1 indicate that identifying such fragments is not a SPARQL$_{\mathsf{LD}}$-specific problem. Instead, related results obtained for certain classes of SPARQL expressions (interpreted under the standard SPARQL query semantics) carry over directly to full-Web query semantics. Therefore, we consider a comprehensive discussion of such fragments out of scope of this dissertation. Nonetheless, for the sake of completeness, we identify some such fragments in our discussion of SPARQL expressions in Appendix C. For a summary of our findings we refer to Table C.1 (cf. page 196).

3.3.2. LD Machine Decidability of Termination

We now discuss (non-)termination of SPARQL$_{\mathsf{LD}}$ query computation. In particular, we focus on the question of whether an LD-machine-based computation of a given SPARQL$_{\mathsf{LD}}$ query (over a given Web of Linked Data) may halt with the complete expected query result. If (and only if) this is the case, we have the guarantee that an actual query execution system (for a Web of Linked Data such as the WWW) might completely answer the query within a finite amount of time. We emphasize that any approach to decide about such a termination question in practice, is restricted to the limited data access capabilities captured by our computation model. Therefore, we formally specify the question as an LD decision problem (cf. Definition 2.12, page 30).

LD Problem:	TERMINATION($\text{SPARQL}_{\text{LD}}$)
Web Input:	a Web of Linked Data W
Ordin. Input:	a $\text{SPARQL}_{\text{LD}}$ query $\mathcal{Q}^P$
Question:	Does there exist an LD machine M whose computation, with W encoded on the Web input tape of M, halts with an encoding of query result $\mathcal{Q}^P(W)$ on the output tape?

We shall see that TERMINATION($\text{SPARQL}_{\text{LD}}$) is not LD machine decidable. To obtain this result we first discuss the dependency of termination on information about the satisfiability of a given $\text{SPARQL}_{\text{LD}}$ query.

Suppose a given $\text{SPARQL}_{\text{LD}}$ query is *not* satisfiable: Any unsatisfiable Linked Data query is finitely computable by an LD machine because such a machine may immediately report the empty result (cf. Proposition 2.3, page 29). Consequently, for any $\text{SPARQL}_{\text{LD}}$ query for which an LD machine can decide that this query is not satisfiable, this "decider" machine may immediately answer the termination problem with yes (independent of what the Web input is). Notice, the decider and the LD machine that may perform the (terminating) computation of the unsatisfiable query are different LD machines.

For unboundedly satisfiable $\text{SPARQL}_{\text{LD}}$ queries an LD machine may also immediately answer the termination problem (again, independent of the Web input). However, in this case the answer is always negative as the following proposition shows.

Proposition 3.2. *There does not exist any unboundedly satisfiable $\text{SPARQL}_{\text{LD}}$ query $\mathcal{Q}^P$ for which there exists an LD machine M that, for any Web of Linked Data W encoded on the Web input tape of M, halts after a finite number of computation steps and produces an encoding of query result $\mathcal{Q}^P(W)$ on its output tape.*

Proof. We use proof by contradiction. W.l.o.g., let $\mathcal{Q}^P$ be an arbitrary unboundedly satisfiable $\text{SPARQL}_{\text{LD}}$ query. To obtain a contradiction we assume that there exists an LD machine M that, for any Web of Linked Data W encoded on the Web input tape of M, halts after a finite number of computation steps and produces a possible encoding of query result $\mathcal{Q}^P(W)$ on its output tape.

To compute query $\mathcal{Q}^P$ over an arbitrary Web of Linked Data $W = (D, \mathit{data}, \mathit{adoc})$ (encoded on the Web input tape of M), M requires access to the data of all LD documents $d \in D$. However, M may only access an LD document $d \in D$ (and its data) after, first, writing to the end of its lookup tape a URI $u \in \mathcal{U}$ such that $\mathit{adoc}(u) = d$ and, then, calling its expand procedure. Initially, the machine has no information about which URIs to use for accessing any $d \in D$ (recall that mapping adoc is not available to the machine). Hence, to ensure that all LD documents $d \in D$ have been accessed, M must expand all URI $u \in \mathcal{U}$. Notice, a real query execution system for the WWW would have to perform a similar procedure: To guarantee that such a system sees all documents, it must enumerate and look up all URIs. However, since the set $\mathcal{U}$ is (countably) infinite, this process does not terminate, which is a contradiction to our assumption that M halts after a finite number of computation steps. ■

We now focus on boundedly satisfiable SPARQL$_{\mathsf{LD}}$ queries. For such a query an LD-machine-based computation may terminate as the following example illustrates.

Example 3.3. Let $tp = (u_1^*, u_2^*, u_3^*) \in \mathcal{U} \times \mathcal{U} \times \mathcal{U}$ be a triple pattern (that contains no variables and, hence, also is an RDF triple), and let $\mathcal{Q}^{tp}$ be the SPARQL$_{\mathsf{LD}}$ query that uses this triple pattern. We emphasize that this triple pattern is boundedly satisfiable (cf. Example 3.1, page 41), and so is $\mathcal{Q}^{tp}$ (cf. Proposition 3.1, page 43). An LD machine for $\mathcal{Q}^{tp}$ may take advantage of this fact: As soon as such a machine discovers an LD document whose data contains RDF triple (u_1^*, u_2^*, u_3^*), the machine may return the complete query result $\{\mu_\emptyset\}$ (with $\mathrm{dom}(\mu_\emptyset) = \emptyset$) and halt.

On the other hand, it is also easy to verify that, for any Web of Linked Data W with $(u_1^*, u_2^*, u_3^*) \notin \mathsf{AllData}(W)$, such a machine does not halt. The following proposition generalizes this claim for the empty Web of Linked Data. □

Proposition 3.3. *Let $W_\emptyset = (D_\emptyset, data_\emptyset, adoc_\emptyset)$ be the empty Web of Linked Data, that is, $D_\emptyset = \emptyset$ (and, thus, $adoc_\emptyset(u) = \bot$ for all $u \in \mathcal{U}$). There does not exist any boundedly satisfiable SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ for which there exists an LD machine M that has the following two properties:*

1. *The computation of M on any Web of Linked Data (encoded on the Web input tape of M) has the two properties given in the definition of eventually computable Linked Data queries (cf. Definition 2.11, page 29).*

2. *The computation of M on $W_\emptyset$ (encoded on the Web input tape of M) halts after a finite number of computation steps and outputs an encoding of $\mathcal{Q}^P(W_\emptyset) = \emptyset$.*

Proof. We use proof by contradiction and argue similarly as in the proof of Proposition 3.2. W.l.o.g., let $\mathcal{Q}^P$ be an arbitrary boundedly satisfiable SPARQL$_{\mathsf{LD}}$ query. To obtain a contradiction we assume that there exists an LD machine M that has the two properties given in Proposition 3.3.

Since M exhibits the first property, M actually attempts to compute query $\mathcal{Q}^P$ over any given Web of Linked Data (including $W_\emptyset$). We discuss the case where M attempts to compute $\mathcal{Q}^P$ over $W_\emptyset$ (that is, $W_\emptyset$ is encoded on the Web input tape of M). Recall, an LD machine has no a-priori information about the Web of Linked Data on its Web input tape (unless such information is given as additional input on the ordinary input tape, which is not the case here). Thus, the fact that $D_\emptyset$ is empty is not available to M.

Due to the lack of information about $W_\emptyset$, machine M must assume that there may at least be a single solution for $\mathcal{Q}^P$ in $W_\emptyset$ (because any boundedly satisfiable query is satisfiable; cf. Definition 2.8, page 24). Since M has the first property given in Proposition 3.3, M starts a computation with which it attempts to compute the (nonexistent) solutions for $\mathcal{Q}^P$ in $W_\emptyset$. To halt this computation and return the (correct) empty query result, M needs to ascertain that it has seen the data of all LD documents in $W_\emptyset$. However, as we argue in our proof of Proposition 3.2, a guarantee that an LD machine accessed all LD documents from an input Web is only given after the machine performed its expand procedure for all URIs in $\mathcal{U}$. Such a process cannot terminate because of the infiniteness of $\mathcal{U}$. Therefore, machine M cannot have the second property given in Proposition 3.3, which contradicts our initial assumption. ■

While Example 3.3 illustrates that the (LD-machine-based) computation of boundedly satisfiable SPARQL$_{\mathsf{LD}}$ queries over particular Webs of Linked Data may terminate, Proposition 3.3 shows that for at least one Web of Linked Data such a computation cannot terminate (if it has to guarantee completeness). This observation leads us to a more specific version of problem TERMINATION(SPARQL$_{\text{LD}}$) that is only concerned with boundedly satisfiable SPARQL$_{\mathsf{LD}}$ queries:

LD Problem:	TERMINATION(BS-SPARQL$_{\text{LD}}$)
Web Input:	a Web of Linked Data W
Ordin. Input:	a boundedly satisfiable SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$
Question:	Does there exist an LD machine M whose computation, with W encoded on the Web input tape of M, halts with an encoding of query result $\mathcal{Q}^P(W)$ on the output tape?

We cannot use LD machines to decide TERMINATION(BS-SPARQL$_{\text{LD}}$).

Proposition 3.4. TERMINATION(BS-SPARQL$_{\text{LD}}$) *is not LD machine decidable.*

Proof. We show that TERMINATION(BS-SPARQL$_{\text{LD}}$) is not LD machine decidable by reducing the well-known halting problem to TERMINATION(BS-SPARQL$_{\text{LD}}$). The halting problem asks whether a given Turing machine halts on a given input [150]. For the reduction we construct an infinite Web of Linked Data W_{TMs} that describes all possible computations of all possible Turing machines. For a formal definition of W_{TMs} we adopt the usual approach to unambiguously describe Turing machines and their input by finite words over the (finite) alphabet of a universal Turing machine (e.g., [125]).

Let $\mathcal{W}$ be the countably infinite set of all words that describe Turing machines. For each such word $w \in \mathcal{W}$, let $M(w)$ denote the Turing machine described by w, let $c^{w,x}$ denote the computation of this Turing machine $M(w)$ on input x, and let $u^{w,x}$ denote a URI that identifies $c^{w,x}$. Furthermore, let $u_i^{w,x}$ denote a URI that identifies the i-th step in computation $c^{w,x}$. To denote the (infinite) set of all such URIs (for all Turing machines and all inputs) we write $\mathcal{U}_{\mathsf{TMsteps}}$. Hence, by using these URIs we may unambiguously identify each step in each possible computation of any Turing machine on any given input. However, if a URI $u \in \mathcal{U}$ could potentially identify a computation step of a Turing machine M on some input x (because u adheres to the pattern used for such URIs) but the corresponding step does not exist (because the computation of M on input x terminates after an earlier step), then $u \notin \mathcal{U}_{\mathsf{TMsteps}}$. For instance, if the computation of a particular Turing machine $M(w_j)$ on a particular input x_k halts with the i'-th step, then $u_i^{w_j,x_k} \notin \mathcal{U}_{\mathsf{TMsteps}}$ for all $i \in \{i'+1, i'+2, ...\}$, whereas $u_i^{w_j,x_k} \in \mathcal{U}_{\mathsf{TMsteps}}$ for all $i \in \{1, ..., i'\}$. Notice, while the set $\mathcal{U}_{\mathsf{TMsteps}}$ is infinite, it is still countable because (i) $\mathcal{W}$ is countably infinite, (ii) the set of all possible input words for Turing machines is countably infinite, and (iii) each computation $c^{w,x}$ consists of a countable number of steps (for all $w \in \mathcal{W}$ and any possible input word x).

We now define W_{TMs} as a Web of Linked Data $(D_{\mathsf{TMs}}, data_{\mathsf{TMs}}, adoc_{\mathsf{TMs}})$ such that:

- D_{TMs} consists of $|\mathcal{U}_{\mathsf{TMsteps}}|$ different LD documents, each of which corresponds to one of the URIs in $\mathcal{U}_{\mathsf{TMsteps}}$ (and, thus, to a particular step in a particular computation of a particular Turing machine). For any $u_i^{w,x} \in \mathcal{U}_{\mathsf{TMsteps}}$ we denote the corresponding LD document by $d_i^{w,x}$.

- Mapping $adoc_{\mathsf{TMs}}$ maps each URI in $\mathcal{U}_{\mathsf{TMsteps}}$ to its corresponding LD document; i.e., $adoc_{\mathsf{TMs}}(u_i^{w,x}) := d_i^{w,x}$ for all $u_i^{w,x} \in \mathcal{U}_{\mathsf{TMsteps}}$. For any other URI $u \notin \mathcal{U}_{\mathsf{TMsteps}}$, $adoc_{\mathsf{TMs}}(u) := \bot$.

- Finally, mapping $data_{\mathsf{TMs}}$ is defined as follows: The set of RDF triples for an LD document $d_i^{w,x} \in D_{\mathsf{TMs}}$ is empty if computation $c^{w,x}$ does not halt with the i-th computation step; otherwise, $data_{\mathsf{TMs}}(d_i^{w,x})$ consists of a single RDF triple $(u^{w,x}, \mathsf{type}, \mathsf{TerminatingComputation})$ where $\mathsf{type}, \mathsf{TerminatingComputation} \in \mathcal{U}$. Formally:

$$data_{\mathsf{TMs}}(d_i^{w,x}) := \begin{cases} \big\{(u^{w,x}, \mathsf{type}, \mathsf{TerminatingComputation})\big\} & \text{if computation } c^{w,x} \text{ halts} \\ & \text{with the } i\text{-th step,} \\ \emptyset & \text{else.} \end{cases}$$

We emphasize that mappings $data_{\mathsf{TMs}}$ and $adoc_{\mathsf{TMs}}$ are Turing computable because a universal Turing machine may determine by simulation whether the computation of a particular Turing machine on a particular input halts before a given number of steps.

We now reduce the halting problem to TERMINATION(BS-SPARQL$_{\text{LD}}$). The input to the halting problem is a pair (w, x) consisting of a Turing machine description w and a possible input word x. For the reduction we need a Turing computable function f that, given such a pair (w, x), produces a tuple $(W, \mathcal{Q}^P)$ as input for TERMINATION(BS-SPARQL$_{\text{LD}}$). We define f as follows: Let (w, x) be an input to the halting problem, then $f(w, x) := (W_{\mathsf{TMs}}, \mathcal{Q}^{P_{w,x}})$ where SPARQL expression $P_{w,x}$ is the triple pattern $(u^{w,x}, \mathsf{type}, \mathsf{TerminatingComputation})$. Notice, $\text{vars}(P_{w,x}) = \emptyset$ and, thus, by Corollary C.1 (cf. page 209) and Proposition 3.1 (cf. page 43), SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^{P_{w,x}}$ is boundedly satisfiable. Given that W_{TMs} is independent of (w, x), it is easy to see that f is computable by a Turing machine (and, thus, by an LD machine).

Then, the reduction is based on the following claim: For any possible (halting problem) input (w, x), and the corresponding $(W_{\mathsf{TMs}}, \mathcal{Q}^{P_{w,x}}) = f(w, x)$, it holds that Turing machine $M(w)$ halts on input x if and only if there exists an LD machine that computes $\mathcal{Q}^{P_{w,x}}(W_{\mathsf{TMs}}) = \{\mu_\emptyset\}$ and halts. The claim is easily verified based on the following two observations:

1. Let (w, x) be an input to the halting problem and let $(W_{\mathsf{TMs}}, \mathcal{Q}^{P_{w,x}}) = f(w, x)$. Due to the definition of Web of Linked Data W_{TMs} and query $\mathcal{Q}^{P_{w,x}}$, it holds that

$$\mathcal{Q}^{P_{w,x}}(W_{\mathsf{TMs}}) = \begin{cases} \{\mu_\emptyset\} & \text{if the computation of Turing machine } M(w) \\ & \text{on input } x \text{ halts,} \\ \emptyset & \text{else,} \end{cases}$$

where $\mu_\emptyset$ is the empty valuation with $\mathrm{dom}(\mu_\emptyset) = \emptyset$.

2. Let (w, x) be an input to the halting problem and let $\big(W_{\mathsf{TMs}}, \mathcal{Q}^{P_{w,x}}\big) = f(w, x)$. If Turing machine $M(w)$ halts on input x, then there exists an LD machine, say $M_{w,x}$, whose computation on Web input W_{TMs} halts eventually with the query result $\mathcal{Q}^{P_{w,x}}(W_{\mathsf{TMs}}) = \{\mu_\emptyset\}$ on the output tape of $M_{w,x}$. For instance, such a machine may enter its expand state successively for all URIs $u_1^{w,x}, u_2^{w,x}, \ldots$ until the RDF triple $(u^{w,x}, \mathsf{type}, \mathsf{TerminatingComputation})$ appears on the lookup tape of the machine; at this point the machine writes $\mathrm{enc}(\mu_\emptyset)$ to its output tape and halts.

To show that Termination(BS-SPARQL$_{\text{LD}}$) is not LD machine decidable, suppose it is LD machine decidable and LD machine M is a decider. Then, due to the aforementioned claim, LD machine M may also be used to answer the halting problem for any input (w, x). However, we know the halting problem is undecidable for Turing machines. Therefore, it is also undecidable for LD machines (which are Turing machines). Hence, we have a contradiction and, thus, Termination(BS-SPARQL$_{\text{LD}}$) cannot be LD machine decidable. ■

Given Proposition 3.4 it is now trivial to show that the general termination problem for SPARQL$_{\mathsf{LD}}$ is not decidable for LD machines.

Theorem 3.1. Termination(SPARQL$_{\text{LD}}$) *is not LD machine decidable.*

Proof. We reduce Termination(BS-SPARQL$_{\text{LD}}$) to Termination(SPARQL$_{\text{LD}}$) using the identity mapping. Given that Termination(BS-SPARQL$_{\text{LD}}$) is not LD machine decidable it is easily shown by reduction that Termination(SPARQL$_{\text{LD}}$) is not LD machine decidable either. ■

In our proof of Theorem 3.1 we use the fact that problem Termination(SPARQL$_{\text{LD}}$) includes boundedly satisfiable SPARQL$_{\mathsf{LD}}$ queries as possible input. We emphasize, however, that if we would explicitly rule out the special case of boundedly satisfiable queries in our definition of Termination(SPARQL$_{\text{LD}}$), the problem would still be undecidable for LD machines: Given a Web of Linked Data W and a SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ that is not boundedly satisfiable, we may easily use Proposition 2.3 (cf. page 29) and Proposition 3.2 (cf. page 48) to show that there exists an LD machine that computes $\mathcal{Q}^P(W)$ and halts if and only if $\mathcal{Q}^P$ is not satisfiable. However, the satisfiability of SPARQL$_{\mathsf{LD}}$ queries is undecidable (cf. Corollary 3.1 on page 47). Thus, we could show by reduction from the satisfiability problem that the adjusted termination problem (which excludes boundedly satisfiable queries) is still not LD machine decidable.

3.3.3. LD Machine Computability

This section discusses LD-machine-based computability of SPARQL$_{\mathsf{LD}}$ queries. Since all unsatisfiable SPARQL$_{\mathsf{LD}}$ queries are finitely computable by an LD machine (cf. Proposition 2.3, page 29), we focus on satisfiable SPARQL$_{\mathsf{LD}}$ queries. The following result shows that the computability of such queries is correlated with their monotonicity.

Theorem 3.2. *If a satisfiable SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ is monotonic, then $\mathcal{Q}^P$ is eventually computable by an LD machine (but not finitely computable); otherwise, $\mathcal{Q}^P$ is not eventually computable by an LD machine.*

To prove Theorem 3.2 we have to show that any monotonic SPARQL$_{\mathsf{LD}}$ query is eventually computable by an LD machine. For this proof we introduce a specific type of LD machine which we call *full-Web machine*. Such a machine exists for any SPARQL$_{\mathsf{LD}}$ query. We shall see that if a satisfiable SPARQL$_{\mathsf{LD}}$ query is monotonic, the corresponding full-Web machine for this query (eventually) computes the query over any given Web of Linked Data without terminating (i.e., the machine satisfies the conditions of eventual computability as given in Definition 2.11, page 29). In the following, we define the full-Web machine, show properties of this machine that are relevant for proving Theorem 3.2, and, afterwards, prove the theorem.

Definition 3.4 (Full-Web Machine). Let P be SPARQL expression. The *full-Web machine* for P is an LD machine (as per Definition 2.9, page 27) that implements Algorithm 3.1. This algorithm makes use of a special subroutine called *lookup*. This subroutine, when called with URI $u \in \mathcal{U}$, (i) writes enc(u) to the right end of the word on the lookup tape, (ii) enters the expand state, and (iii) performs the expand procedure as specified in Definition 2.9. □

Algorithm 3.1 Program of the full-Web machine for SPARQL expression P.

```
1: $j := 1$
2: **for all** $u \in \mathcal{U}$ **do**
3:     Call subroutine *lookup* for $u$.
4:     Use the work tape to enumerate the set of valuations $[\![P]\!]_{T_j}$ where $T_j$ denotes the
       set of all RDF triples currently encoded on the lookup tape.
5:     For each valuation $\mu \in [\![P]\!]_{T_j}$ check whether $\mu$ is already encoded on the output
       tape; if not, then append enc($\mu$) to word on the output tape.
6:     $j := j + 1$
7: **end for**
```

In Algorithm 3.1 we see that any computation performed by a full-Web machine enters a loop that iterates over the set of all possible URIs. As discussed before (see our proofs of Propositions 3.2 and 3.3), expanding all URIs $u \in \mathcal{U}$ is necessary to access all LD documents in the queried Web of Linked Data and, thus, to guarantee completeness of the computed query result. However, since $\mathcal{U}$ is (countably) infinite, the algorithm does not terminate (which is not a requirement for eventual computability; cf. Definition 2.11, page 29).

During each iteration of its main processing loop, a full-Web machine generates valuations using all data that is currently encoded on its lookup tape. The following lemma shows that these valuations are part of the corresponding query result.

Lemma 3.1. *Let $\mathcal{Q}^P$ be a satisfiable SPARQL$_{\mathsf{LD}}$ query that is monotonic; let M^P denote the full-Web machine for the SPARQL expression P used by $\mathcal{Q}^P$; and let W be an arbitrary Web of Linked Data encoded on the Web input tape of M^P. During the execution of Algorithm 3.1 by M^P, $[\![P]\!]_{T_j} \subseteq \mathcal{Q}^P(W)$ holds for all $j \in \{1, 2, \ldots\}$.*

For the proof of Lemma 3.1—and of any other auxiliary result in this thesis—we refer to Appendix E (in particular, the proof of Lemma 3.1 is given in Section E.1, page 219).

Lemma 3.1 provides the basis for showing that the computation of a full-Web machine has the first property required for eventual computability (cf. Definition 2.11, page 29). To verify that the computation also has the second property in Definition 2.11, it is important to note that Algorithm 3.1 looks up no more than a single URI per iteration (cf. line 3). Hence, a full-Web machines prioritizes result construction over data retrieval. This feature allows us to show that for each solution in an expected query result, there exists an iteration during which that solution is computed.

Lemma 3.2. *Let $\mathcal{Q}^P$ be a satisfiable SPARQL$_{\mathsf{LD}}$ query that is monotonic; let M^P denote the full-Web machine for the SPARQL expression P used by $\mathcal{Q}^P$; and let W be an arbitrary Web of Linked Data encoded on the Web input tape of M^P. For each solution $\mu \in \mathcal{Q}^P(W)$ there exists a $j_\mu \in \{1, 2, \ldots\}$ such that during the execution of Algorithm 3.1 by M^P, $\mu \in [\![P]\!]_{T_j}$ holds for all $j \in \{j_\mu, j_\mu{+}1, \ldots\}$.*

The proof of Lemma 3.2 can be found in the appendix (cf. Section E.2, page 219ff).

We are now ready to prove Theorem 3.2 (cf. page 53).

Proof of Theorem 3.2. The proof consists of three parts: First, we show that satisfiable, monotonic SPARQL$_{\mathsf{LD}}$ queries are eventually computable by an LD machine. Next, we prove that there does not exist any satisfiable SPARQL$_{\mathsf{LD}}$ query that is finitely computable by an LD machine. Finally, we focus on SPARQL$_{\mathsf{LD}}$ queries that are not monotonic and show that these queries are not eventually computable by an LD machine.

Part 1: To show that satisfiable, monotonic SPARQL$_{\mathsf{LD}}$ queries are eventually computable by an LD machine we use the aforementioned full-Web machine. Let $\mathcal{Q}^{P^*}$ be a satisfiable SPARQL$_{\mathsf{LD}}$ query that is monotonic and let W^* be an arbitrary Web of Linked Data encoded on the Web input tape of the full-Web machine for P^* (where P^* is the SPARQL expression used by $\mathcal{Q}^{P^*}$). To denote this machine we write M^{P^*}. W.l.o.g. it suffices to show that the computation of M^{P^*} on Web input enc(W^*) has the two properties required for eventual computability (cf. Definition 2.11, page 29).

During the computation, machine M^{P^*} only writes to its output tape when it appends (encoded) valuations $\mu \in [\![P^*]\!]_{T_j}$ (for $j = 1, 2, \ldots$). Since all these valuations are solutions for $\mathcal{Q}^{P^*}$ in W^* (cf. Lemma 3.1, page 54) and line 5 in Algorithm 3.1 ensures that the output is free of duplicates, the word on the output tape of M^{P^*} is always a prefix of a possible encoding of $\mathcal{Q}^{P^*}(W^*)$. Hence, the computation of M^{P^*} has the first property required for eventual computability.

The second property requires that the encoding $\text{enc}(\mu')$ of any solution $\mu' \in \mathcal{Q}^{P^*}(W^*)$ becomes part of the word on the output tape of M^{P^*} after a finite number of computation steps. From Lemma 3.2 (cf. page 54) we know that for any such solution there exists an iteration of the loop in Algorithm 3.1 during which M^{P^*} computes the solution (for the first time). In this iteration, machine M^{P^*} appends the encoding of the solution to its output tape (cf. line 5 in Algorithm 3.1).

It remains to show that the computation of M^{P^*} definitely reaches each iteration of the loop after a finite number of computation steps. To prove this property we show that each iteration of the loop finishes after a finite number of computation steps:

- The call of subroutine *lookup* (cf. Definition 3.4) at line 3 in Algorithm 3.1 terminates because the encoding of W^* on the Web input tape of M^{P^*} is ordered (cf. Appendix B, page 193f).

- At any point in the computation the word on the lookup tape of M^{P^*} is finite because M^{P^*} only gradually appends (encoded) LD documents to the lookup tape and the encoding of each document is finite (because the set of RDF triples $data(d)$ for each LD document $d \in D$ in any Web of Linked Data $W = (D, data, adoc)$ is finite). Due to the finiteness of the word on the lookup tape, each set of valuations $[\![P^*]\!]_{T_j}$ (for $j = 1, 2, ...$) is finite, resulting in a finite number of computation steps for line 4 during any iteration.

- Finally, line 5 requires only a finite number of computation steps because the word on the lookup tape of M^{P^*} is finite at any point in the computation, and so is the word on the output tape.

Part 2: We now show that there does not exist any satisfiable SPARQL$_{\mathsf{LD}}$ query that is finitely computable by an LD machine. By Definition 2.10 (cf. page 29), a Linked Data query Q is finitely computable by an LD machine if there exists an LD machine that, for any Web of Linked Data W encoded on its Web input tape, halts after a finite number of computation steps and produces a possible encoding of the expected query result $Q(W)$ on its output tape. By Proposition 3.2 (cf. page 48) and Proposition 3.3 (cf. page 49), such a machine does not exist for any satisfiable SPARQL$_{\mathsf{LD}}$ query (which may either be boundedly satisfiable or unboundedly satisfiable; cf. Definition 2.8, page 24).

Part 3: In the remainder of this proof, we show that non-monotonic SPARQL$_{\mathsf{LD}}$ queries are not eventually computable by an LD machine. Let $\mathcal{Q}^P$ be an arbitrary non-monotonic SPARQL$_{\mathsf{LD}}$ query. W.l.o.g., we show that $\mathcal{Q}^P$ is not eventually computable by an LD machine. As in our proofs of Propositions 3.2 and 3.3, we use proof by contradiction. To obtain a contradiction we assume that there exists an LD machine, say M, whose computation of $\mathcal{Q}^P$ on any Web of Linked Data W has the two properties required for eventual computability (cf. Definition 2.11, page 29).

Let W_1 and W_2 be two Webs of Linked Data such that (i) W_1 is a subweb of W_2 and (ii) $\mathcal{Q}^P(W_1) \nsubseteq \mathcal{Q}^P(W_2)$ (such a pair of Webs exists because query $\mathcal{Q}^P$ is non-monotonic). Then, there exists a solution $\mu \in \mathcal{Q}^P(W_1)$ such that $\mu \notin \mathcal{Q}^P(W_2)$.

Consider the computation of LD machine M on Web input enc(W_1). Based on our assumption, machine M writes enc(μ) to its output tape after a finite number of computation steps (cf. property 2 in Definition 2.11). We argue that this is impossible: Since query $\mathcal{Q}^P$ is not monotonic, M may not add μ to the output until it is guaranteed that M accessed all LD documents in the queried Web of Linked Data W_1. As discussed before, such a guarantee requires expanding all URIs $u \in \mathcal{U}$ because M has no a-priory information about W_1. However, expanding all $u \in \mathcal{U}$ is a non-terminating process (due to the infiniteness of $\mathcal{U}$) and, thus, M does not write μ to its output after a finite number of steps. As a consequence, the computation of $\mathcal{Q}^P(W)$ by M does not have property 2 given in Definition 2.11, which contradicts our initial assumption. This contradiction shows that non-monotonic SPARQL$_{\mathsf{LD}}$ queries are not eventually computable by an LD machine (which concludes our proof of Theorem 3.2, page 53). ■

Theorem 3.2 shows a direct correlation between the monotonicity and the (LD-machine-based) computability of satisfiable SPARQL$_{\mathsf{LD}}$ queries. Furthermore, it shows that not any satisfiable SPARQL$_{\mathsf{LD}}$ query is finitely computable. Our proof of the theorem reveals that the reason for this limitation is the infiniteness of the set of all URIs $\mathcal{U}$. Hence, even if our data model would allow only for Webs of Linked Data that are finite, we would obtain the same result as given by Theorem 3.2.

We conclude our discussion of theoretical properties of SPARQL$_{\mathsf{LD}}$ by showing that (LD-machine-based) computability is undecidable for SPARQL$_{\mathsf{LD}}$. As a basis we introduce the following (ordinary) decision problems:

Problem:	FiniteComputability(SPARQL$_{\text{LD}}$)
Input:	a SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$
Question:	Is $\mathcal{Q}^P$ finitely computable by an LD machine?

Problem:	EventualComputability(Sat-SPARQL$_{\text{LD}}$)
Input:	a SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ that is satisfiable
Question:	Is $\mathcal{Q}^P$ eventually computable by an LD machine?

Based on the fact that satisfiability and monotonicity are undecidable for SPARQL$_{\mathsf{LD}}$ we show the aforementioned result:

Corollary 3.2. FiniteComputability(SPARQL$_{\text{LD}}$) *and* EventualComputability(Sat-SPARQL$_{\text{LD}}$) *are undecidable.*

Proof. By Theorem 3.2 and based on the fact that unsatisfiable SPARQL$_{\mathsf{LD}}$ queries are finitely computable (cf. Proposition 2.3, page 29), it follows that any SPARQL$_{\mathsf{LD}}$ query is finitely computable if and only if this query is not satisfiable. Given the undecidability of Satisfiability(SPARQL$_{\text{LD}}$) (shown in Corollary 3.1, page 47), it is trivial to show by reduction that FiniteComputability(SPARQL$_{\text{LD}}$) is undecidable. Similarly, the undecidability of EventualComputability(Sat-SPARQL$_{\text{LD}}$) follows readily from Theorem 3.2 and the fact that Monotonicity(SPARQL$_{\text{LD}}$) is undecidable (also shown in Corollary 3.1). ■

3.3.4. Finiteness of Expected Query Results

The limited computational feasibility of SPARQL$_{\mathsf{LD}}$ queries that our results in the previous sections show is a consequence of the infiniteness of the set of all URIs. As a consequence, a possible infiniteness of queried Webs of Linked Data has no impact on the computational feasibility of SPARQL$_{\mathsf{LD}}$ queries. Nonetheless, we are interested in the implications of allowing for an infinite Web of Linked Data in our data model. Therefore, in this section we discuss the impact of infiniteness on SPARQL$_{\mathsf{LD}}$ queries . In particular, we focus on the (expected) query results of such queries. In the following, we first assume a finite Web of Linked Data; for the sake of completeness, we formally prove that the result of any SPARQL$_{\mathsf{LD}}$ query over such a Web is finite. A similarly general statement does not exist when the queried Web of Linked Data is infinite. As a consequence, we then discuss the decision problem that is related to the finiteness of expected query results.

As mentioned before, we first show that the result of any SPARQL$_{\mathsf{LD}}$ query over a *finite* Web of Linked Data is finite:

Proposition 3.5. *For any SPARQL$_{LD}$ query $\mathcal{Q}^P$ and any Web of Linked Data W, query result $\mathcal{Q}^P(W)$ is finite if W is finite.*

Proof. Let $W = (D, data, adoc)$ be a Web of Linked Data that is finite, and let $G_W = \mathsf{AllData}(W)$. Given that $\mathcal{Q}^P(W) = [\![P]\!]_{G_W}$ for any SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ (cf. Definition 3.1, page 42), we prove the proposition by showing that $[\![P]\!]_{G_W}$ is finite for any SPARQL expression P. We use an induction on the structure of SPARQL expressions.

Base case: Suppose SPARQL expression P is a triple pattern tp. In this case we have:

$$[\![P]\!]_{G_W} = \{\mu \,|\, \mu \text{ is a valuation with } \mathrm{dom}(\mu) = \mathrm{vars}(tp) \text{ and } \mu[tp] \in G_W\}\,.$$

Since W (and, thus, D) is finite and, for each LD document $d \in D$, the set of RDF triples $data(d)$ is finite, we only have a finite number of RDF triples in $G_W = \bigcup_{d \in D} data(d)$. Hence, the number of all possible valuations $\mu \in [\![P]\!]_{G_W}$ (with $\mathrm{dom}(\mu) = \mathrm{vars}(tp)$ and $\mu[tp] \in G_W$) is finite and, thus, $[\![P]\!]_{G_W}$ must be finite.

Induction step: Let P_1 and P_2 be SPARQL expressions such that $[\![P_1]\!]_{G_W}$ and $[\![P_2]\!]_{G_W}$ is finite, respectively. We show that $[\![P]\!]_{G_W}$ is finite for any SPARQL expression P that can be constructed using P_1 and P_2. Thus, we have to consider four cases that we summarize in the following table (in which R denotes an arbitrary filter condition):

P	$[\![P]\!]_{G_W}$	Maximum possible result cardinality
$(P_1 \text{ AND } P_2)$	$[\![P_1]\!]_{G_W} \bowtie [\![P_2]\!]_{G_W}$	$\lvert[\![P_1]\!]_{G_W}\rvert \cdot \lvert[\![P_2]\!]_{G_W}\rvert$
$(P_1 \text{ UNION } P_2)$	$[\![P_1]\!]_{G_W} \cup [\![P_2]\!]_{G_W}$	$\lvert[\![P_1]\!]_{G_W}\rvert + \lvert[\![P_2]\!]_{G_W}\rvert$
$(P_1 \text{ OPT } P_2)$	$[\![P_1]\!]_{G_W} ⟕ [\![P_2]\!]_{G_W}$	$\lvert[\![P_1]\!]_{G_W}\rvert \cdot \lvert[\![P_2]\!]_{G_W}\rvert$
$(P_1 \text{ FILTER } R)$	$\{\mu \in [\![P_1]\!]_{G_W} \,\vert\, \mu \text{ satisfies } R\}$	$\lvert[\![P_1]\!]_{G_W}\rvert$

For all four cases the maximum possible result cardinality (third column in the table) is a finite number because $[\![P_1]\!]_{G_W}$ and $[\![P_2]\!]_{G_W}$ are finite, respectively. ■

The following example illustrates that a similarly general statement as in Proposition 3.5 does not exist when the queried Web of Linked Data is infinite such as the WWW.

Example 3.4. Let $W_{\mathsf{inf}} = (D_{\mathsf{inf}}, data_{\mathsf{inf}}, adoc_{\mathsf{inf}})$ be an infinite Web of Linked Data that contains LD documents for all integers (similar to the documents for natural numbers in Example 1.2 on page 3). The data in these documents refers to the predecessor and to the successor of the corresponding integer. Hence, for each integer $i \in \mathbb{Z}$, identified by URI $\mathsf{no}_i \in \mathcal{U}$, there exists an LD document $d_i \in D_{\mathsf{inf}}$ such that $adoc_{\mathsf{inf}}(\mathsf{no}_i) = d_i$ and

$$data_{\mathsf{inf}}(d_i) = \big\{(\mathsf{no}_i, \mathsf{pred}, \mathsf{no}_{i-1}), (\mathsf{no}_i, \mathsf{succ}, \mathsf{no}_{i+1})\big\}$$

where URIs $\mathsf{pred} \in \mathcal{U}$ and $\mathsf{succ} \in \mathcal{U}$ identify the predecessor relation and the successor relation for integers, respectively.

Furthermore, let tp_1 and tp_2 be triple pattern $(\mathsf{no}_0, \mathsf{succ}, ?v)$ and $(?x, \mathsf{succ}, ?y)$, respectively. The result of SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^{tp_1}$ over W_{inf} is finite; it consists of a single solution: $\mathcal{Q}^{tp_1}(W_{\mathsf{inf}}) = \{\{?v \to \mathsf{no}_1\}\}$. In contrast, the result of SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^{tp_2}$ over W_{inf} is infinite: $\mathcal{Q}^{tp_2}(W_{\mathsf{inf}}) = \big\{\,\dots, \{?x \to \mathsf{no}_{\text{-}1}, ?y \to \mathsf{no}_0\}, \{?x \to \mathsf{no}_0, ?y \to \mathsf{no}_1\}, \dots\big\}$. □

The example illustrates that some SPARQL$_{\mathsf{LD}}$ queries have a finite result over an infinite Web of Linked Data, whereas other SPARQL$_{\mathsf{LD}}$ queries have an infinite result. Consequently, we are interested in the following LD decision problem:

LD Problem:	FINITENESS(SPARQL$_{\text{LD}}$)
Web Input:	a (potentially infinite) Web of Linked Data W
Ordin. Input:	a SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$
Question:	Is the expected query result $\mathcal{Q}^P(W)$ finite?

We note that the result of any unsatisfiable SPARQL$_{\mathsf{LD}}$ query is trivially finite in any Web of Linked Data. Hence, for these unsatisfiable queries an LD machine may immediately answer the question posed by FINITENESS(SPARQL$_{\text{LD}}$). However, if we take satisfiable SPARQL$_{\mathsf{LD}}$ queries into account, the problem is undecidable for LD machines:

Theorem 3.3. FINITENESS(SPARQL$_{\text{LD}}$) *is not LD machine decidable.*

Proof. We reduce the halting problem to FINITENESS(SPARQL$_{\text{LD}}$). While this proof is similar to the proof of Proposition 3.4 (cf. page 50), we use a different Web of Linked Data. This Web, denoted by W_{TMs2}, differs from W_{TMs} (used in the proof of Proposition 3.4) in the way it describes all possible computations of all Turing machines.

For the proof we use the same symbols as in the proof of Proposition 3.4. That is, $\mathcal{W}$ denotes the countably infinite set of all words that describe Turing machines. For each such word $w \in \mathcal{W}$, $M(w)$ denotes the Turing machine described by w, $c^{w,x}$ denotes the computation of machine $M(w)$ on input x, and URI $u_i^{w,x} \in \mathcal{U}$ identifies the i-th step in computation $c^{w,x}$. The set of all these identifiers is denoted by $\mathcal{U}_{\mathsf{TMsteps}}$. $\mathcal{U}_{\mathsf{TMsteps}}$ is countably infinite.

We now define W_{TMs2} as a Web of Linked Data $(D_{\mathsf{TMs2}}, data_{\mathsf{TMs2}}, adoc_{\mathsf{TMs2}})$ similar to the Web of Linked Data W_{TMs} used in the proof of Proposition 3.4: D_{TMs2} and

$adoc_{\mathsf{TMs2}}$ are the same is in W_{TMs}. That is, D_{TMs2} consists of $|\mathcal{U}_{\mathsf{TMsteps}}|$ different LD documents, each of which corresponds to one of the URIs in $\mathcal{U}_{\mathsf{TMsteps}}$. Mapping $adoc_{\mathsf{TMs2}}$ maps each URI $u_i^{w,x} \in \mathcal{U}_{\mathsf{TMsteps}}$ to the corresponding LD document $d_i^{w,x} \in D_{\mathsf{TMs2}}$ and any other URI to $\perp$. Mapping $data_{\mathsf{TMs2}}$ for W_{TMs2} is different from the corresponding mapping for W_{TMs}: For *each* LD document $d_i^{w,x} \in D_{\mathsf{TMs2}}$, the corresponding set of RDF triples $data_{\mathsf{TMs2}}(d_i^{w,x})$ contains a single RDF triple $(u_i^{w,x}, \mathsf{first}, u_1^{w,x})$ which associates the i-th computation step (identified by $u_i^{w,x}$) with the first step of the corresponding computation $c^{w,x}$ ($\mathsf{first} \in \mathcal{U}$ denotes a URI for this relationship).

Before we come to the reduction we highlight a property of W_{TMs2} that is important for our proof. Each RDF triple of the form $(u_i^{w,x}, \mathsf{first}, u_1^{w,x})$ establishes a data link from LD document $d_i^{w,x}$ to document $d_1^{w,x}$. Hence, the link graph of W_{TMs2} consists of an infinite number of separate subgraphs, each of which (i) corresponds to a particular computation $c^{w,x}$, (ii) is weakly connected, and (iii) has a star-like form in which the LD document $d_1^{w,x}$ is the center of the star. More precisely, the subgraph that corresponds to a computation c^{w_j,x_k} is a directed graph $(V^{w_j,x_k}, E^{w_j,x_k})$ such that:

$$V^{w_j,x_k} = \{ d_i^{w,x} \in D_{\mathsf{TMs2}} \,|\, w = w_j \text{ and } x = x_k \} \quad \text{and} \quad E^{w_j,x_k} = V^{w_j,x_k} \times \{ d_1^{w_j,x_k} \} .$$

Each of these subgraphs is infinitely large (i.e., has an infinite number of vertices) if and only if the corresponding computation halts.

For each Turing machine description w and each possible input word x for Turing machine $M(w)$, let $tp_{w,x}$ denote the triple pattern $(?v, \mathsf{first}, u_1^{w,x})$ where $?v \in \mathcal{V}$ is an arbitrary query variable. Then, the following property is easily verified: Turing machine $M(w)$ halts on input x if and only if the result of SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^{tp_{w,x}}$ (that uses triple pattern $tp_{w,x}$) over W_{TMs2} is finite.

For the reduction we use mapping f that is defined as follows: Let w be the description of a Turing machine $M(w)$ and let x be a possible input word for $M(w)$, then $f(w,x) = (W_{\mathsf{TMs2}}, \mathcal{Q}^{P_{w,x}})$. Given that W_{TMs2} is independent of (w,x), it is easy to see that f can be computed by a Turing machine (and, thus, by an LD machine).

To show that FINITENESS(SPARQL$_{\text{LD}}$) is not LD machine decidable, suppose it is LD machine decidable. In such a case an LD machine could answer the halting problem for any input (w,x) because: Turing machine $M(w)$ halts on input x if and only if $\mathcal{Q}^{P_{w,x}}(W_{\mathsf{TMs2}})$ is finite. However, we know the halting problem is undecidable for Turing machines, including LD machines. Hence, we have a contradiction and, thus, FINITENESS(SPARQL$_{\text{LD}}$) cannot be LD machine decidable. ■

3.4. Summary

This chapter introduces a query semantics, called full-Web semantics, that allows us to use SPARQL expressions as queries over a Web of Linked Data. The scope of evaluating a SPARQL expression under this semantics is the complete set of all data in the queried Web. After providing a formal definition of this type of Linked Data queries, which we call SPARQL$_{\mathsf{LD}}$ queries, we analyzed theoretical properties of SPARQL$_{\mathsf{LD}}$.

Our analysis verifies formally the common assumption that the computational feasibility of queries under a full-Web semantics is limited. In particular, our main result shows that not any satisfiable SPARQL$_{\mathsf{LD}}$ query is finitely computable by an LD machine. Furthermore, non-monotonic SPARQL$_{\mathsf{LD}}$ queries are not even eventually computable by an LD machine. As the reason for this limitation we identify the infiniteness of the set of all URIs. This finding shows that allowing for an infinite Web of Linked Data in our data model has no impact on the computational feasibility of SPARQL$_{\mathsf{LD}}$ queries.

4. Reachability-Based Query Semantics

The full-Web query semantics discussed in the previous chapter is an initial, straightforward approach to use SPARQL as a language for expressing Linked Data queries. Our results show that the computational feasibility of $\mathrm{SPARQL}_{\mathsf{LD}}$ queries (under full-Web semantics) is very limited. Consequently, any execution approach for such queries requires some ad hoc mechanism to abort query executions; we have to accept a potentially incomplete answer for any (satisfiable) $\mathrm{SPARQL}_{\mathsf{LD}}$ query. Furthermore, depending on the abort mechanism, query execution may even be nondeterministic; that is, executing a query multiple times (over the same Web of Linked Data) may result in different answers. Since a query semantics that inherently gives rise to such issues is undesirable for various reasons, we are interested in alternative semantics for SPARQL-based Linked Data queries. In particular, we are interested in semantics that are closer to the capabilities of systems captured by our computation model (i.e., systems for executing Linked Data queries over an implementation of a Web of Linked Data such as the WWW).

In this chapter we discuss a family of such query semantics which we call *reachability-based query semantics*. Informally, the scope of a query under such a semantics is the set of all data in a well-defined, reachable subweb of the queried Web of Linked Data (a formal definition follows shortly). We emphasize that this approach still allows systems to make use of data from initially unknown data sources and, thus, enables applications to tap the full potential of the Web. Each of the different reachability-based query semantics applies a specific notion of reachability. To this end, we shall provide a generic definition that does not prescribe a single possible notion of reachability. Instead, our definition introduces the concept of a reachability criterion. By using such a criterion the notion of reachability can be made explicit for each query.

This chapter is organized as follows: Section 4.1 introduces formally our notion of reachability-based query semantics for SPARQL. Section 4.2 compares query results under full-Web query semantics and under different reachability-based query semantics. Section 4.3 provides a detailed discussion of reachability criteria. Section 4.4 analyzes theoretical properties of queries under reachability-based semantics. Analogously to the analysis for full-Web semantics in the previous chapter, we focus on basic properties (such as satisfiability and monotonicity) and on computation-related properties. Finally, Section 4.5 concludes our analysis by comparing properties shown for full-Web query semantics with those for reachability-based query semantics.

4.1. Definition

The definition of reachability-based query semantics is based on a two-step approach: First, we define the subweb of a (queried) Web of Linked Data that is potentially reachable when traversing specific data links using seed URIs given as part of a query as a starting point. Then, we formalize the result of such a query as the set of all valuations that map the query to a subset of all data in the reachable subweb of the queried Web. While this two-step definition approach provides for a straightforward presentation of the query semantics, an actual execution of a query (interpreted under the defined semantics) is not required to apply a corresponding two-step execution approach.

4.1.1. Reachability

The basis of any reachability-based query semantics is a notion of reachability of LD documents. Informally, an LD document is reachable in a Web of Linked Data if there exists a (specific) path in the link graph of the Web to that document. The potential starting points for such a path are given as a set of seed URIs. However, allowing for arbitrary paths might not be feasible in practice because this approach requires following *all* data links (recursively) for answering a query completely. A more restrictive approach is the notion of *query-based reachability*: a data link qualifies as a part of paths to reachable LD documents only if that link corresponds to a triple pattern in the executed query. However, other criteria for specifying which data links qualify might prove to be more suitable for certain cases. For this reason, we do not prescribe a specific criterion. Instead, we enable supporting any possible criterion by parameterizing this choice.

Definition 4.1 (Reachability Criterion). Let $\mathcal{T}$, $\mathcal{U}$, and $\mathcal{P}$ denote the infinite sets of all possible RDF triples, URIs, and SPARQL expressions, respectively. A *reachability criterion* is a (Turing) computable function $c : \mathcal{T} \times \mathcal{U} \times \mathcal{P} \rightarrow \{\text{true}, \text{false}\}$. □

An example for a reachability criterion is c_{All} which corresponds to the aforementioned approach of allowing for arbitrary paths to reach LD documents; hence, $c_{\mathsf{All}}(t, u, P)$ is defined to be true for every tuple $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$. The complement of c_{All} is c_{None} which returns false for every tuple $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$. Another alternative is c_{Match} which corresponds to the aforementioned query-based reachability. We define c_{Match} based on the notion of matching RDF triples (introduced on page 40 in Section 3.2.1): For each tuple $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ we define:

$$c_{\mathsf{Match}}(t, u, P) := \begin{cases} \text{true} & \text{if there exists a triple pattern } tp \text{ in } P \text{ and } t \text{ matches } tp, \\ \text{false} & \text{else.} \end{cases}$$

Section 4.3 shall provide a more comprehensive discussion of different reachability criteria and the relationships among them (cf. page 69ff).

Using the concept of a reachability criterion, we define reachability of LD documents:

Definition 4.2 (Reachability). Let $S \subseteq \mathcal{U}$ be a finite set of URIs; let c be a reachability criterion; let P be a SPARQL expression; and let $W = (D, data, adoc)$ be a Web of Linked Data. Then, an LD document $d \in D$ is *(c, P)-reachable from S in W* if any of the following two conditions holds:

1. There exists a URI $u \in S$ such that $adoc(u) = d$; or

2. there exist (another) LD document $d' \in D$, an RDF triple $t \in data(d')$, and a URI $u \in \mathrm{uris}(t)$ such that
   a) d' is (c, P)-reachable from S in W,
   b) $c(t, u, P) = \text{true}$,
   c) $adoc(u) = d$. □

From Condition 1 in Definition 4.2 we see that any LD document that is authoritative for any of the URIs given in S (which serve as *seed*), is always reachable from S in the corresponding Web of Linked Data, independent of the reachability criterion and the SPARQL expression used. We call such a reachable LD document (that satisfies the first condition in Definition 4.2) a *seed document*.

Example 4.1. Consider a set of seed URIs $S_{\mathsf{ex}} = \{\mathsf{producer1}\}$ and a SPARQL expression P_{ex} that is given as follows:

$$\Big((?product, \mathsf{producedBy}, \mathsf{producer1}) \ \ \mathsf{AND} \ \ (?product, \mathsf{name}, ?productName)\Big)$$

The only LD document that is $(c_{\mathsf{None}}, P_{\mathsf{ex}})$-reachable from S_{ex} in our example Web of Linked Data W_{ex} (cf. Example 2.1, page 18) is d_{Pr1}. This document is also $(c_{\mathsf{Match}}, P_{\mathsf{ex}})$-reachable from S_{ex} in W_{ex} and it is $(c_{\mathsf{All}}, P_{\mathsf{ex}})$-reachable from S_{ex} in W_{ex}. In all three cases LD document d_{Pr1} is the (only) seed document.

Further LD documents that are $(c_{\mathsf{Match}}, P_{\mathsf{ex}})$-reachable from S_{ex} in W_{ex} are d_{p2} and d_{p3}. The set of all LD documents that are $(c_{\mathsf{All}}, P_{\mathsf{ex}})$-reachable from S_{ex} in W_{ex} also includes these documents and, additionally, d_{p1}. □

Based on reachability of LD documents we define reachable subwebs of a Web of Linked Data. Such a subweb is an induced subweb covering all reachable LD documents.

Definition 4.3 (Reachable Subweb). Let $S \subseteq \mathcal{U}$ be a finite set of URIs; let c be a reachability criterion; let P be a SPARQL expression; and let $W = (D, data, adoc)$ be a Web of Linked Data. The *(S, c, P)-reachable subweb of W* is an induced subweb $(D_{\mathfrak{R}}, data_{\mathfrak{R}}, adoc_{\mathfrak{R}})$ of W defined by:

$$D_{\mathfrak{R}} := \big\{ d \in D \,\big|\, d \text{ is } (c, P)\text{-reachable from } S \text{ in } W \big\}. \qquad \square$$

4.1.2. SPARQL$_{\mathsf{LD(R)}}$

We now use the concept of a reachable subweb to define reachability-based semantics for SPARQL-based Linked Data queries. We refer to such queries as *SPARQL$_{\mathsf{LD(R)}}$ queries*.

Definition 4.4 (SPARQL$_{\mathsf{LD(R)}}$ Query). Let $S \subseteq \mathcal{U}$ be a finite set of URIs; let c be a reachability criterion; and let P be a SPARQL expression. The *SPARQL$_{\mathsf{LD(R)}}$ query* that uses P, S, and c, denoted by $\mathcal{Q}_c^{P,S}$, is a Linked Data query that, for any Web of Linked Data W, is defined by $\mathcal{Q}_c^{P,S}(W) := [\![P]\!]_{\mathsf{AllData}(R)}$ where R denotes the (S, c, P)-reachable subweb of W. □

According to Definition 4.4, our notion of SPARQL$_{\mathsf{LD(R)}}$ consists of a family of (reachability-based) query semantics, each of which is characterized by a certain reachability criterion. Therefore, we refer to SPARQL$_{\mathsf{LD(R)}}$ queries for which we use a particular reachability criterion c as SPARQL$_{\mathsf{LD(R)}}$ queries *under c-semantics*.

Definition 4.4 also shows that query results depend on the given set of seed URIs $S \subseteq \mathcal{U}$. It is easy to see that any SPARQL$_{\mathsf{LD(R)}}$ query that uses an empty set of seed URIs is not satisfiable and, thus, trivially monotonic and finitely computable by an LD machine. We therefore consider only nonempty sets of seed URIs.

Example 4.2. Let $\mathcal{Q}^{P_{\mathsf{ex}},S_{\mathsf{ex}}}_{c_{\mathsf{None}}}$ be the SPARQL$_{\mathsf{LD(R)}}$ query under *c_{None}-semantics* that uses the SPARQL expression P_{ex} and the set of seed URIs S_{ex} as given in Example 4.1. The corresponding $(S_{\mathsf{ex}}, c_{\mathsf{None}}, P_{\mathsf{ex}})$-reachable subweb of our example Web of Linked Data W_{ex} consists of LD document d_{Pr1} only (cf. Example 4.1). Although $data_{\mathsf{ex}}(d_{\mathsf{Pr1}})$ includes two matching triples for the first triple pattern in SPARQL expression P_{ex}, none of the RDF triples in $data_{\mathsf{ex}}(d_{\mathsf{Pr1}})$ is a matching triple for the second triple pattern (cf. Example 2.1, page 18). Therefore, the result of query $\mathcal{Q}^{P_{\mathsf{ex}},S_{\mathsf{ex}}}_{c_{\mathsf{None}}}$ over W_{ex} is empty:

$$\mathcal{Q}^{P_{\mathsf{ex}},S_{\mathsf{ex}}}_{c_{\mathsf{None}}}(W_{\mathsf{ex}}) = \emptyset\,.$$

Let $\mathcal{Q}^{P_{\mathsf{ex}},S_{\mathsf{ex}}}_{c_{\mathsf{Match}}}$ and $\mathcal{Q}^{P_{\mathsf{ex}},S_{\mathsf{ex}}}_{c_{\mathsf{All}}}$ be the corresponding SPARQL$_{\mathsf{LD(R)}}$ queries under *c_{Match}-semantics* and under *c_{All}-semantics*, respectively. For c_{Match}-semantics the $(S_{\mathsf{ex}}, c_{\mathsf{Match}}, P_{\mathsf{ex}})$-reachable subweb of W_{ex} consists of LD documents d_{Pr1}, d_{p2}, and d_{p3} (cf. Example 2.1). The $(S_{\mathsf{ex}}, c_{\mathsf{All}}, P_{\mathsf{ex}})$-reachable subweb of W_{ex} additionally includes d_{p1}. Therefore:

$$\mathcal{Q}^{P_{\mathsf{ex}},S_{\mathsf{ex}}}_{c_{\mathsf{Match}}}(W_{\mathsf{ex}}) = \{\mu_1, \mu_2\} \qquad \text{and} \qquad \mathcal{Q}^{P_{\mathsf{ex}},S_{\mathsf{ex}}}_{c_{\mathsf{All}}}(W_{\mathsf{ex}}) = \{\mu_1, \mu_2, \mu_3\},$$

where $\mu_1 = \{?product \to \mathsf{product2}, ?productName \to \text{"Product 2"}\}$, $\mu_2 = \{?product \to \mathsf{product3}, ?productName \to \text{"Product 3"}\}$, and $\mu_3 = \{?product \to \mathsf{product1}, ?productName \to \text{"Product 1"}\}$, respectively. □

In the next section we discuss more extensively the results of SPARQL$_{\mathsf{LD(R)}}$ queries. In particular, we provide a general comparison of SPARQL$_{\mathsf{LD(R)}}$ and SPARQL$_{\mathsf{LD}}$ w.r.t. containment of query results. Based on this comparison we discuss query results under reachability-based semantics in the context of infinitely large Webs of Linked Data. Following sections then focus on the concept of reachability criteria, and analyze theoretical properties of SPARQL$_{\mathsf{LD(R)}}$ queries.

4.2. Result Containment and Infiniteness

Definition 4.4 defines precisely what the sound and complete result of any SPARQL$_{\mathsf{LD(R)}}$ query over any Web of Linked Data W is. However, in contrast to SPARQL$_{\mathsf{LD}}$ (as discussed in Chapter 3), there is no guarantee that such a (complete) SPARQL$_{\mathsf{LD(R)}}$ query result is complete w.r.t. all data in W since the corresponding (S, c, P)-reachable subweb of W may not cover W as a whole. We emphasize that such an incomplete coverage is even possible for the reachability criterion c_{All} because the link graph of W

may not be connected; therefore, the c_{All}-semantics differs from the full-Web semantics. The following proposition relates the result of any SPARQL$_{\mathsf{LD(R)}}$ query to the result of its SPARQL$_{\mathsf{LD}}$ counterpart.

Proposition 4.1. *Let $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query; let $\mathcal{Q}^P$ be the SPARQL$_{\mathsf{LD}}$ query that uses the same SPARQL expression P as used by $\mathcal{Q}_c^{P,S}$; let W be a Web of Linked Data. Then, the following two properties hold:*

1. *$\mathcal{Q}_c^{P,S}(W) = \mathcal{Q}^P(R)$ with R being the (S,c,P)-reachable subweb of W.*
2. *If $\mathcal{Q}^P$ is monotonic, then $\mathcal{Q}_c^{P,S}(W) \subseteq \mathcal{Q}^P(W)$.*

Proof. We first prove *Property 1*: By Definition 4.4, $\mathcal{Q}_c^{P,S}(W) = [\![P]\!]_{\mathsf{AllData}(R)}$ (cf. page 63) and, by Definition 3.1, $\mathcal{Q}^P(R) = [\![P]\!]_{\mathsf{AllData}(R)}$ (cf. page 42). Hence, we have $\mathcal{Q}_c^{P,S}(W) = \mathcal{Q}^P(R)$, as stated.

We now focus on *Property 2*: Suppose SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ is monotonic. By Definition 4.3 (cf. page 63), R is an induced subweb of W. Therefore, by Proposition 2.1 (cf. page 21), $\mathsf{AllData}(R) \subseteq \mathsf{AllData}(W)$ holds. Then, $\mathcal{Q}^P(R) \subseteq \mathcal{Q}^P(W)$ because $\mathcal{Q}^P$ is monotonic. Using the previously shown Property 1, we conclude $\mathcal{Q}_c^{P,S}(W) \subseteq \mathcal{Q}^P(W)$. ■

Since the result of any SPARQL$_{\mathsf{LD}}$ query over a finite Web of Linked Data is finite, we may use Proposition 4.1 (Property 1) to show the same for SPARQL$_{\mathsf{LD(R)}}$ queries:

Corollary 4.1. *The result of any SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ over a finite Web of Linked Data W is finite, and so is the (S,c,P)-reachable subweb of W.*

Proof. Let $W = (D, data, adoc)$, and let $R = (D_{\mathfrak{R}}, data_{\mathfrak{R}}, adoc_{\mathfrak{R}})$ be the (S,c,P)-reachable subweb of W. We first show finiteness for R: By Definition 4.3, R is an induced subweb of W (cf. page 63) and, thus, by Definition 2.4, we have $D_{\mathfrak{R}} \subseteq D$ (cf. page 21). Then, using the finiteness of D, $D_{\mathfrak{R}}$ is finite and hence R is finite.

Given the finiteness of R, the finiteness of $\mathcal{Q}_c^{P,S}(W)$ follows directly from (i) Proposition 4.1, Property 1, and (ii) Proposition 3.5 (which shows that the result of any SPARQL$_{\mathsf{LD}}$ query over a finite Web of Linked Data is finite; cf. page 57). ■

Corollary 4.1 focuses on a finite Web of Linked Data. Now, we study the implications of querying an infinite Web of Linked Data (using reachability-based query semantics). We first take a look at some example queries:

Example 4.3. For the example we assume the same infinite Web of Linked Data W_{inf} as used in Example 3.4 (cf. page 58). We recall that $W_{\mathsf{inf}} = (D_{\mathsf{inf}}, data_{\mathsf{inf}}, adoc_{\mathsf{inf}})$ contains an LD document $d_i \in D_{\mathsf{inf}}$ for every integer $i \in \mathbb{Z}$, that is, $adoc_{\mathsf{inf}}(\mathsf{no}_i) = d_i$ where URI $\mathsf{no}_i \in \mathcal{U}$ identifies integer i. The data of each of these documents consists of two RDF triples that refer to the predecessor and to the successor of the corresponding integer: $data_{\mathsf{inf}}(d_i) = \big\{(\mathsf{no}_i, \mathsf{pred}, \mathsf{no}_{i-1}), (\mathsf{no}_i, \mathsf{succ}, \mathsf{no}_{i+1})\big\}$ for all $i \in \mathbb{Z}$. Furthermore, as a basis for two SPARQL$_{\mathsf{LD}}$ queries, Example 3.4 uses two triple patterns: $tp_1 = (\mathsf{no}_0, \mathsf{succ}, ?v)$ and $tp_2 = (?x, \mathsf{succ}, ?y)$.

We now revisit this example in the context of reachability-based query semantics. We consider the aforementioned reachability criteria c_{All}, c_{Match}, and c_{None} (cf. page 62 in Section 4.1) and use URI no_0 as seed URI; i.e., $S = \{\mathsf{no}_0\}$.

First, we focus on triple pattern tp_1: If we assume $adoc_{\mathsf{inf}}(\mathsf{pred}) = adoc_{\mathsf{inf}}(\mathsf{succ}) = \bot$, then the $(S, c_{\mathsf{All}}, tp_1)$-reachable subweb of W_{inf} consists of the LD documents for all integers and, thus, is infinite. In contrast, the corresponding reachable subwebs for c_{Match} and c_{None} are finite: The $(S, c_{\mathsf{Match}}, tp_1)$-reachable subweb of W_{inf} consists of LD documents d_0 and d_1, whereas the $(S, c_{\mathsf{None}}, tp_1)$-reachable subweb of W_{inf} consists only of d_0. Irrespective of these differences, the query result is the same in all three cases: $\mathcal{Q}^{tp_1,S}_{c_{\mathsf{All}}}(W_{\mathsf{inf}}) = \mathcal{Q}^{tp_1,S}_{c_{\mathsf{Match}}}(W_{\mathsf{inf}}) = \mathcal{Q}^{tp_1,S}_{c_{\mathsf{None}}}(W_{\mathsf{inf}}) = \{\{?v \to \mathsf{no}_1\}\}$.

We now consider triple pattern tp_2: Under c_{None}-semantics the query result is the same as in the case of tp_1 because the $(S, c_{\mathsf{None}}, tp_2)$-reachable subweb of W_{inf} consists only of LD document d_0 (as before). For c_{All} and c_{Match} the reachable subwebs are infinite but different: The $(S, c_{\mathsf{All}}, tp_1)$-reachable subweb of W_{inf} consists, again, of the LD documents for all integers, whereas the $(S, c_{\mathsf{Match}}, tp_1)$-reachable subweb of W_{inf} consists of the LD documents $d_0, d_1, d_2, \ldots$. The query results for both criteria are also infinite and different from each other: $\mathcal{Q}^{tp_2,S}_{c_{\mathsf{All}}}(W_{\mathsf{inf}}) = \{\ldots, \{?x \to \mathsf{no}_{\text{-}1}, ?y \to \mathsf{no}_0\}, \{?x \to \mathsf{no}_0, ?y \to \mathsf{no}_1\}, \ldots\}$ and $\mathcal{Q}^{tp_2,S}_{c_{\mathsf{Match}}}(W_{\mathsf{inf}}) = \{\{?x \to \mathsf{no}_0, ?y \to \mathsf{no}_1\}, \{?x \to \mathsf{no}_1, ?y \to \mathsf{no}_2\}, \ldots\} \subset \mathcal{Q}^{tp_2,S}_{c_{\mathsf{All}}}(W_{\mathsf{inf}})$. □

The example illustrates that, for the case of an *infinite* Web of Linked Data, the results of SPARQL$_{\mathsf{LD(R)}}$ queries may be either finite or infinite. In Example 3.4 we found the same heterogeneity for SPARQL$_{\mathsf{LD}}$ queries (cf. page 58). However, for SPARQL$_{\mathsf{LD(R)}}$ we may identify dependencies between query results and the corresponding reachable subwebs of the queried Web:

Proposition 4.2. *Let $S \subseteq \mathcal{U}$ be a finite set of URIs; let c be a reachability criterion; let P be a SPARQL expression; let W be a (potentially infinite) Web of Linked Data, and let R denote the (S, c, P)-reachable subweb of W. Then, the following properties hold:*

1. *If R is finite, then $\mathcal{Q}^{P,S}_{c}(W)$ is finite.*
2. *If $\mathcal{Q}^{P,S}_{c}(W)$ is infinite, then R is infinite.*
3. *If c is c_{None}, then R is finite, and so is $\mathcal{Q}^{P,S}_{c_{\mathsf{None}}}(W)$.*

Proof. *Property 1:* By Proposition 4.1 (Property 1), it holds that $\mathcal{Q}^{P,S}_{c}(W) = \mathcal{Q}^{P}(R)$ where $\mathcal{Q}^{P}$ is the SPARQL$_{\mathsf{LD}}$ query that uses the same SPARQL expression as used by $\mathcal{Q}^{P,S}_{c}$. Since the result of any SPARQL$_{\mathsf{LD}}$ query over a finite Web of Linked Data is finite (as shown in Proposition 3.5, page 57), Property 1 follows immediately.

Property 2: Suppose $\mathcal{Q}^{P,S}_{c}(W)$ is infinite. We use proof by contradiction, that is, we assume R is finite. Then, by Property 1, $\mathcal{Q}^{P,S}_{c}(W)$ is also finite, a contradiction. Hence, R must be infinite.

Property 3: Let $W = (D, data, adoc)$ and $R = (D_{\mathfrak{R}}, data_{\mathfrak{R}}, adoc_{\mathfrak{R}})$. Suppose c is c_{None}. Since c_{None} always returns false, it is easily verified that there does not exist an LD document $d \in D$ that satisfies Case 2 in Definition 4.2 (cf. page 62). Hence, $D_{\mathfrak{R}}$ contains the seed documents only, that is, $D_{\mathfrak{R}} = \{d \in D \,|\, u \in S \text{ and } adoc(u) = d\}$ (cf. Case 1 in Definition 4.2). Since S is finite, $D_{\mathfrak{R}}$ is finite, and so is R. Then, the finiteness of $\mathcal{Q}^{P,S}_{c_{\mathsf{None}}}(W)$ follows by Property 1. ■

Proposition 4.2 provides valuable insight into the dependencies between the (in)finiteness of reachable subwebs of an infinite Web and the (in)finiteness of query results. In practice, however, we are primarily interested in answering the following questions: Does the execution of a given SPARQL$_{\mathsf{LD(R)}}$ query reach an infinite number of LD documents? Do we have to expect an infinite query result? We formalize these questions as the following LD decision problems and discuss them in the remainder of this section.

LD Problem:	FINITENESSREACHABLEPART
Web Input:	a (potentially infinite) Web of Linked Data W
Ordin. Input:	a SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$
Question:	Is the (S, c, P)-reachable subweb of W finite?

LD Problem:	FINITENESS(SPARQL$_{\mathsf{LD(R)}}$)
Web Input:	a (potentially infinite) Web of Linked Data W
Ordin. Input:	a SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$
Question:	Is query result $\mathcal{Q}_c^{P,S}(W)$ finite?

As in the case of FINITENESS(SPARQL$_{\mathsf{LD}}$), discussed on page 58, an LD machine can trivially decide FINITENESS(SPARQL$_{\mathsf{LD(R)}}$) for unsatisfiable[1] SPARQL$_{\mathsf{LD(R)}}$ queries. In contrast, the satisfiability property of queries is irrelevant for FINITENESSREACHABLEPART: The reachable subweb of a queried Web of Linked Data may be infinite regardless of whether the corresponding SPARQL$_{\mathsf{LD(R)}}$ query is unsatisfiable or satisfiable. Nonetheless, for a particular class of SPARQL$_{\mathsf{LD(R)}}$ queries we can rule out the existence of infinitely large reachable subwebs. This class comprises all queries that use a reachability criterion that ensures the finiteness of reachable subwebs in any possible case by definition. We define such property of reachability criteria as follows:

Definition 4.5 (Ensuring Finiteness). A reachability criterion c *ensures finiteness* if, for any Web of Linked Data W, any (finite) set $S \subseteq \mathcal{U}$ of URIs, and any SPARQL expression P, the (S, c, P)-reachable subweb of W is finite. □

From the reachability criteria discussed so far, only c_{None} ensures finiteness (see Proposition 4.2); this property does not hold for c_{All} and c_{Match} (as shown by Example 4.3). We refer to the next section, in particular, Subsections 4.3.3 and 4.3.4 (cf. page 73ff), for a more comprehensive discussion of reachability criteria that ensure finiteness. However, due to its relevance for FINITENESS(SPARQL$_{\mathsf{LD(R)}}$), we emphasize that using reachability criteria that ensure finiteness also guarantee finite query results:

Corollary 4.2. *Let c be a reachability criterion that ensures finiteness. For any SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ under c-semantics and any Web of Linked Data W, query result $\mathcal{Q}_c^{P,S}(W)$ is finite.*

Proof. The corollary follows readily from Definition 4.5 and Proposition 4.2. ■

Given Definition 4.5 and Corollary 4.2 we see that, for a SPARQL$_{\mathsf{LD(R)}}$ query whose reachability criterion ensures finiteness, an LD machine can immediately answer the

[1]We shall discuss satisfiability of SPARQL$_{\mathsf{LD(R)}}$ queries in Section 4.4.1 (cf. page 77ff).

questions posed by FINITENESSREACHABLEPART and by FINITENESS($\text{SPARQL}_{\text{LD(R)}}$). Thus, for both of these problems we have decision criteria that cover certain classes of queries. In general, however, both problems are undecidable for LD machines.

Theorem 4.1. *FINITENESSREACHABLEPART and FINITENESS($\text{SPARQL}_{\text{LD(R)}}$) are not LD machine decidable.*

Proof. We prove Theorem 4.1 by reducing the halting problem to FINITENESSREACHABLEPART and to FINITENESS($\text{SPARQL}_{\text{LD(R)}}$). While this proof resembles the proofs of Proposition 3.4 (cf. page 50) and Theorem 3.3 (cf. page 58), we need to use a Web of Linked Data W_{TMs3} that differs from Webs W_{TMs} and W_{TMs2} (used in the aforementioned proofs). Although W_{TMs3} also describes all possible computations of all Turing machines, this description differs from the descriptions in W_{TMs} and W_{TMs2}.

We use the same symbols as in the aforementioned proofs: $\mathcal{W}$ denotes the countably infinite set of all words that describe Turing machines. For all $w \in \mathcal{W}$, $M(w)$ denotes the machine described by word w; $c^{w,x}$ denotes the computation of machine $M(w)$ on input x; URI $u_i^{w,x} \in \mathcal{U}$ identifies the i-th step in computation $c^{w,x}$. The countably infinite set of all these identifiers is denoted by $\mathcal{U}_{\mathsf{TMsteps}}$.

We now define W_{TMs3} as a Web of Linked Data $(D_{\mathsf{TMs3}}, data_{\mathsf{TMs3}}, adoc_{\mathsf{TMs3}})$ similar to the Web W_{TMs} used for proving Proposition 3.4: D_{TMs3} and $adoc_{\mathsf{TMs3}}$ are the same is in W_{TMs}. That is, D_{TMs3} consists of $|\mathcal{U}_{\mathsf{TMsteps}}|$ different LD documents, each of which corresponds to one of the URIs in $\mathcal{U}_{\mathsf{TMsteps}}$. Mapping $adoc_{\mathsf{TMs3}}$ maps each URI $u_i^{w,x} \in \mathcal{U}_{\mathsf{TMsteps}}$ to the corresponding LD document $d_i^{w,x} \in D_{\mathsf{TMs3}}$. Mapping $data_{\mathsf{TMs3}}$ for W_{TMs3} is different from the corresponding mapping for W_{TMs}: The set $data_{\mathsf{TMs3}}(d_i^{w,x})$ of RDF triples for an LD document $d_i^{w,x}$ is empty if computation $c^{w,x}$ halts with the i-th computation step. Otherwise, $data_{\mathsf{TMs3}}(d_i^{w,x})$ contains a single RDF triple $(u_i^{w,x}, \mathsf{next}, u_{i+1}^{w,x})$ which associates the computation step identified by URI $u_i^{w,x}$ with the next step in $c^{w,x}$ ($\mathsf{next} \in \mathcal{U}$ denotes a URI for this relationship). Formally:

$$data_{\mathsf{TMs3}}(d_i^{w,x}) := \begin{cases} \emptyset & \text{if computation } c^{w,x} \text{ halts with the } i\text{-th step,} \\ \{(u_i^{w,x}, \mathsf{next}, u_{i+1}^{w,x})\} & \text{else.} \end{cases}$$

Mappings $adoc_{\mathsf{TMs3}}$ and $data_{\mathsf{TMs3}}$ are Turing computable (by simulation).

For the reduction we use mapping f which is defined as follows: Let (w,x) be an input to the halting problem and let $?a, ?b \in \mathcal{V}$ be two distinct query variables, then $f(w,x) = \big(W_{\mathsf{TMs3}}, \mathcal{Q}_{c_{\mathsf{Match}}}^{P_{w,x}, S_{w,x}}\big)$ where $S_{w,x} = \{u_1^{w,x}\}$ and $P_{w,x} = (?a, \mathsf{next}, ?b)$. Given that c_{Match} and W_{TMs3} are independent of (w,x), it can be easily seen that f is computable by Turing machines (including LD machines).

Before we present the reduction we highlight a property of W_{TMs3} that is important for our proof. Any RDF triple of the form $(u_i^{w,x}, \mathsf{next}, u_{i+1}^{w,x})$ establishes a data link from LD document $d_i^{w,x}$ to LD document $d_{i+1}^{w,x}$. Based on such links we may reach all LD documents about all steps in a particular computation of any Turing machine (given the corresponding w and x). Hence, for each possible computation $c^{w,x}$ of any Turing machine $M(w)$ we have a (potentially infinite) simple path $(d_1^{w,x}, \dots, d_i^{w,x}, \dots)$ in the link graph of W_{TMs3}. Each of these paths is finite if and only if the corresponding computation halts. Moreover, each of these paths forms a separate subgraph of the link graph of W_{TMs3} because we use a separate set of step URIs for each computation and the RDF triples in the corresponding LD documents mention steps from the same computation only. As a consequence, the following two properties hold for any halting problem input (w, x):

1. Turing machine $M(w)$ halts on input x if and only if the $(S_{w,x}, c_{\mathsf{Match}}, P_{w,x})$-reachable subweb of W_{TMs3} is finite.
2. Turing machine $M(w)$ halts on input x if and only if query result $\mathcal{Q}^{P_{w,x},S_{w,x}}_{c_{\mathsf{Match}}}(W_{\mathsf{TMs3}})$ is finite.

To show that FINITENESSREACHABLEPART is not LD machine decidable, suppose the problem is LD machine decidable. Then, an LD machine can answer the halting problem for any input (w, x) by using the first of the two aforementioned properties (i.e., Turing machine $M(w)$ halts on x if and only if the $(S_{w,x}, c_{\mathsf{Match}}, P_{w,x})$-reachable subweb of W_{TMs3} is finite). Since the halting problem is undecidable for Turing machines (and, thus, for LD machines), we have a contradiction. Therefore, FINITENESSREACHABLEPART cannot be LD machine decidable.

The proof that FINITENESS(SPARQL$_{\mathrm{LD(R)}}$) is undecidable for LD machines follows similar steps. Hence, we only outline the idea of the proof: Instead of reducing the halting problem to FINITENESSREACHABLEPART based on mapping f we now reduce the halting problem to FINITENESS(SPARQL$_{\mathrm{LD(R)}}$) using the same mapping. If FINITENESS(SPARQL$_{\mathrm{LD(R)}}$) is decidable, the halting problem for any (w, x) is decidable since the corresponding Turing machine $M(w)$ halts on x if and only if $\mathcal{Q}^{P_{w,x},S_{w,x}}_{c_{\mathsf{Match}}}(W_{\mathsf{TMs3}})$ is finite. ■

In summary, this section showed that the reachable subweb of some Webs of Linked Data is infinite for some SPARQL$_{\mathsf{LD(R)}}$ queries. In Section 4.4.3 we shall see that the existence of these cases impacts the computational feasibility of SPARQL$_{\mathsf{LD(R)}}$ queries (cf. page 90ff). However, we first elaborate further on reachability criteria before discussing theoretical properties of SPARQL$_{\mathsf{LD(R)}}$.

4.3. Reachability Criteria

The concept of a reachability criterion is a key concept for our notion of reachability-based query semantics. As a consequence, this section provides a more detailed discussion of these criteria. In particular, we elaborate on how to compare and how to combine such criteria. Furthermore, we discuss the finiteness property defined in the previous section, and we introduce a particular class of reachability criteria for which this property holds.

4.3.1. Comparing Reachability Criteria

We compare reachability criteria based on the data links they accept. Informally, a criterion c_1 is *less restrictive than* criterion c_2 if c_1 accepts a proper superset of the links that c_2 accepts. Formally, we define this notion of restrictiveness as follows:

Definition 4.6 (Less Restrictive). Let $\mathcal{T}$, $\mathcal{U}$, and $\mathcal{P}$ denote the infinite sets of all possible RDF triples, URIs, and SPARQL expressions, respectively. A reachability criterion c_1 is *less restrictive than* a reachability criterion c_2, denoted by $c_1 \lhd c_2$, if (i) for each tuple $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ for which $c_2(t, u, P) = \text{true}$, also $c_1(t, u, P) = \text{true}$, and (ii) there exists a tuple $(t', u', P') \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ such that $c_1(t', u', P') = \text{true}$ but $c_2(t', u', P') = \text{false}$. □

In addition to less restrictiveness we may say two reachability criteria c_1 and c_2 are *equally restrictive* if $c_1(t, u, P) = c_2(t, u, P)$ for all tuples $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$. However, in this dissertation we understand two equally restrictive reachability criteria as a single criterion for which multiple definitions exist. Then, it can be easily seen that reachability criterion c_{All} is the least restrictive criterion, that is, for any other reachability criterion c it holds that $c_{\mathsf{All}} \lhd c$. Similarly, c_{None} is the most restrictive reachability criterion (i.e., $c \lhd c_{\mathsf{None}}$ for any other reachability criterion c).

We note that the relation $\lhd$ is a strict order over the set of all possible reachability criteria. However, $\lhd$ is only partial. That is, not any pair of distinct reachability criteria can be compared (using $\lhd$) as the following example illustrates.

Example 4.4. Consider two distinct RDF triples $t_1, t_2 \in \mathcal{U} \times \mathcal{U} \times \mathcal{U}$ that consist of URIs only. Using t_1 and t_2 we may define two simple (and fairly restrictive) reachability criteria c_{t_1} and c_{t_2} as follows. For each tuple $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$, let:

$$c_{t_1}(t,u,P) := \begin{cases} \text{true} & \text{if } t \text{ is } t_1, \\ \text{false} & \text{else,} \end{cases} \qquad \text{and} \qquad c_{t_2}(t,u,P) := \begin{cases} \text{true} & \text{if } t \text{ is } t_2, \\ \text{false} & \text{else.} \end{cases}$$

Thus, in addition to its seed documents, an (S, c_{t_1}, P)-reachable subweb of an arbitrary Web of Linked Data W (for any set of seed URIs S and any SPARQL expression P) may contain at most three additional LD documents. This requires however that (i) RDF triple t_1 is available in at least one of the seed documents and (ii) there actually exist LD documents in W that are authoritative for the three URIs in t_1. Similarly, for c_{t_2}.

Apparently, neither c_{t_1} is less restrictive than c_{t_2} nor vice versa (nor are they equally restrictive). However, we may introduce an additional criterion $c_{\{t_1,t_2\}}$ that maps any tuple $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ to true if and only if t is t_1 or t is t_2. Then, $c_{\{t_1,t_2\}} \lhd c_{t_1}$ and $c_{\{t_1,t_2\}} \lhd c_{t_2}$. Notice, c_{t_1}, c_{t_2}, and $c_{\{t_1,t_2\}}$ are *constant reachability criteria*, a particular class of reachability criteria that we shall discuss in Section 4.3.4 (cf. page 74ff). □

Given the comparability of (some) reachability criteria we now can, at least in theory, compare corresponding reachable subwebs of a queried Web of Linked Data, as well as corresponding query results:

Proposition 4.3. *Let W be a Web of Linked Data; let $S \subseteq \mathcal{U}$ be a finite set of URIs; let P be a SPARQL expression; and let c and c' be reachability criteria such that $c \lhd c'$. Then, the (S, c', P)-reachable subweb of W is an induced subweb of the (S, c, P)-reachable subweb of W. Furthermore, if P is monotonic, then $\mathcal{Q}^{P,S}_{c'}(W) \subseteq \mathcal{Q}^{P,S}_{c}(W)$.*

Proof. Let $R = (D_{\mathfrak{R}}, data_{\mathfrak{R}}, adoc_{\mathfrak{R}})$ and $R' = (D'_{\mathfrak{R}}, data'_{\mathfrak{R}}, adoc'_{\mathfrak{R}})$ be the (S, c, P)-reachable subweb and the (S, c', P)-reachable subweb of W, respectively.

To show that R' is an induced subweb of R it suffices to show $D'_{\mathfrak{R}} \subseteq D_{\mathfrak{R}}$, because, by Definition 4.3 (cf. page 63), both reachable subwebs are induced subwebs of W. $D'_{\mathfrak{R}} \subseteq D_{\mathfrak{R}}$ holds because any LD document $d \in D'_{\mathfrak{R}}$ is not only (c', P)-reachable from S in W but, due to $c \lhd c'$, document d is also (c, P)-reachable from S in W and, thus, $d \in D_{\mathfrak{R}}$.

Since R' is an induced subweb of R we also have $\mathsf{AllData}(R') \subseteq \mathsf{AllData}(R)$ (cf. Proposition 2.1, page 21). If P is monotonic, it holds that $[\![P]\!]_{\mathsf{AllData}(R')} \subseteq [\![P]\!]_{\mathsf{AllData}(R)}$ and, thus, by Definition 4.4 (cf. page 63), $\mathcal{Q}^{P,S}_{c'}(W) \subseteq \mathcal{Q}^{P,S}_{c}(W)$. ■

The following properties are a trivial consequence of Proposition 4.3.

Corollary 4.3. *Let W be a Web of Linked Data; let $S \subseteq \mathcal{U}$ be a finite set of URIs; let P be a SPARQL expression; and let c and c' be reachability criteria such that $c \lhd c'$. Furthermore, let R and R' denote the (S, c, P)-reachable subweb and the (S, c', P)-reachable subweb of W, respectively. The following two properties hold:*

1. *If R is finite, then R' is finite.*
2. *If R' is infinite, then R' is infinite.*

Proof. The result follows immediately from Proposition 4.3 and the fact that $D'_{\mathfrak{R}} \subseteq D_{\mathfrak{R}}$ (where $D'_{\mathfrak{R}}$ and $D_{\mathfrak{R}}$ denote the set of LD documents in R' and in R, respectively). ■

4.3.2. Combining Reachability Criteria

Consider reachability criteria c_{t_1}, c_{t_2}, and $c_{\{t_1,t_2\}}$ that we introduce in the previous example (cf. Example 4.4). We note that $c_{\{t_1,t_2\}}$ is a (disjunctive) combination of c_{t_1} and c_{t_2}. In this section we provide a general formalism for such combinations by introducing an algebraic structure over the set of all possible reachability criteria. As a basis we define the following algebraic operations:

Definition 4.7 (Combining Reachability Criteria). Let $\mathcal{T}$, $\mathcal{U}$, $\mathcal{P}$, and $\mathcal{C}$ denote the infinite sets of all possible RDF triples, all URIs, all possible SPARQL expressions, and all possible reachability criteria, respectively. The *disjunctive combination* of reachability criteria, denoted by $\sqcup$, and the *conjunctive combination* of reachability criteria, denoted by $\sqcap$, are binary operations over $\mathcal{C}$ such that the following two properties hold for any pair of reachability criteria $(c_1, c_2) \in \mathcal{C} \times \mathcal{C}$ and any tuple $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$:

1. Let c be the reachability criterion $c_1 \sqcup c_2$, then $c(t, u, P) = \text{true}$ if and only if $c_1(t, u, P) = \text{true}$ or $c_2(t, u, P) = \text{true}$.
2. Let c be the reachability criterion $c_1 \sqcap c_2$, then $c(t, u, P) = \text{true}$ if and only if $c_1(t, u, P) = \text{true}$ and $c_2(t, u, P) = \text{true}$. □

Example 4.5. For the reachability criteria c_{t_1}, c_{t_2}, and $c_{\{t_1,t_2\}}$ that we introduce in Example 4.4 it holds that $c_{\{t_1,t_2\}}$ is the same as $c_{t_1} \sqcup c_{t_2}$. □

Based on their definition it is easy to verify that both operations, $\sqcup$ and $\sqcap$, have the following, desirable properties (where $\mathcal{C}$ denotes the set all possible reachability criteria):

- Operations $\sqcup$ and $\sqcap$ are *commutative*.
  That is, for each pair $(c_1, c_2) \in \mathcal{C} \times \mathcal{C}$ it holds that (i) $c_1 \sqcup c_2$ is the same as $c_2 \sqcup c_1$ and (ii) $c_1 \sqcap c_2$ is the same as $c_2 \sqcap c_1$.

- Set $\mathcal{C}$ is *closed* under $\sqcup$ and under $\sqcap$.
  That is, for each pair $(c_1, c_2) \in \mathcal{C} \times \mathcal{C}$ it holds that (i) $c_1 \sqcup c_2 \in \mathcal{C}$ and (ii) $c_1 \sqcap c_2 \in \mathcal{C}$.

- Operations $\sqcup$ and $\sqcap$ are *associative*.
  That is, for each triple $(c_1, c_2, c_3) \in \mathcal{C} \times \mathcal{C} \times \mathcal{C}$ it holds that (i) $(c_1 \sqcup c_2) \sqcup c_3$ is the same as $c_1 \sqcup (c_2 \sqcup c_3)$ and (ii) $(c_1 \sqcap c_2) \sqcap c_3$ is the same as $c_1 \sqcap (c_2 \sqcap c_3)$.

- Reachability criteria c_{All} and c_{None} are *identity* element for $\sqcup$ and $\sqcap$, respectively.
  That is, $c_{\mathsf{All}} \sqcup c$ is the same as c_{All} for all $c \in \mathcal{C}$; similarly, $c_{\mathsf{None}} \sqcap c$ is the same as c_{None} for all $c \in \mathcal{C}$.

- Any reachability criterion is *invertible* w.r.t. $\sqcup$ and to $\sqcap$.
  That is, if $\bar{c}$ denotes the inverse of reachability criterion c, then $c \sqcup \bar{c}$ is the same as identity element c_{All} for all $c \in \mathcal{C}$; similarly, $c \sqcap \bar{c}$ is the same as identity element c_{None} for all $c \in \mathcal{C}$. Formally, we define the *inverse* of a reachability criterion c as a reachability criterion $\bar{c}$ such that for each tuple $(t, u, P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ it holds that $\bar{c}(t, u, P) = \text{true}$ if and only if $c(t, u, P) = \text{false}$.

- Operation $\sqcup$ is *distributive* over $\sqcap$.
  That is, for each $(c_1, c_2, c_3) \in \mathcal{C} \times \mathcal{C} \times \mathcal{C}$ it holds that (i) $c_1 \sqcup (c_2 \sqcap c_3)$ is the same as $(c_1 \sqcup c_2) \sqcap (c_1 \sqcup c_3)$ and (ii) $(c_1 \sqcap c_2) \sqcup c_3$ is the same as $(c_1 \sqcup c_3) \sqcap (c_2 \sqcup c_3)$.

- Operation $\sqcap$ is *distributive* over $\sqcup$.
  That is, for each $(c_1, c_2, c_3) \in \mathcal{C} \times \mathcal{C} \times \mathcal{C}$ it holds that (i) $c_1 \sqcap (c_2 \sqcup c_3)$ is the same as $(c_1 \sqcap c_2) \sqcup (c_1 \sqcap c_3)$ and (ii) $(c_1 \sqcup c_2) \sqcap c_3$ is the same as $(c_1 \sqcap c_3) \sqcup (c_2 \sqcap c_3)$.

- Operations $\sqcup$ and $\sqcap$ are connected by the *absorption* law.
  That is, for each pair $(c_1, c_2) \in \mathcal{C} \times \mathcal{C}$ it holds that (i) $c_1 \sqcup (c_1 \sqcap c_2)$ is the same as c_1 and (ii) $c_1 \sqcap (c_1 \sqcup c_2)$ is the same as c_1.

As a consequence of these properties, in terms of ring theory [124], set $\mathcal{C}$ and operations $\sqcup$ and $\sqcap$ form two *commutative rings* $(\mathcal{C}, \sqcup, \sqcap)$ and $(\mathcal{C}, \sqcap, \sqcup)$. Furthermore, if we let $\trianglelefteq$ be the non-strict partial order (over $\mathcal{C}$) that corresponds to our strict partial order $\triangleleft$, then, in terms of order theory [38], $(\mathcal{C}, \trianglelefteq)$ is a *bounded lattice* (where $\sqcup$ and $\sqcap$ are the join and the meet operation of the lattice, respectively, and c_{All} and c_{None} are the top and the bottom element, respectively).

Finally, the following dependencies follow trivially from the corresponding definitions.

Proposition 4.4. *Let $\mathcal{C}$ denote the infinite set all possible reachability criteria. The following properties hold for each pair of reachability criteria $(c_1, c_2) \in \mathcal{C} \times \mathcal{C}$:*

1. $(c_1 \sqcup c_2) \trianglelefteq c_1$ *and* $(c_1 \sqcup c_2) \trianglelefteq c_2$.
2. $c_1 \trianglelefteq (c_1 \sqcap c_2)$ *and* $c_2 \trianglelefteq (c_1 \sqcap c_2)$.

3. *If $c_1 \lhd c_2$, then $c_1 \sqcup c_2$ is the same as c_1 and $c_1 \sqcap c_2$ is the same as c_2.*

4. *If c_1 is* not *the same as c_2, then $(c_1 \sqcup c_2) \lhd (c_1 \sqcap c_2)$.*

Proof. First, we define $\trianglelefteq$ formally: For any pair $(c_1, c_2) \in \mathcal{C} \times \mathcal{C}$ it holds that $c_1 \trianglelefteq c_2$ if either (i) $c_1 \lhd c_2$ or (ii) c_1 is the same as c_2. Then, Proposition 4.4 follows readily from our definition of $\lhd$ and our definition of $\sqcup$ and $\sqcap$. ■

4.3.3. Reachability Criteria that Ensure Finiteness

We now focus on the finiteness property for reachability criteria that we introduce in Section 4.2. Recall, a reachability criterion c^* ensures finiteness if any possible (S, c^*, P)-reachable subweb of any Web of Linked Data W is finite (cf. Definition 4.5, page 67). In what follows we first revisit this property given the possibility to compare and to combine reachability criteria. Afterwards, in Section 4.3.4, we introduce constant reachability criteria as a particular class of reachability criteria that ensure finiteness.

Using Corollary 4.3 we may easily show that a reachability criterion ensures finiteness if there exists another, less restrictive criterion that also ensures finiteness. Formally:

Corollary 4.4. *Let $\mathcal{C}_{ef}$ denote the set of all reachability criteria that ensure finiteness. For any pair of reachability criteria c and c' the following property holds: If $c \in \mathcal{C}_{ef}$ and $c \trianglelefteq c'$, then $c' \in \mathcal{C}_{ef}$.*

Proof. The corollary follows immediately from our definition of the finiteness property (that is, Definition 4.5) and Corollary 4.3 (cf. page 4.3). ■

The following result classifies (disjunctive and conjunctive) combinations of reachability criteria w.r.t. our finiteness property.

Proposition 4.5. *Let $\mathcal{C}_{ef}$ denote the set of all reachability criteria that ensure finiteness. For any two reachability criteria c_1 and c_2 with $c_1 \in \mathcal{C}_{ef}$, the following properties hold:*

1. $(c_1 \sqcap c_2) \in \mathcal{C}_{ef}$

2. $(c_1 \sqcup c_2) \in \mathcal{C}_{ef}$ *if and only if* $c_2 \in \mathcal{C}_{ef}$

To prove Proposition 4.5 (in particular, Property 2), we need the following characteristic that distinguishes reachability criteria that ensure finiteness from those that do not.

Lemma 4.1. *Let $\mathcal{T}$, $\mathcal{U}$, $\mathcal{P}$, and $\mathcal{C}$ denote the infinite sets of all possible RDF triples, all URIs, all possible SPARQL expressions, and all possible reachability criteria, respectively. Furthermore, we define a function $X : \mathcal{C} \times \mathcal{P} \rightarrow \mathcal{U}$ that maps any pair of a reachability criterion $c \in \mathcal{C}$ and a SPARQL expression $P \in \mathcal{P}$ to a set of URIs:*

$$X(c, P) := \{ u \in \mathcal{U} \mid \exists t \in \mathcal{T} : u \in \mathrm{uris}(t) \text{ and } c(t, u, P) = \mathrm{true} \} .$$

Then, for each reachability criterion $c \in \mathcal{C}$ it holds that c ensures finiteness if and only if $X(c, P)$ is finite for all SPARQL expressions $P \in \mathcal{P}$.

For the proof of Lemma 4.1 we refer to the appendix (cf. Section E.3, page 221f).

Before we prove Proposition 4.5 (based on Lemma 4.1), we emphasize that the sets $X(c,P)$ in Lemma 4.1 present upper bounds for all URIs based on which LD documents may be reached by applying the recursive (second) step in our definition of reachable LD documents (cf. Definition 4.2, page 62). Of course, it is not necessarily the case that all these URIs are discovered (and used) during such a recursive application of Definition 4.2 in a particular Web of Linked Data, starting from a particular set of seed documents.

We now use Lemma 4.1 to prove Proposition 4.5.

Proof of Proposition 4.5. Property 1 in Proposition 4.5 (i.e., $(c_1 \sqcap c_2) \in \mathcal{C}_{\mathsf{ef}}$) follows readily from Proposition 4.4 (cf. page 72), Corollary 4.4 (cf. page 73), and $c_1 \in \mathcal{C}_{\mathsf{ef}}$.

To show the second property in Proposition 4.5 (i.e., $(c_1 \sqcup c_2) \in \mathcal{C}_{\mathsf{ef}} \Leftrightarrow c_2 \in \mathcal{C}_{\mathsf{ef}}$), we first note that for any two reachability criteria c and c', $X(c \sqcup c', P) = X(c,P) \cup X(c',P)$ holds for all SPARQL expressions $P \in \mathcal{P}$ (where X is the function defined in Lemma 4.1).

We now distinguish whether c_2 ensures finiteness and use Lemma 4.1 in both cases:

- If $c_2 \in \mathcal{C}_{\mathsf{ef}}$, then both $X(c_2,P)$ and $X(c_1,P)$ are finite for all $P \in \mathcal{P}$ (regarding c_1 we recall $c_1 \in \mathcal{C}_{\mathsf{ef}}$). Then, $X(c_1 \sqcup c_2, P) = X(c_1,P) \cup X(c_2,P)$ is also finite for all $P \in \mathcal{P}$ and, thus, $(c_1 \sqcup c_2) \in \mathcal{C}_{\mathsf{ef}}$.

- If $c_2 \notin \mathcal{C}_{\mathsf{ef}}$, there exists a SPARQL expression $P \in \mathcal{P}$ such that $X(c_2,P)$ is infinite. Consequently, for this SPARQL expression P, $X(c_1 \sqcup c_2, P) = X(c_1,P) \cup X(c_2,P)$ is also infinite. Therefore, $(c_1 \sqcup c_2) \notin \mathcal{C}_{\mathsf{ef}}$. ■

While we leave the decidability of our finiteness property an open question for future research, the following section introduces a class of reachability criteria for which the property holds.

4.3.4. Constant Reachability Criteria

This section discusses a particular class of reachability criteria which we call *constant reachability criteria*. These criteria always only accept a given, constant set of data links. As a consequence, each of these criteria ensures finiteness. In the following we introduce constant reachability criteria and prove that they ensure finiteness.

The (fixed) set of data links that a constant reachability criterion accepts may be specified differently. Accordingly, we distinguish two basic types of constant reachability criteria. Formally, we define them as follows:

Definition 4.8 (URI-Constant Criterion and Triple-Constant Criterion). Let $\mathcal{T}$, $\mathcal{U}$, and $\mathcal{P}$ denote the infinite sets of all possible RDF triples, all URIs, and all possible SPARQL expressions, respectively. For any finite set of URIs $U \subseteq \mathcal{U}$ and any finite set of RDF triples $T \subseteq \mathcal{T}$, the *URI-constant criterion* for U, denoted by c^U, and the *triple-constant criterion* for T, denoted by c^T, are reachability criteria that for each tuple $(t,u,P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ are defined as follows:

$$c^U\Big(t,u,P\Big) := \begin{cases}\text{true} & \text{if } u \in U,\\ \text{false} & \text{else,}\end{cases} \qquad \text{and} \qquad c^T\Big(t,u,P\Big) := \begin{cases}\text{true} & \text{if } t \in T,\\ \text{false} & \text{else.}\end{cases} \qquad \square$$

As can be seen from the definition, URI-constant criteria use a (finite) set of URIs to specify the data links that they accept. Similarly, triple-constant criteria use a (finite) set of RDF triples. An example for triple-constant criteria are the the reachability criteria c_{t_1}, c_{t_2}, and $c_{\{t_1,t_2\}}$ in Example 4.4 (cf. page 70). Another example for such criteria is c_{None}, which presents the following special case: c_{None} is the URI-constant criterion that uses an empty set of URIs and c_{None} is the triple-constant criterion that uses an empty set of RDF triples. The following properties are trivial to verify:

Property 4.1. *If c^U and $c^{U'}$ are URI-constant criteria such that $U' \subset U$, then $c^U \lhd c^{U'}$. Similarly, if c^T and $c^{T'}$ are triple-constant criteria such that $T' \subset T$, then $c^T \lhd c^{T'}$.*

As any other reachability criteria, URI-constant criteria and triple-constant criteria may be combined using operations $\sqcup$ and $\sqcap$. Our understanding of *constant reachability criteria* covers all criteria in the closure of such combinations:

Definition 4.9 (Constant Reachability Criterion). *Constant reachability criteria* are defined recursively as follows:

1. Any URI-constant criterion is a constant reachability criterion.

2. Any triple-constant criterion is a constant reachability criterion.

3. If c_1 and c_2 are constant reachability criteria, then both $c_1 \sqcup c_2$ and $c_1 \sqcap c_2$ are constant reachability criteria. $\square$

Since the set of all URIs, $\mathcal{U}$, is infinite, the number of finite subsets of $\mathcal{U}$ is also infinite and, thus, there exist infinitely many distinct URI-constant criteria; the same holds for triple-constant criteria. As a consequence, the set of all constant reachability criteria (that satisfy Definition 4.9) is also infinite.

We now show that any criterion in this set ensures finiteness (and there exist additional reachability criteria that ensure finiteness but are not constant by our definition):

Proposition 4.6. *If $\mathcal{C}_{\mathsf{const}}$ and $\mathcal{C}_{\mathsf{ef}}$ denote the infinite sets of all constant reachability criteria and all reachability criteria that ensure finiteness, respectively, then $\mathcal{C}_{\mathsf{const}} \subset \mathcal{C}_{\mathsf{ef}}$.*

Proof. To prove $\mathcal{C}_{\mathsf{const}} \subset \mathcal{C}_{\mathsf{ef}}$ we show (i) $c \in \mathcal{C}_{\mathsf{ef}}$ for all $c \in \mathcal{C}_{\mathsf{const}}$, and (ii) $\mathcal{C}_{\mathsf{const}} \neq \mathcal{C}_{\mathsf{ef}}$.

We first show $\mathcal{C}_{\mathsf{const}} \neq \mathcal{C}_{\mathsf{ef}}$ using the following counterexample: Let $c_{\text{uris}(P)}$ be a reachability criterion such that for each tuple $(t,u,P) \in \mathcal{T} \times \mathcal{U} \times \mathcal{P}$ it holds that $c_{\text{uris}(P)}(t,u,P) = \text{true}$ if and only if $u \in \text{uris}(P)$. Since uris(P) is finite for any given SPARQL expression $P \in \mathcal{P}$, it is easy to verify that $c_{\text{uris}(P)} \in \mathcal{C}_{\mathsf{ef}}$. On the other hand, there does not exist a constant reachability criterion that is the same as $c_{\text{uris}(P)}$, because, by definition, any constant reachability criterion ignores the given SPARQL expression, whereas the set of all possible data links accepted by $c_{\mathrm{uris}(P)}$ always depends on the given SPARQL expression. Hence, $c_{\mathrm{uris}(P)} \notin \mathcal{C}_{\mathsf{const}}$ and, thus, $\mathcal{C}_{\mathsf{const}} \neq \mathcal{C}_{\mathsf{ef}}$.

We now show $c \in \mathcal{C}_{\mathsf{ef}}$ for all $c \in \mathcal{C}_{\mathsf{const}}$. For the proof we use an induction on the definition of constant reachability criteria (that is, Definition 4.9).

Base case: The base case includes URI-constant criteria and triple-constant criteria (as defined in Definition 4.8, page 74). Given Lemma 4.1 (cf. page 73), it suffices to show for each such criterion c that set $X(c,P)$ is finite for all SPARQL expressions $P \in \mathcal{P}$.

W.l.o.g., let $P \in \mathcal{P}$ be an arbitrary SPARQL expression, and let c^U and c^T be an arbitrary URI-constant criterion and an arbitrary triple-constant criterion, respectively. Then, $|X(c^U,P)| = |U|$ and $|X(c^T,P)| \leq 3\,|T|$. Consequently, $X(c^U,P)$ and $X(c^T,P)$ are finite, because U and T are finite (as required by our definition of URI-constant criteria and triple-constant criteria; cf. Definition 4.8). Thus, $c^U \in \mathcal{C}_{\mathsf{ef}}$ and $c^T \in \mathcal{C}_{\mathsf{ef}}$.

Induction step: Let $c_1 \in \mathcal{C}_{\mathsf{const}}$ and $c_2 \in \mathcal{C}_{\mathsf{const}}$ be two constant reachability criteria such that $c_1 \in \mathcal{C}_{\mathsf{ef}}$ and $c_2 \in \mathcal{C}_{\mathsf{ef}}$. For any constant reachability criterion $c \in \mathcal{C}_{\mathsf{const}}$ that can be obtained by combining c_1 and c_2, we have to show $c \in \mathcal{C}_{\mathsf{ef}}$. Two such combinations are possible (cf. Definition 4.9): Either c is $c_1 \sqcup c_2$ or c is $c_1 \sqcap c_2$. In both cases, $c \in \mathcal{C}_{\mathsf{ef}}$ follows from Proposition 4.5 (cf. page 73). ■

We conclude our discussion of constant reachability criteria by interpreting them in terms of abstract algebra. By Definition 4.9, the set of all constant reachability criteria, $\mathcal{C}_{\mathsf{const}}$, is closed under $\sqcup$ and under $\sqcap$. Therefore, $\mathcal{C}_{\mathsf{const}}$ is a *subring* of our commutative ring $(\mathcal{C},\sqcap,\sqcup)$ (introduced in Section 4.3.2, page 71ff). However, this subring is a (commutative) *pseudo-ring* only; it has no multiplicative identity. That is, for the restriction of $\sqcup$ to $\mathcal{C}_{\mathsf{const}}$ there does not exist an identity element in $\mathcal{C}_{\mathsf{const}}$. In other words, the corresponding sublattice $(\mathcal{C}_{\mathsf{const}},\trianglelefteq)$ of the lattice $(\mathcal{C},\trianglelefteq)$, introduced in Section 4.3.2, has no top element (and, thus, is not bounded). To see this, consider our definition of URI-constant criteria and the fact that the set of all URIs is infinite; then, for any URI-constant criterion $c^U \in \mathcal{C}_{\mathsf{const}}$ there exists another URI-constant criterion $c^{U'} \in \mathcal{C}_{\mathsf{const}}$ such that $|U| < |U'|$. Hence, there exists no least restrictive URI-constant criterion (and, thus, no top element for $\mathcal{C}_{\mathsf{const}}$). Although sublattice $(\mathcal{C}_{\mathsf{const}},\trianglelefteq)$ is not bounded, we note that it is a *convex* sublattice of lattice $(\mathcal{C},\trianglelefteq)$. That is, for each triple $(c_1,c_2,c_3) \in \mathcal{C}\times\mathcal{C}\times\mathcal{C}$ the following property holds: If $c_1, c_3 \in \mathcal{C}_{\mathsf{const}}$ and $c_1 \trianglelefteq c_2 \trianglelefteq c_3$, then $c_2 \in \mathcal{C}_{\mathsf{const}}$.

4.4. Theoretical Properties

We now analyze theoretical properties of SPARQL$_{\mathsf{LD(R)}}$ queries. This analysis resembles our analysis of SPARQL$_{\mathsf{LD}}$ (cf. Section 3.3, page 42ff). That is, we use our computation model for the analysis and organize the discussion as follows: Section 4.4.1 focuses on the basic properties, Section 4.4.2 studies termination of (LD-machine-based) query computation, and Section 4.4.3 classifies SPARQL$_{\mathsf{LD(R)}}$ queries using the notions of finite computability and eventual computability. During this discussion we identify commonalities and differences between SPARQL$_{\mathsf{LD}}$ and SPARQL$_{\mathsf{LD(R)}}$. Section 4.5 summarizes the key points in which SPARQL$_{\mathsf{LD}}$ and SPARQL$_{\mathsf{LD(R)}}$ differ w.r.t. the analyzed properties.

4.4.1. Satisfiability, (Un)bounded Satisfiability, and Monotonicity

For the basic properties of a SPARQL$_{\mathsf{LD(R)}}$ query we show the following relationships:

Proposition 4.7. *Let $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query that uses SPARQL expression P and a nonempty set of (seed) URIs $S \subseteq \mathcal{U}$. The following relationships hold:*

1. *$\mathcal{Q}_c^{P,S}$ is satisfiable if and only if P is satisfiable.*

2. *$\mathcal{Q}_c^{P,S}$ is unboundedly satisfiable if and only if P is unboundedly satisfiable.*

3. *$\mathcal{Q}_c^{P,S}$ is boundedly satisfiable if and only if P is boundedly satisfiable.*

4. *$\mathcal{Q}_c^{P,S}$ is monotonic if P is monotonic.*[2]

Proving Proposition 4.7 is more complex than proving the corresponding result in the context of full-Web semantics (that is, Proposition 3.1, page 43). In the proof for the full-Web semantics case we construct Webs of Linked Data from sets of RDF triples. Since these sets may be infinitely large, we split up these sets and distribute their triples over multiple LD documents in the constructed Web (after dealing with their blank nodes). In the case of reachability-based semantics we cannot use such a construction because the LD documents that contain relevant RDF triples from the original, split up set may not be reachable. For this reason, we use an alternative approach for our proof of Proposition 4.7. This alternative is based on a particular notion of lineage defined for solutions in SPARQL query results. Informally, the lineage of such a solution μ is a subset of the queried set of RDF triples that is required to construct μ. Formally:

Definition 4.10 (Lineage). Let P be a SPARQL expression and G be a (potentially infinite) set of RDF triples. For every solution $\mu \in [\![P]\!]_G$ the *(P,G)-lineage of* μ, denoted by $\mathrm{lin}^{P,G}(\mu)$, is defined recursively as follows:

1. If P is a triple pattern tp, then $\mathrm{lin}^{P,G}(\mu) := \{\mu[tp]\}$.

2. If P is $(P_1\ \mathsf{AND}\ P_2)$, then $\mathrm{lin}^{P,G}(\mu) := \mathrm{lin}^{P_1,G}(\mu_1) \cup \mathrm{lin}^{P_2,G}(\mu_2)$, where $\mu_1 \in [\![P_1]\!]_G$ and $\mu_2 \in [\![P_2]\!]_G$ such that $\mu_1 \sim \mu_2$ and $\mu = \mu_1 \cup \mu_2$. (Since $\mu \in [\![P]\!]_G$, there exists a pair of valuations μ_1, μ_2 with the given properties.)

3. If P is $(P_1\ \mathsf{UNION}\ P_2)$, then

$$\mathrm{lin}^{P,G}(\mu) := \begin{cases} \mathrm{lin}^{P_1,G}(\mu_1) & \text{if } \exists\, \mu_1 \in [\![P_1]\!]_G : \mu_1 = \mu, \\ \mathrm{lin}^{P_2,G}(\mu_2) & \text{if } \exists\, \mu_2 \in [\![P_2]\!]_G : \mu_2 = \mu. \end{cases}$$

(If valuation μ_1 does not exist, then there exists valuation μ_2 because $\mu \in [\![P]\!]_G$.)

[2]Using the material conditional in the statement about monotonicity (instead of the material biconditional as used in the other three statements) is not a mistake. We elaborate more on this issue after proving Proposition 4.7.

4. If P is $(P_1 \text{ OPT } P_2)$, then

$$\text{lin}^{P,G}(\mu) := \begin{cases} \text{lin}^{P_1,G}(\mu_1) \cup \text{lin}^{P_2,G}(\mu_2) & \text{if } \exists\,(\mu_1,\mu_2) \in [\![P_1]\!]_G \times [\![P_2]\!]_G : \\ & \qquad (\mu_1 \sim \mu_2 \wedge \mu = \mu_1 \cup \mu_2), \\ \text{lin}^{P_1,G}(\mu') & \text{if } \exists\,\mu' \in [\![P_1]\!]_G : \\ & \qquad (\mu' = \mu \,\wedge\, \forall\,\mu^* \in [\![P_2]\!]_G : \mu^* \not\sim \mu'). \end{cases}$$

(Since $\mu \in [\![P]\!]_G$, either there exist valuations μ_1, μ_2, or there exists valuation μ'.)

5. If P is $(P' \text{ FILTER } R)$, then $\text{lin}^{P,G}(\mu) := \text{lin}^{P',G}(\mu')$ where $\mu' \in [\![P']\!]_G$ such that $\mu = \mu'$. (Valuation μ' exists because $\mu \in [\![P]\!]_G$.) □

Example 4.6. Consider an infinite set of RDF triples $G_{\mathsf{inf}} = \mathsf{AllData}(W_{\mathsf{inf}})$ that contains all RDF triples distributed over LD documents in our infinite example Web W_{inf} (as used in Examples 3.4 and 4.3 on page 58 and 65, respectively). That is, for each integer $i \in \mathbb{Z}$, identified by URI $\mathsf{no}_i \in \mathcal{U}$, set G_{inf} contains two RDF triples: $(\mathsf{no}_i, \mathsf{pred}, \mathsf{no}_{i-1}) \in G_{\mathsf{inf}}$ and $(\mathsf{no}_i, \mathsf{succ}, \mathsf{no}_{i+1}) \in G_{\mathsf{inf}}$. Let P_{ex} be the following SPARQL expression:

$$\Big(((?x, \mathsf{succ}, ?y) \text{ FILTER } ?x = \mathsf{no}_1) \text{ UNION } (\mathsf{no}_2, \mathsf{succ}, ?y)\Big).$$

There exist two solutions for P_{ex} in G_{inf}, namely $\mu_1 = \{?x \to \mathsf{no}_1, ?y \to \mathsf{no}_2\} \in [\![P_{\mathsf{ex}}]\!]_{G_{\mathsf{inf}}}$ and $\mu_2 = \{?y \to \mathsf{no}_3\} \in [\![P_{\mathsf{ex}}]\!]_{G_{\mathsf{inf}}}$. The $(P_{\mathsf{ex}}, G_{\mathsf{inf}})$-lineage of μ_1 consists of a single RDF triple: $\text{lin}^{P_{\mathsf{ex}},G_{\mathsf{inf}}}(\mu_1) = \{(\mathsf{no}_1, \mathsf{succ}, \mathsf{no}_2)\}$. Similarly, $\text{lin}^{P_{\mathsf{ex}},G_{\mathsf{inf}}}(\mu_2) = \{(\mathsf{no}_2, \mathsf{succ}, \mathsf{no}_3)\}$. □

Remark 4.1. If we let $G' = \text{lin}^{P,G}(\mu)$ for a SPARQL expression P, a potentially infinite set of RDF triples G, and a valuation $\mu \in [\![P]\!]_G$, then it follows from Definition 4.10 that (i) $G' \subseteq G$, (ii) G' is finite, and (iii) $\mu \in [\![P]\!]_{G'}$.

We now prove Proposition 4.7 by discussing its claims one after another:

Proof of Proposition 4.7, Claim 1 (Satisfiability). Let $\mathcal{Q}^{P,S}_c$ be a $\text{SPARQL}_{\mathsf{LD(R)}}$ query that uses SPARQL expression P and a nonempty set of seed URIs $S \subseteq \mathcal{U}$.

If: Suppose P is satisfiable. Then, there exists a set of RDF triples G such that $[\![P]\!]_G \neq \emptyset$. Let μ be an arbitrary solution for P in G, that is, $\mu \in [\![P]\!]_G$. Furthermore, let $G' = \text{lin}^{P,G}(\mu)$ be the (P,G)-lineage of μ. We use G' to construct a Web of Linked Data $W_\mu = (W_\mu, data_\mu, adoc_\mu)$ that consists of a single LD document. This document can be retrieved using any URI from the (nonempty) set of seed URI S of query $\mathcal{Q}^{P,S}_c$ and it contains the (P,G)-lineage of μ (which is finite). Formally:

$$D_\mu = \{d\} \qquad data_\mu(d) = G' \qquad \forall\, u \in \mathcal{U} : adoc_\mu(u) = \begin{cases} d & \text{if } u \in S, \\ \bot & \text{else.} \end{cases}$$

Due to our construction, $\mathsf{AllData}(W_\mu) = \mathsf{AllData}(R) = G'$ where R denotes the (S,c,P)-reachable subweb of W_μ. Then, by Definition 4.4 (cf. page 63), $\mathcal{Q}^{P,S}_c(W_\mu) = [\![P]\!]_{G'}$. Since $\mu \in [\![P]\!]_{G'}$, it holds that $\mathcal{Q}^{P,S}_c(W_\mu) \neq \emptyset$. Hence, $\mathcal{Q}^{P,S}_c$ is satisfiable.

Only if: Suppose SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ is satisfiable. Then, there exists a Web of Linked Data W such that $\mathcal{Q}_c^{P,S}(W) \neq \emptyset$. By Definition 4.4 (cf. page 63), we have $\mathcal{Q}_c^{P,S}(W) = [\![P]\!]_{\mathsf{AllData}(R)}$ where R denotes the (S,c,P)-reachable subweb of W. Thus, we may conclude that P is satisfiable. ■

Proof of Proposition 4.7, Claim 2 (Unbounded satisfiability). Let $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query that uses a nonempty set of seed URIs $S \subseteq \mathcal{U}$.

If: Suppose SPARQL expression P (used by $\mathcal{Q}_c^{P,S}$) is unboundedly satisfiable. W.l.o.g., let $k \in \{0,1,2,...\}$ be an arbitrary natural number. To prove that $\mathcal{Q}_c^{P,S}$ is unboundedly satisfiable it is sufficient to show that there exists a Web of Linked Data W such that $\left|\mathcal{Q}_c^{P,S}(W)\right| > k$. Since P is unboundedly satisfiable, there exists a set of RDF triples G such that $\left|[\![P]\!]_G\right| > k$. Let G be such a set and let $\Omega \subseteq [\![P]\!]_G$ be a subset of query result $[\![P]\!]_G$ such that $|\Omega| = k+1$ (such a subset exists because $\left|[\![P]\!]_G\right| > k$). Let G_Ω be the union of the (P,G)-lineages of all $\mu \in \Omega$, that is, $G_\Omega = \bigcup_{\mu\in\Omega} \mathrm{lin}^{P,G}(\mu)$. Then, $\Omega \subseteq [\![P]\!]_{G_\Omega}$ (cf. Remark 4.1). Furthermore, since Ω is finite and the (P,G)-lineage of each $\mu \in \Omega$ is finite, G_Ω is finite. Thus, we may construct a Web of Linked Data that consists of a single LD document with all RDF triples from G_Ω. Let $W_\Omega = (D_\Omega, data_\Omega, adoc_\Omega)$ with

$$D_\Omega = \{d\}, \qquad data_\Omega(d) = G_\Omega, \qquad \text{and} \qquad \forall\, u \in \mathcal{U} : adoc_\Omega(u) = \begin{cases} d & \text{if } u \in S, \\ \bot & \text{else,} \end{cases}$$

be such a Web of Linked Data. Based on our construction of this Web it holds that $\mathsf{AllData}(W_\Omega) = \mathsf{AllData}(R) = G_\Omega$ where R denotes the (S,c,P)-reachable subweb of W_Ω. Then, by Definition 4.4 (cf. page 63), we have $\mathcal{Q}_c^{P,S}(W_\Omega) = [\![P]\!]_{G_\Omega}$ and, because of $[\![P]\!]_{G_\Omega} = \Omega$, it thus holds that $\mathcal{Q}_c^{P,S}(W_\Omega) = \Omega$. Therefore, $\left|\mathcal{Q}_c^{P,S}(W_\Omega)\right| = k+1 > k$. Hence, W_Ω is a Web of Linked Data that shows that $\mathcal{Q}_c^{P,S}$ is unboundedly satisfiable.

Only if: Suppose SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ is unboundedly satisfiable. W.l.o.g., let $k \in \{0,1,2,...\}$ be an arbitrary natural number. To prove that SPARQL expression P (used by $\mathcal{Q}_c^{P,S}$) is unboundedly satisfiable it suffices to show that there exists a set of RDF triples G such that $\left|[\![P]\!]_G\right| > k$. Since $\mathcal{Q}_c^{P,S}$ is unboundedly satisfiable, there exists a Web of Linked Data W such that $\left|\mathcal{Q}_c^{P,S}(W)\right| > k$. Let R denote the (S,c,P)-reachable subweb of this Web W. By using $\mathcal{Q}_c^{P,S}(W) = [\![P]\!]_{\mathsf{AllData}(R)}$ (cf. Definition 4.4), we have that $\mathsf{AllData}(R)$ is such a set of RDF triples that we need to find for P. Hence, P is unboundedly satisfiable. ■

Proof of Proposition 4.7, Claim 3 (Bounded satisfiability). Claim 3 follows trivially from Claims 1 and 2: Suppose SPARQL expression P is boundedly satisfiable. In this case, P is satisfiable and not unboundedly satisfiable (cf. Section 3.2.1, page 38ff). By Claims 1 and 2, SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ (which uses P) is also satisfiable and not unboundedly satisfiable. Therefore, $\mathcal{Q}_c^{P,S}$ is boundedly satisfiable (cf. Definition 2.8, page 24). The same argument applies for the other direction of Claim 3. ■

Proof of Proposition 4.7, Claim 4 (Monotonicity). Let $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query that uses SPARQL expression P and a nonempty set of seed URIs $S \subseteq \mathcal{U}$.

Suppose SPARQL expression P is monotonic. Let W_1, W_2 be an arbitrary pair of Webs of Linked Data such that W_1 is a subweb of W_2. To prove that $\mathcal{Q}_c^{P,S}$ is monotonic it suffices to show $\mathcal{Q}_c^{P,S}(W_1) \subseteq \mathcal{Q}_c^{P,S}(W_2)$.

Let $R_1 = (D_{\mathfrak{R}1}, data_{\mathfrak{R}1}, adoc_{\mathfrak{R}1})$ and $R_2 = (D_{\mathfrak{R}2}, data_{\mathfrak{R}2}, adoc_{\mathfrak{R}2})$ denote the (S, c, P)-reachable subweb of W_1 and of W_2, respectively. Then, by Definition 4.4 (cf. page 63), $\mathcal{Q}_c^{P,S}(W_1) = [\![P]\!]_{\mathsf{AllData}(R_1)}$ and $\mathcal{Q}_c^{P,S}(W_2) = [\![P]\!]_{\mathsf{AllData}(R_2)}$. Furthermore, given that W_1 is a subweb of W_2, any LD document that is (c, P)-reachable from S in W_1 is also (c, P)-reachable from S in W_2. Therefore, R_1 is a subweb of R_2 and, thus, by Property 1 of Proposition 2.1 (cf. page 21), $\mathsf{AllData}(R_1) \subseteq \mathsf{AllData}(R_2)$. Thus, by using the monotonicity of P we have $[\![P]\!]_{\mathsf{AllData}(R_1)} \subseteq [\![P]\!]_{\mathsf{AllData}(R_2)}$. Hence, $\mathcal{Q}_c^{P,S}(W_1) \subseteq \mathcal{Q}_c^{P,S}(W_2)$. ■

This concludes our proof of Proposition 4.7. We emphasize that the proposition reveals a first major difference between SPARQL$_{\mathsf{LD(R)}}$ and SPARQL$_{\mathsf{LD}}$: The statement about monotonicity in Proposition 4.7 is a material conditional only, whereas it is a biconditional in the case of SPARQL$_{\mathsf{LD}}$ (cf. Proposition 3.1, page 43). The reason for this disparity is the existence of SPARQL$_{\mathsf{LD(R)}}$ queries for which monotonicity is independent of whether the corresponding SPARQL expression is monotonic. A simple example for such a case are SPARQL$_{\mathsf{LD(R)}}$ queries with a single seed URI under c_{None}-semantics:

Proposition 4.8. *Any SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}$ is monotonic if $|S| = 1$.*

Proof. Suppose $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}$ is a SPARQL$_{\mathsf{LD(R)}}$ query (under c_{None}-semantics) such that $|S| = 1$. Let u denote the single seed URI, that is, $u \in S = \{u\}$. W.l.o.g., let W_1, W_2 be an arbitrary pair of Webs of Linked Data such that W_1 is a subweb of W_2. To proof Proposition 4.8 we show $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_1) \subseteq \mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_2)$.

Let $W_1 = (D_1, data_1, adoc_1)$ and $W_2 = (D_2, data_2, adoc_2)$. Furthermore, R_1 denotes the $(S, c_{\mathsf{None}}, P)$-reachable subweb of W_1 and R_2 denotes the $(S, c_{\mathsf{None}}, P)$-reachable subweb of W_2. We distinguish the following four cases for seed URI u:

1. $adoc_1(u) = \perp$ and $adoc_2(u) = \perp$.
   In this case, R_1 and R_2 are equal to the empty Web (which contains no LD documents), respectively. Hence, $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_1) = \mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_2) = \emptyset$.

2. $adoc_1(u) = \perp$ and $adoc_2(u) = d$ with $d \in D_2$.
   In this case, R_1 is equal to the empty Web, whereas R_2 contains a single LD document, namely d. Hence, $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_1) = \emptyset$ and $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_2) = [\![P]\!]_{data_2(d)}$ and, thus, $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_1) \subseteq \mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_2)$.

3. $adoc_1(u) = d$ and $adoc_2(u) = d$ with $d \in D_1$ and, thus, $d \in D_2$ (because $D_1 \subseteq D_2$).
   In this case, both reachable subwebs, R_1 and R_2, contain a single LD document, namely d. Hence, $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_1) = \mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_2)$.

4. $adoc_1(u) \in d$ and $adoc_2(u) = \perp$ with $d \in D_1$.
   This case is impossible because W_1 is a subweb of W_2 (see Requirement 4 in Definition 2.3, page 18).

For all possible cases we have $\mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_1) \subseteq \mathcal{Q}_{c_{\mathsf{None}}}^{P,S}(W_2)$. ■

Proposition 4.8 verifies the impossibility for showing in general that SPARQL$_{\mathsf{LD(R)}}$ queries (with a nonempty set of seed URIs) are monotonic *only if* their SPARQL expression is monotonic. However, if we exclude queries whose reachability criterion ensures finiteness, then it is possible to show the dependency that is missing in Proposition 4.7:

Proposition 4.9. *Let $\mathcal{Q}^{P,S}_{c_{nf}}$ be a SPARQL$_{\mathsf{LD(R)}}$ query that uses SPARQL expression P, a nonempty set of (seed) URIs $S \subseteq \mathcal{U}$, and a reachability criterion c_{nf} that does not ensure finiteness. The following relationship holds:*

4. $\mathcal{Q}^{P,S}_{c_{nf}}$ is monotonic only if P is monotonic.*

Proof. Let $\mathcal{Q}^{P,S}_{c_{nf}}$ be a SPARQL$_{\mathsf{LD(R)}}$ query that uses SPARQL expression P, a nonempty set of seed URIs $S \subset \mathcal{U}$ and a reachability criterion c_{nf} which does not ensure finiteness.

Suppose $\mathcal{Q}^{P,S}_{c_{nf}}$ is monotonic. We have to show that the SPARQL expression P (used by $\mathcal{Q}^{P,S}_{c_{nf}}$) is monotonic as well. We distinguish two cases: P is satisfiable or P is unsatisfiable. In the latter case, P is trivially monotonic (cf. Property C.1, page 201). Hence, we only have to discuss the first case.

Let G_1, G_2 be an arbitrary pair of sets of RDF triples such that $G_1 \subseteq G_2$. To prove that (the satisfiable) P is monotonic it suffices to show $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$. Similar to the proof in the full-Web semantics case we construct two Webs of Linked Data W_1 and W_2 such that (i) W_1 is an induced subweb of W_2 and (ii) the data of G_1 and G_2 is distributed over W_1 and W_2, respectively. We then use W_1 and W_2 to show the monotonicity of P based on the monotonicity of $\mathcal{Q}^{P,S}_{c_{nf}}$.

We emphasize that this proof cannot be based on the notion of lineage which we use for proving the satisfiability-related claims in Proposition 4.7. Instead, we have to use an approach that resembles the approach that we use for monotonicity in the full-Web semantics case. We shall see that this is possible because reachability criterion c_{nf} does not ensure finiteness. However, the construction of W_1 and W_2 is more complex than the corresponding construction for the full-Web semantics case because we have to ensure reachability of all LD documents that contain RDF triples from G_1 and G_2.

As discussed in the context of Proposition 3.1, we may lose certain solutions of query results if we naively distribute RDF triples from G_1 and G_2 over separate LD documents in W_1 and W_2, respectively (recall, each LD document in a Web of Linked Data must use a unique set of blank nodes). We address this problem by applying the grounding isomorphism introduced for our proof of Proposition 3.1 (cf. Definition 3.2, page 44). That is, we let ϱ be a grounding isomorphism for G_2 and construct two sets of RDF triples, G'_1 and G'_2, by replacing the blank nodes in G_1 and in G_2 according to ϱ; i.e.,

$$G'_1 = \{\varrho[t] \mid t \in G_1\} \qquad \text{and} \qquad G'_2 = \{\varrho[t] \mid t \in G_2\}.$$

Then, $G'_1 \subseteq G'_2$ because of $G_1 \subseteq G_2$. Furthermore, for each $j \in \{1,2\}$, (i) $|G_j| = |G'_j|$, (ii) $\forall \mu \in [\![P]\!]_{G_j} : \varrho[\mu] \in [\![P]\!]_{G'_j}$, and (iii) $\forall \mu' \in [\![P]\!]_{G'_j} : \varrho^{-1}[\mu'] \in [\![P]\!]_{G_j}$ where ϱ^{-1} denotes the inverse of ϱ (cf. Property 3.1 and Property 3.2, page 44).

We aim to construct Webs W_1 and W_2 (by using G'_1 and G'_2) such that all LD documents that contain RDF triples from G'_1 and G'_2 are reachable. To achieve this goal we use a reachable subweb of another Web of Linked Data for the construction. This reachable subweb must be infinite because G_1 and G_2 may be (countably) infinite. To find a Web of Linked Data with such a reachable subweb we exploit the fact that query $\mathcal{Q}^{P,S}_{c_{nf}}$ uses a reachability criterion that does not ensure finiteness: Since c_{nf} does not ensure finiteness, there exist a Web of Linked Data $W^* = (D^*, data^*, adoc^*)$, a (finite, nonempty) set $S^* \subseteq \mathcal{U}$ of seed URIs, and a SPARQL expression P^* such that the (S^*, c_{nf}, P^*)-reachable subweb of W^* is infinite (cf. Definition 4.5, page 67). Notice, S^* and P^* are not necessarily the same as S and P.

While the (S^*, c_{nf}, P^*)-reachable subweb of W^* presents the basis for our construction of W_1 and W_2, we cannot use it directly because the data in that subweb may cause undesired side-effects for the evaluation of P. To avoid this issue we define an isomorphism ρ for W^*, S^*, and P^* such that the images of W^*, S^*, and P^* under ρ do not use any RDF term or query variable from G'_2 or from P.

To define ρ formally we need to introduce several symbols: First, we write U, L, and V to denote the sets of all URIs, literals, and variables in G'_2 and P, respectively (neither G'_2 nor P contain blank nodes). Formally:

$$\begin{aligned} U &= \big(\mathrm{terms}(G'_2) \cup \mathrm{terms}(P)\big) \cap \mathcal{U}, \\ L &= \big(\mathrm{terms}(G'_2) \cup \mathrm{terms}(P)\big) \cap \mathcal{L}, \text{ and} \\ V &= \mathrm{vars}(P) \cup \mathrm{vars}_{\mathsf{F}}(P), \end{aligned}$$

where $\mathrm{vars}_{\mathsf{F}}(P)$ denotes the set of all variables in all filter conditions of P (if any).

Similar to U, L, and V, we write U^*, L^*, and V^* to denote the sets of all URIs, literals, and variables in W^*, S^*, and P^*:

$$\begin{aligned} U^* &= S^* \cup \mathrm{terms}\big(\mathsf{AllData}(W^*)\big) \cap \mathcal{U}, \\ L^* &= \mathrm{terms}\big(\mathsf{AllData}(W^*)\big) \cap \mathcal{L}, \qquad \text{and} \\ V^* &= \mathrm{vars}(P^*) \cup \mathrm{vars}_{\mathsf{F}}(P^*)\,. \end{aligned}$$

Moreover, we assume three new sets of URIs, literals, and variables, denoted by U_{new}, L_{new}, and V_{new}, respectively, such that the following properties hold:

$$\begin{aligned} &U_{\mathsf{new}} \subseteq \mathcal{U} \text{ such that } |U_{\mathsf{new}}| = |U| \text{ and } U_{\mathsf{new}} \cap (U \cup U^*) = \emptyset; \\ &L_{\mathsf{new}} \subseteq \mathcal{L} \text{ such that } |L_{\mathsf{new}}| = |L| \text{ and } L_{\mathsf{new}} \cap (L \cup L^*) = \emptyset; \text{ and} \\ &V_{\mathsf{new}} \subseteq \mathcal{V} \text{ such that } |V_{\mathsf{new}}| = |V| \text{ and } V_{\mathsf{new}} \cap (V \cup V^*) = \emptyset\,. \end{aligned}$$

Furthermore, we assume three total, bijective mappings:

$$\rho_U : U \to U_{\mathsf{new}} \qquad \rho_L : L \to L_{\mathsf{new}} \qquad \rho_V : V \to V_{\mathsf{new}}\,.$$

Now we define ρ as a total, bijective mapping

$$\rho : \Big((\mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V}) \setminus (U_{\mathsf{new}} \cup L_{\mathsf{new}} \cup V_{\mathsf{new}}) \Big) \to \Big((\mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V}) \setminus (U \cup L \cup V) \Big)$$

such that, for each $x \in \mathrm{dom}(\rho)$,

$$\rho(x) = \begin{cases} \rho_U(x) & \text{if } x \in U, \\ \rho_L(x) & \text{if } x \in L, \\ \rho_V(x) & \text{if } x \in V, \\ x & \text{else.} \end{cases}$$

The *application* of isomorphism ρ to structures relevant for our proof is defined as follows:

- The application of ρ to a valuation μ, denoted by $\rho[\mu]$, results in a valuation μ' such that (i) $\mathrm{dom}(\mu') = \mathrm{dom}(\mu)$ and (ii) $\mu'(?v) = \rho\big(\mu(?v)\big)$ for all $?v \in \mathrm{dom}(\mu)$.
- The application of ρ to an RDF triple $t = (x_1, x_2, x_3)$, denoted by $\rho[t]$, results in an RDF triple (x'_1, x'_2, x'_3) such that $x'_i = \rho(x_i)$ for all $i \in \{1,2,3\}$.
- The application of ρ to the aforementioned Web $W^* = (D^*, \mathit{data}^*, \mathit{adoc}^*)$, denoted by $\rho[W^*]$, results in a Web of Linked Data $W^{*\prime} = (D^{*\prime}, \mathit{data}^{*\prime}, \mathit{adoc}^{*\prime})$ such that $D^{*\prime} = D^*$ and mappings $\mathit{data}^{*\prime}$ and $\mathit{adoc}^{*\prime}$ are defined as follows:

$$\forall\, d \in D^{*\prime}\colon\ \mathit{data}^{*\prime}(d) = \big\{\rho[t] \,\big|\, t \in \mathit{data}^*(d)\big\}$$
$$\forall\, u \in \mathcal{U}\colon\ \mathit{adoc}^{*\prime}(u) = \mathit{adoc}^*\big(\rho^{-1}(u)\big)$$

  where ρ^{-1} is the inverse of the bijective mapping ρ.
- The application of ρ to a (SPARQL) filter condition R, denoted by $\rho[R]$, results in a filter condition that is defined recursively as follows:
  1. If R is of the form $?x = c$, $?x = ?y$, or $\mathrm{bound}(?x)$, then $\rho[R]$ is of the form $?x' = c'$, $?x' = ?y'$, and $\mathrm{bound}(?x')$, respectively, where $?x' = \rho(?x)$, $?y' = \rho(?y)$, and $c' = \rho(c)$.
  2. If R is of the form $(\neg R_1)$, $(R_1 \wedge R_2)$, or $(R_1 \vee R_2)$, then $\rho[R]$ is of the form $(\neg R'_1)$, $(R'_1 \wedge R'_2)$, or $(R'_1 \vee R'_2)$, respectively, where $R'_1 = \rho[R_1]$ and $R'_2 = \rho[R_2]$.
- The application of ρ to an arbitrary SPARQL expression P', denoted by $\rho[P']$, results in a SPARQL expression that is defined recursively as follows:
  1. If P' is a triple pattern (x'_1, x'_2, x'_3), then $\rho[P']$ is (x''_1, x''_2, x''_3) where $x''_i = \rho(x'_i)$ for all $i \in \{1,2,3\}$.
  2. If P' is $(P'_1 \text{ AND } P'_2)$, $(P'_1 \text{ UNION } P'_2)$, $(P'_1 \text{ OPT } P'_2)$, or $(P'_1 \text{ FILTER } R')$, then $\rho[P']$ is $(P''_1 \text{ AND } P''_2)$, $(P''_1 \text{ UNION } P''_2)$, or $(P''_1 \text{ OPT } P''_2)$, and $(P''_1 \text{ FILTER } R'')$, respectively, where $P''_1 = \rho[P'_1]$, $P''_2 = \rho[P'_2]$, and $R'' = \rho[R']$.

We introduce $W^{*\prime}$, $S^{*\prime}$, and $P^{*\prime}$ as image of W^*, S^*, and P^* under ρ, respectively; i.e.,

$$W^{*\prime} = \rho[W^*]\,, \qquad S^{*\prime} = \{\rho(u) \,|\, u \in S^*\}\,, \qquad P^{*\prime} = \rho[P^*]\,.$$

Web of Linked Data $W^{*\prime}$ is structurally identical to W^*. Furthermore, the $(S^{*\prime}, c_{\mathit{nf}}, P^{*\prime})$-reachable subweb of $W^{*\prime}$ is infinite because the $(S^*, c_{\mathit{nf}}, P^*)$-reachable subweb of W^* is infinite. Let $R = (D_{\mathfrak{R}}, \mathit{data}_{\mathfrak{R}}, \mathit{adoc}_{\mathfrak{R}})$ be the $(S^{*\prime}, c_{\mathit{nf}}, P^{*\prime})$-reachable subweb of $W^{*\prime}$.

We now use R to construct Webs of Linked Data that contain all RDF triples from G'_1 and G'_2, respectively. Since R is infinite, there exists at least one infinite path in the link graph of R. Let $p = d_1, d_2, ...$ be such a path. Hence, for all $i \in \{1, 2, ...\}$,

$$d_i \in D_{\mathfrak{R}} \qquad \text{and} \qquad \exists\, t \in \mathit{data}_{\mathfrak{R}}(d_i) : \Big(\exists\, u \in \mathrm{uris}(t) : \mathit{adoc}_{\mathfrak{R}}(u) = d_{i+1}\Big)$$

We may use this path to construct Webs of Linked Data W_1 and W_2 from R such that W_1 and W_2 contain the data from G'_1 and G'_2, respectively. However, to allow us to use the monotonicity of SPARQL$_{\mathsf{LD(R)}}$ queries for our proof, it is necessary to construct W_1 and W_2 such that W_1 is an induced subweb of W_2. To achieve this goal we assume a strict total order on G'_2 such that each RDF triple $t \in G'_1 \subseteq G'_2$ comes before any RDF triple $t' \in G'_2 \setminus G'_1$ in that order. Formally, we denote this order by infix $<$ and, thus, require $t < t'$ for all $(t, t') \in G'_1 \times (G'_2 \setminus G'_1)$. Furthermore, we assume a total, injective function $\mathit{pdoc} : G'_2 \rightarrow \{d \in D_{\mathfrak{R}} \,|\, d \text{ is on path } p\}$ that is order-preserving, that is, for each pair $(t, t') \in G'_2 \times G'_2$, if $t < t'$, then LD document $\mathit{pdoc}(t)$ comes before LD document $\mathit{pdoc}(t')$ on path p.

We now use pdoc, G'_2, and $R = (D_{\mathfrak{R}}, \mathit{data}_{\mathfrak{R}}, \mathit{adoc}_{\mathfrak{R}})$ to construct a Web of Linked Data $W_2 = (D_2, \mathit{data}_2, \mathit{adoc}_2)$ with the following three elements:

$$\begin{aligned} D_2 &= D_{\mathfrak{R}}, \\ \forall\, d \in D_2 \colon\ \mathit{data}_2(d) &= \begin{cases} \mathit{data}_{\mathfrak{R}}(d) \cup \{t\} & \text{if } \exists\, t \in G'_2 : \mathit{pdoc}(t) = d, \\ \mathit{data}_{\mathfrak{R}}(d) & \text{else,} \end{cases} \\ \forall\, u \in \mathcal{U} \colon\ \mathit{adoc}_2(u) &= \mathit{adoc}_{\mathfrak{R}}(u)\,. \end{aligned}$$

In addition to W_2, we introduce a Web of Linked Data $W_1 = (D_1, \mathit{data}_1, \mathit{adoc}_1)$ that is an induced subweb of W_2, specified by:

$$D_1 := \big\{d \in D_2 \,\big|\, \text{either } d \text{ is not on path } p \text{ or } \exists\, t \in G'_1 : d = \mathit{pdoc}(t)\big\}\,.$$

(Recall that any induced subweb is specified unambiguously by defining its set of LD documents; cf. Proposition 2.1, page 21.)

The following properties are verified easily:

Property 4.2. *For any $j \in \{1, 2\}$, $G'_j \subset \mathsf{AllData}(W_j) = G'_j \cup \mathsf{AllData}(R)$.*

Property 4.3. *For any $j \in \{1, 2\}$, the $(S^{*\prime}, c_{nf}, P^{*\prime})$-reachable subweb of W_j is W_j itself.*

Property 4.4. *For any $j \in \{1, 2\}$, $[\![P]\!]_{G'_j} = [\![P]\!]_{\mathsf{AllData}(W_j)}$.*

We now consider a SPARQL expression $(P \text{ UNION } P^{*\prime})$. In the following, we write $\tilde{P}$ to denote this expression. Since G'_2 and the data in R have no RDF terms in common (i.e., $\mathrm{terms}(G'_2) \cap \mathrm{terms}(\mathsf{AllData}(R)) = \emptyset$), we conclude the following properties for $\tilde{P}$ w.r.t. the Webs of Linked Data W_1 and W_2 (that we constructed from G'_1, G'_2, and R):

Property 4.5. *For any $j \in \{1, 2\}$ the following three properties hold:*

1. *The $(S^{*\prime}, c_{nf}, \tilde{P})$-reachable subweb of W_j is W_j itself,*

2. $[\![P]\!]_{\mathsf{AllData}(W_j)} \cup [\![P^{*\prime}]\!]_{\mathsf{AllData}(W_j)} = [\![\tilde{P}]\!]_{\mathsf{AllData}(W_j)}$, *and*

3. $[\![P]\!]_{\mathsf{AllData}(W_j)} \cap [\![P^{*\prime}]\!]_{\mathsf{AllData}(W_j)} = \emptyset$.

Since (i) W_1 is an (induced) subweb of W_2, (ii) $\tilde{P}$ is (P UNION $P^{*\prime}$), and (iii) $\mathcal{Q}^{P,S}_{c_{nf}}$ is monotonic, we may use Property 4.5 and our definition of SPARQL$_{\mathsf{LD(R)}}$ queries (cf. Definition 4.4, page 63) to obtain the following inclusion:

$$\big(\mathcal{Q}^{\tilde{P},S^{*\prime}}_{c_{nf}}(W_1) \setminus [\![P^{*\prime}]\!]_{\mathsf{AllData}(W_1)}\big) \subseteq \big(\mathcal{Q}^{\tilde{P},S^{*\prime}}_{c_{nf}}(W_2) \setminus [\![P^{*\prime}]\!]_{\mathsf{AllData}(W_2)}\big)\,. \tag{4.1}$$

Finally, we now use W_1 and W_2 and the monotonicity of $\mathcal{Q}^{P,S}_{c_{nf}}$ to show $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$ (which proves that SPARQL expression P is monotonic). Let μ be an arbitrary solution for P in G_1, that is, $\mu \in [\![P]\!]_{G_1}$. Notice, such a solution exists because we assume P is satisfiable (see a). To prove $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$ it suffices to show $\mu \in [\![P]\!]_{G_2}$.

Due to Property 3.2 (cf. page 44) it holds that

$$\varrho[\mu] \in [\![P]\!]_{G_1'}$$

and with Property 4.4, Property 4.5, and the definition of SPARQL$_{\mathsf{LD(R)}}$ queries, we have:

$$\varrho[\mu] \in \big(\mathcal{Q}^{\tilde{P},S^{*\prime}}_{c_{nf}}(W_1) \setminus [\![P^{*\prime}]\!]_{\mathsf{AllData}(W_1)}\big)\,.$$

Due to the inclusion in (4.1), we obtain:

$$\varrho[\mu] \in \big(\mathcal{Q}^{\tilde{P},S^{*\prime}}_{c_{nf}}(W_2) \setminus [\![P^{*\prime}]\!]_{\mathsf{AllData}(W_2)}\big)\,.$$

We again use the definition of SPARQL$_{\mathsf{LD(R)}}$, Property 4.5, and Property 4.4 to show:

$$\varrho[\mu] \in [\![P]\!]_{G_2'}\,.$$

Finally, we use Property 3.2 again and find:

$$\varrho^{-1}\big[\varrho[\mu]\big] \in [\![P]\!]_{G_2}\,.$$

Since ϱ^{-1} is the inverse of bijective mapping ϱ, it holds that $\varrho^{-1}\big[\varrho[\mu]\big] = \mu$ and, thus, we have $\mu \in [\![P]\!]_{G_2}$. This concludes our proof of Proposition 4.9 (cf. page 81). ■

By using the relationships shown in Propositions 4.7 and 4.9, we may carry over our results on the undecidability of basic properties of SPARQL expressions (cf. Appendix C, page 195ff) to SPARQL$_{\mathsf{LD(R)}}$ queries:

Problem:	SATISFIABILITY(SPARQL$_{\mathrm{LD(R)}}$)
Input:	a SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}^{P,S}_{c}$
Question:	Is $\mathcal{Q}^{P,S}_{c}$ satisfiable?

Problem:	MONOTONICITY($\text{SPARQL}_{\text{LD(R)}}$)
Input:	a $\text{SPARQL}_{\text{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$
Question:	Is $\mathcal{Q}_c^{P,S}$ monotonic?

Problem:	BOUNDEDSATISFIABILITY($\text{SPARQL}_{\text{LD(R)}}$)
Input:	a $\text{SPARQL}_{\text{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$
Question:	Is $\mathcal{Q}_c^{P,S}$ boundedly satisfiable?

Problem:	UNBOUNDEDSATISFIABILITY($\text{SPARQL}_{\text{LD(R)}}$)
Input:	a $\text{SPARQL}_{\text{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$
Question:	Is $\mathcal{Q}_c^{P,S}$ unboundedly satisfiable?

Corollary 4.5. *Any of the four problems,* SATISFIABILITY($\text{SPARQL}_{\text{LD(R)}}$)*,* MONOTONICITY($\text{SPARQL}_{\text{LD(R)}}$)*,* BOUNDEDSATISFIABILITY($\text{SPARQL}_{\text{LD(R)}}$)*, and* UNBOUNDEDSATISFIABILITY($\text{SPARQL}_{\text{LD(R)}}$)*, is undecidable.*

Proof. We prove the undecidability of MONOTONICITY($\text{SPARQL}_{\text{LD(R)}}$). Since we may prove the undecidability of SATISFIABILITY($\text{SPARQL}_{\text{LD(R)}}$), of BOUNDEDSATISFIABILITY($\text{SPARQL}_{\text{LD(R)}}$), and of UNBOUNDEDSATISFIABILITY($\text{SPARQL}_{\text{LD(R)}}$) in an analogous manner, we omit these proofs.

To show that MONOTONICITY($\text{SPARQL}_{\text{LD(R)}}$) is undecidable, we use the undecidability of MONOTONICITY(SPARQL) (cf. Proposition C.5 on page 201). Hence, we reduce MONOTONICITY(SPARQL) to MONOTONICITY($\text{SPARQL}_{\text{LD(R)}}$). For the reduction we need a Turing computable function f that maps any possible input of MONOTONICITY(SPARQL) to a possible input of MONOTONICITY($\text{SPARQL}_{\text{LD(R)}}$). While the input of MONOTONICITY(SPARQL) is a SPARQL expression (cf. Section C.2, page 201ff), the input of MONOTONICITY($\text{SPARQL}_{\text{LD(R)}}$) is a $\text{SPARQL}_{\text{LD(R)}}$ query. We define f as follows: Let $u^* \in \mathcal{U}$ be an arbitrary URI and let c^* be an arbitrary reachability criterion that ensures finiteness. For any possible SPARQL expression P, $f(P)$ is $\text{SPARQL}_{\text{LD(R)}}$ query $\mathcal{Q}_{c^*}^{P,S^*}$ that uses SPARQL expression P, reachability criterion c^*, and the (nonempty) set of seed URIs $S^* := \{u^*\}$. If MONOTONICITY($\text{SPARQL}_{\text{LD(R)}}$) were decidable we could decide MONOTONICITY(SPARQL) because it holds: SPARQL expression P is monotonic if and only if $\text{SPARQL}_{\text{LD(R)}}$ query $f(P) = \mathcal{Q}_{c^*}^{P,S^*}$ is monotonic (cf. Propositions 4.7 and 4.9). However, since MONOTONICITY(SPARQL) is undecidable, MONOTONICITY($\text{SPARQL}_{\text{LD(R)}}$) must be undecidable as well. ■

While the basic properties are undecidable for $\text{SPARQL}_{\text{LD(R)}}$ queries (as they are for SPARQL expressions under standard SPARQL semantics, as well as for $\text{SPARQL}_{\text{LD}}$ queries), we obtain decidable fragments trivially by using the decidable fragments identified for SPARQL expressions in Appendix C (see, in particular, Table C.1 on page 196).

4.4.2. LD Machine Decidability of Termination

After discussing basic properties of $\text{SPARQL}_{\text{LD(R)}}$, we now focus on computation-related properties. This section discusses termination for an LD-machine-based computation of

SPARQL$_{\mathsf{LD(R)}}$ queries. Afterwards, in the next section, we identify the (LD-machine-based) computability of SPARQL$_{\mathsf{LD(R)}}$.

The termination problem for SPARQL$_{\mathsf{LD(R)}}$ is the following LD decision problem:

LD Problem:	TERMINATION(SPARQL$_{\mathrm{LD(R)}}$)
Web Input:	a Web of Linked Data W
Ordin. Input:	a SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$
Question:	Does there exist an LD machine M whose computation, with W encoded on the Web input tape of M, halts with an encoding of query result $\mathcal{Q}_c^{P,S}(W)$ on the output tape?

Section 3.3.2 discussed the termination problem for SPARQL$_{\mathsf{LD}}$ and showed that this problem is not LD machine decidable (cf. Theorem 3.1, page 52). Although we shall see that the same holds for TERMINATION(SPARQL$_{\mathrm{LD(R)}}$), we note a remarkable difference between SPARQL$_{\mathsf{LD}}$ and SPARQL$_{\mathsf{LD(R)}}$ w.r.t. termination of query computations: For SPARQL$_{\mathsf{LD}}$ there does not exist any (LD-machine-based) computation of an unboundedly satisfiable SPARQL$_{\mathsf{LD}}$ query that terminates with a complete query result (cf. Proposition 3.2, page 48). Such a general limitation does not exist for SPARQL$_{\mathsf{LD(R)}}$. Instead, for any unboundedly satisfiable SPARQL$_{\mathsf{LD(R)}}$ query there exist particular Webs of Linked Data in which a complete, terminating computation is possible:

Proposition 4.10. *Let $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query that is unboundedly satisfiable. There exists an LD machine that computes $\mathcal{Q}_c^{P,S}$ over any (potentially infinite) Web of Linked Data W and halts after a finite number of computation steps with an encoding of $\mathcal{Q}_c^{P,S}(W)$ on its output tape if and only if the (S,c,P)-reachable subweb of W is finite.*

Before we prove the proposition we recall that the finiteness of reachable subwebs, which the proposition establishes as a necessary and sufficient condition for complete, terminating computation, is not LD machine decidable (cf. Theorem 4.1 on page 68). Hence, in practice, we cannot use the given condition to decide about termination of any unboundedly satisfiable SPARQL$_{\mathsf{LD(R)}}$ query. However, a class of SPARQL$_{\mathsf{LD(R)}}$ queries for which the reachable subweb of any Web of Linked Data is finite, are queries whose reachability criterion has the finiteness property introduced by Definition 4.5 (cf. page 67). For these queries we generalize the previous proposition as follows:

Proposition 4.11. *Let $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query such that reachability criterion c ensures finiteness. There exists an LD machine that computes $\mathcal{Q}_c^{P,S}$ over any (potentially infinite) Web of Linked Data W and halts after a finite number of computation steps with an encoding of $\mathcal{Q}_c^{P,S}(W)$ on its output tape.*

In the remainder of this section we first prove Propositions 4.10 and 4.11. After these proofs we come back to the general termination problem for SPARQL$_{\mathsf{LD(R)}}$ and show that it is not LD machine decidable (refer to page 90 for the corresponding result).

To prove Propositions 4.10 and 4.11 we introduce a specific type of LD machine, called *two phase SPARQL$_{\mathsf{LD(R)}}$ machine* (or *2P machine*, for short). Such a machine exists for every SPARQL$_{\mathsf{LD(R)}}$ query and implements a generic computation of the particular query for which the machine is defined. We shall see that these computations consist of two main phases (hence the name of the machine). In the following, we define the notion of a 2P machine formally, discuss properties of these machines as they are relevant for our proofs, and then prove Propositions 4.10 and 4.11.

Definition 4.11 (2P Machine). Let $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query. The *2P machine* for $\mathcal{Q}_c^{P,S}$ is an LD machine (as per Definition 2.9, page 27) that implements Algorithm 4.1. This algorithm uses a special subroutine called *lookup*, which, when called with URI $u \in \mathcal{U}$, (i) writes $\mathrm{enc}(u)$ to the right end of the word on the lookup tape, (ii) enters the expand state, and (iii) performs the expand procedure as specified in Definition 2.9. □

Algorithm 4.1 Program of the 2P machine for SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$.

```
1: Call lookup for all URIs $u \in S$.
2: $expansionCompleted :=$ false
3: while $expansionCompleted =$ false do
4:     Scan the lookup tape (starting from the leftmost position) for an RDF triple $t$ and
       a URI $u \in \mathrm{uris}(t)$ such that (i) $c(t,u,P) =$ true and (ii) the word on the lookup
       tape neither contains $\mathrm{enc}(u)\,\mathrm{enc}(adoc(u))\,\sharp$ nor $\mathrm{enc}(u)\,\sharp$. If such $t$ and $u$ exist,
       call lookup for $u$; otherwise $expansionCompleted :=$ true.
5: end while
6: Let $G$ denote the set of all RDF triples currently encoded on the lookup tape. For
   each valuation $\mu \in [\![P]\!]_G$ append $\mathrm{enc}(\mu)$ to the word on the output tape.
```

As can be seen in Algorithm 4.1, the computation of a 2P machine consists of two separate phases: (i) a data retrieval phase (lines 1 to 5) and (ii) a subsequent, result computation phase (line 6). Data retrieval starts with an initialization (cf. line 1). After the initialization, the machine enters a (potentially nonterminating) loop that recursively discovers (i.e., expands) LD documents of the corresponding reachable subweb of the queried Web of Linked Data (encoded on the Web input tape of the machine). Since each scan of the lookup tape at line 4 starts from the leftmost position, the recursive discovery of LD documents resembles a breadth-first traversal of the reachable subweb. This breadth-first strategy guarantees that the 2P machine eventually discovers each of the reachable documents. The following lemma verifies this guarantee formally (the proof of this lemma can be found in Section E.4, page 222f).

Lemma 4.2. *Let M be the 2P machine for a SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$; let $W = (D, data, adoc)$ be a Web of Linked Data encoded on the Web input tape of M; and let $d \in D$ be an LD document that is (c,P)-reachable from S in W. During the execution of Algorithm 4.1 by M there exists an iteration of the loop (lines 3 to 5) after which the word on the lookup tape of M contains* $\mathrm{enc}(d)$ *permanently.*

While Lemma 4.2 shows that Algorithm 4.1 discovers all reachable LD documents, the following lemma verifies that the algorithm does not copy data from unreachable documents to the lookup tape (Section E.5 provides the proof of this lemma; cf. page 223f).

Lemma 4.3. *Let M be the 2P machine for a SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$; let W be a Web of Linked Data encoded on the Web input tape of M; and let R denote the (S, c, P)-reachable subweb of W. For any RDF triple t for which $\mathrm{enc}(t)$ eventually appears on the lookup tape of M during the execution of Algorithm 4.1 by M it holds that $t \in \mathsf{AllData}(R)$.*

After verifying that the data retrieval phase of Algorithm 4.1 (i.e., lines 1 to 5) is sound (cf. Lemma 4.3) and complete (cf. Lemma 4.2), we now show that an execution of the algorithm terminates if the corresponding reachable subweb of the input Web is finite.

Lemma 4.4. *Let M be the 2P machine for a SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ and let W be a Web of Linked Data encoded on the Web input tape of M. The computation of M halts after a finite number of steps if and only if the (S, c, P)-reachable subweb of W is finite.*

For the proof of Lemma 4.4 we refer to Section E.6 (cf. page 224f).

We are now ready to now use 2P machines to prove Proposition 4.10 (cf. page 87) and Proposition 4.11 (cf. page 87).

Proof of Proposition 4.10. Let $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query that is unboundedly satisfiable; let W be an arbitrary Web of Linked Data; and let $R = (D_{\mathfrak{R}}, data_{\mathfrak{R}}, adoc_{\mathfrak{R}})$ be the (S, c, P)-reachable subweb of W. We prove the proposition by showing that there exists an LD machine whose computation, with $\mathrm{enc}(W)$ on the Web input tape of that machine, halts after a finite number of computation steps with an encoding of $\mathcal{Q}_c^{P,S}(W)$ on its output tape if and only if R is finite.

If: Suppose R is finite. Based on Lemmas 4.2 to 4.4 it is easy to verify that the 2P machine for $\mathcal{Q}_c^{P,S}$ is an LD machine whose computation over W (encoded on the Web input tape of this 2P machine) halts after a finite number of computation steps with an encoding of $\mathcal{Q}_c^{P,S}(W)$ on the output tape of the 2P machine.

Only if: Let M be an LD machine (not necessarily a 2P machine) that computes $\mathcal{Q}_c^{P,S}(W)$ (given $\mathrm{enc}(W)$ on the Web input tape of M) and halts after a finite number of computation steps. We have to show that R is finite. We use proof by contradiction, that is, we assume R is infinite. In this case, $D_{\mathfrak{R}}$ is infinite. Since SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ is unboundedly satisfiable, machine M cannot assume a particular upper bound k for the cardinality of query result $\mathcal{Q}_c^{P,S}(W)$. Hence, in order to compute $\mathcal{Q}_c^{P,S}$ over W, machine M must (recursively) expand the word on its lookup tape until this word contains the encodings of (at least) each LD document in $D_{\mathfrak{R}}$. Such an expansion is necessary to ensure that the computed query result is complete. Since $D_{\mathfrak{R}}$ is infinite the expansion requires infinitely many computing steps. However, we know that M halts after a finite number of computation steps. Hence, we have a contradiction and, thus, R must be finite. ■

Proof of Proposition 4.11. Let $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query such that reachability criterion c ensures finiteness. We have to show that there exists an LD machine that computes $\mathcal{Q}_c^{P,S}$ over any Web of Linked Data W and halts after a finite number of computation steps with an encoding of $\mathcal{Q}_c^{P,S}(W)$ on its output tape. Based on Lemmas 4.2 to 4.4 it is easy to verify that the 2P machine for $\mathcal{Q}_c^{P,S}$ is such a machine (notice, 2P machines are not restricted to SPARQL$_{\mathsf{LD(R)}}$ queries that are unboundedly satisfiable, neither are Lemmas 4.2 to 4.4). ■

Proposition 4.11 shows that answering the question posed by the LD decision problem Termination($\mathrm{SPARQL}_{\mathrm{LD(R)}}$) (cf. page 87) is trivial if the given $\mathrm{SPARQL}_{\mathsf{LD(R)}}$ query uses a reachability criterion that ensures finiteness. In general, however, the problem is undecidable for LD machines.

Theorem 4.2. Termination($\mathrm{SPARQL}_{\mathrm{LD(R)}}$) *is not LD machine decidable.*

Proof. To prove that Termination($\mathrm{SPARQL}_{\mathrm{LD(R)}}$) is not LD machine decidable we reduce the halting problem to Termination($\mathrm{SPARQL}_{\mathrm{LD(R)}}$). For this reduction we use the same argumentation, including the same Web of Linked Data, as used for proving Proposition 3.4 (which shows that Termination(BS-$\mathrm{SPARQL}_{\mathrm{LD}}$) is not LD machine decidable; cf. page 50).

We define the mapping from any possible input for the halting problem to an input for Termination($\mathrm{SPARQL}_{\mathrm{LD(R)}}$) as follows: Let (w, x) be an input to the halting problem, that is, w is the description of a Turing machine $M(w)$ and x is a possible input word for $M(w)$; then, $f(w,x) := (W_{\mathsf{TMs}}, \mathcal{Q}^{P_{w,x},S_{w,x}}_{c_{\mathsf{All}}})$ where: (i) W_{TMs} is the (infinite) Web of Linked Data defined in the proof of Proposition 3.4 (cf. page 50 in Section 3.3.2), (ii) SPARQL expression $P_{w,x}$ is the triple pattern $(u^{w,x}, \mathsf{type}, \mathsf{TerminatingComputation})$, and (iii) $S_{w,x} := \{u_1^{w,x}\}$ (where $u^{w,x}$ denotes a URI that identifies the computation of Turing machine $M(w)$ on input x and $u_1^{w,x}$ denotes a URI that identifies the first step in this computation). As in the proof of Proposition 3.4, f is computable by Turing machines (and, thus, by LD machines).

To show that Termination($\mathrm{SPARQL}_{\mathrm{LD(R)}}$) is not LD machine decidable, suppose it is LD machine decidable. Then, an LD machine decides the halting problem for any input (w, x): Turing machine $M(w)$ halts on input x if and only if there exists an LD machine that computes $\mathcal{Q}^{P_{w,x},S_{w,x}}_{c_{\mathsf{All}}}(W_{\mathsf{TMs}})$ and halts. However, the halting problem is undecidable for Turing machines and, thus, for LD machines. Hence, we have a contradiction. As a consequence, Termination($\mathrm{SPARQL}_{\mathrm{LD(R)}}$) cannot be LD machine decidable. ■

Proving that Termination($\mathrm{SPARQL}_{\mathrm{LD(R)}}$) is not LD machine decidable concludes our discussion of the termination problem for $\mathrm{SPARQL}_{\mathsf{LD(R)}}$. While the results in this section are related to termination, they also present a basis for proving computability-related results in the following section.

4.4.3. LD Machine Computability

We now classify $\mathrm{SPARQL}_{\mathsf{LD(R)}}$ queries using the notions of finite computability and eventual computability (cf. Section 2.2.3, page 29). Our discussion focuses on satisfiable queries because unsatisfiable queries are trivially finitely computable by an LD machine (cf. Proposition 2.3, page 29).

From the previous section we know that $\mathrm{SPARQL}_{\mathsf{LD(R)}}$ queries whose reachability criterion ensures finiteness present a special case w.r.t. LD-machine-based computability: For each of these queries there exists an LD machine that performs a complete, terminating computation of the query over any Web of Linked Data (cf. Proposition 4.11, page 87). As a result, these queries are finitely computable by an LD machine.

Proposition 4.12. *Let c be a reachability criterion that ensures finiteness, then any SPARQL$_{\mathsf{LD(R)}}$ query under c-semantics is finitely computable by an LD machine.*

Proof. Proposition 4.12 follows readily from Proposition 4.11 (compare the property shown by Proposition 4.11, page 87, to the requirements for finitely computable Linked Data queries given in Definition 2.10, page 29). ■

Proposition 4.12 reveals another difference between SPARQL$_{\mathsf{LD(R)}}$ and SPARQL$_{\mathsf{LD}}$: The proposition shows (indirectly) that LD-machine-based computability is independent of monotonicity for all SPARQL$_{\mathsf{LD(R)}}$ queries whose reachability criterion ensures finiteness. Such an independence does not exist for any (satisfiable) SPARQL$_{\mathsf{LD}}$ query (cf. Theorem 3.2, page 53).

However, even in the case of SPARQL$_{\mathsf{LD(R)}}$ this independence exists only for reachability criteria that ensure finiteness. For any other criterion (including c_{Match} and c_{All}), we may show a similar relationship between monotonicity and computability as Theorem 3.2 shows in the context of SPARQL$_{\mathsf{LD}}$:

Theorem 4.3. *Let c^* be a reachability criterion that does not ensure finiteness. If a satisfiable SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}^{P,S}_{c^*}$ (under c^*-semantics) is monotonic, then $\mathcal{Q}^{P,S}_{c^*}$ is either finitely computable or eventually computable by an LD machine; otherwise, $\mathcal{Q}^{P,S}_{c^*}$ may not even be eventually computable by an LD machine.*

The remainder of this section is dedicated to the proof of Theorem 4.3.

Proof. Let $\mathcal{Q}^{P^*,S^*}_{c^*}$ be an arbitrary satisfiable SPARQL$_{\mathsf{LD(R)}}$ query under c^*-semantics (c^* does not ensure finiteness). To prove Theorem 4.3 we distinguish four cases:

(1) The (S^*, c^*, P^*)-reachable subweb of *every* Web of Linked Data is finite (which is possible even if c^* does not ensure finiteness; cf. Definition 4.5, page 67).

(2) The (S^*, c^*, P^*)-reachable subweb of some Web of Linked Data is infinite and $\mathcal{Q}^{P^*,S^*}_{c^*}$ is monotonic.

(3) The (S^*, c^*, P^*)-reachable subweb of some Web of Linked Data is infinite, $\mathcal{Q}^{P^*,S^*}_{c^*}$ is *not* monotonic, and there exists a Web of Linked Data W in which the (S^*, c^*, P^*)-reachable subweb is infinite such that $\mathcal{Q}^{P^*,S^*}_{c^*}(W) \neq \emptyset$.

(4) The (S^*, c^*, P^*)-reachable subweb of some Web of Linked Data is infinite, $\mathcal{Q}^{P^*,S^*}_{c^*}$ is *not* monotonic, and for *each* Web of Linked Data W in which the (S^*, c^*, P^*)-reachable subweb is infinite, it holds that $\mathcal{Q}^{P^*,S^*}_{c^*}(W) = \emptyset$.

We shall see that, in case (1), query $\mathcal{Q}^{P^*,S^*}_{c^*}$ is finitely computable by an LD machine; in case (2), $\mathcal{Q}^{P^*,S^*}_{c^*}$ is at least eventually computable by an LD machine; in case (3), $\mathcal{Q}^{P^*,S^*}_{c^*}$ is not even eventually computable by an LD machine; and, in case (4), $\mathcal{Q}^{P^*,S^*}_{c^*}$ is eventually computable by an LD machine. In the following we discuss each of these cases.

Case (1): Suppose the (S^*, c^*, P^*)-reachable subweb of *every* Web of Linked Data is finite. We claim that in this case query $\mathcal{Q}^{P^*,S^*}_{c^*}$ is finitely computable by an LD machine (independent of whether the query monotonic). To prove this claim we use the same argument as we use for proving Proposition 4.11 (cf. page 87): Based on Lemmas 4.2 to 4.4 (cf. Section 4.4.2, page 86ff) and based on the fact that the (S^*, c^*, P^*)-reachable subweb of every Web of Linked Data is finite, it is easy to verify that the 2P machine for $\mathcal{Q}_{c^*}^{P^*,S^*}$ is an LD machine that computes $\mathcal{Q}_{c^*}^{P^*,S^*}$ over any Web of Linked Data W and halts after a finite number of computation steps (with an encoding of $\mathcal{Q}_{c^*}^{P^*,S^*}(W)$ on its output tape). Hence, the 2P machine for $\mathcal{Q}_{c^*}^{P^*,S^*}$ satisfies the requirements in our definition of finitely computable Linked Data queries (cf. Definition 2.10, page 29) and, thus, $\mathcal{Q}_{c^*}^{P^*,S^*}$ is finitely computable.

Case (2): Suppose query $\mathcal{Q}_{c^*}^{P^*,S^*}$ is monotonic and there exists a Web of Linked Data W^* such that the (S^*, c^*, P^*)-reachable subweb of W^* is infinite. We show that $\mathcal{Q}_{c^*}^{P^*,S^*}$ is eventually computable by an LD machine. However, for this case we cannot use the 2P machine for $\mathcal{Q}_{c^*}^{P^*,S^*}$ because the computation of this machine on Web input W^* does not have the two properties given in our definition of eventually computable Linked Data queries (cf. Definition 2.11, page 29). More precisely, during this particular computation the data retrieval phase (i.e., the loop in lines 3 to 5 of Algorithm 4.1, page 88) would not terminate (because the (S^*, c^*, P^*)-reachable subweb of W^* is infinite). As a result, this computation cannot report any solution for $\mathcal{Q}_{c^*}^{P^*,S^*}$ in W^* (if there is any).

Hence, to show that $\mathcal{Q}_{c^*}^{P^*,S^*}$ is eventually computable by an LD machine we introduce another type of LD machine called *early reporting SPARQL$_{\mathsf{LD(R)}}$ machine* (*ER machine*). As the name suggests, such a machine reports any solution (for its SPARQL$_{\mathsf{LD(R)}}$ query over the given input Web) as early as possible. We emphasize that this strategy is sound only for monotonic queries (thus, we cannot use an ER machine for previous proofs that are based on a 2P machine). We define an ER machine as follows:

Definition 4.12 (ER Machine). Let $\mathcal{Q}_c^{P,S}$ be a SPARQL$_{\mathsf{LD(R)}}$ query. The *ER machine* for $\mathcal{Q}_c^{P,S}$ is an LD machine (as per Definition 2.9, page 27) that implements Algorithm 4.2. This algorithm uses a special subroutine called *lookup*, which, when called with a URI $u \in \mathcal{U}$, (i) writes enc(u) to the right end of the word on the lookup tape, (ii) enters the expand state, and (iii) performs the expand procedure as specified in Definition 2.9. □

As can be seen in Algorithm 4.2, the computation of an ER machine starts with an initialization (cf. line 1). After the initialization, the machine enters a (potentially non-terminating) loop. During each iteration of this loop, the machine generates valuations using all data that is currently encoded on the lookup tape. The following lemma shows that these valuations are part of the corresponding query result:

Lemma 4.5. *Let M be the ER machine for a* monotonic *SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ and let W be a Web of Linked Data encoded on the Web input tape of M. During the execution of Algorithm 4.2 by M it holds that $[\![P]\!]_{T_j} \subseteq \mathcal{Q}_c^{P,S}(W)$ for all $j \in \{1, 2, \dots\}$.*

The proof of Lemma 4.5 can be found in Section E.7 (cf. page 225f); unsurprisingly, this proof resembles the proofs of Lemmas 3.1 and 4.3 (which show soundness of full-Web machines and of 2P machines, respectively; cf. page 54 and page 89).

Lemma 4.5 provides a basis for proving the soundness of (monotonic) query results computed by an ER machine. To verify the completeness of these results it is important

Algorithm 4.2 Program of the ER machine for SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$.

```
1: Call *lookup* for all URIs $u \in S$.
2: **for** $j = 1, 2, \dots$ **do**
3:     Use the work tape to enumerate the set $[\![P]\!]_{T_j}$, where $T_j$ denotes the set of all RDF triples currently encoded on the lookup tape.
4:     For each valuation $\mu \in [\![P]\!]_{T_j}$, check whether $\mu$ is already encoded on the output tape; if this is not the case, then append $\mathrm{enc}(\mu)$ to the word on the output tape.
5:     Scan the lookup tape (starting from the leftmost position) for an RDF triple $t$ that contains a URI $u \in \mathrm{uris}(t)$ such that (i) $c(t, u, P) = \mathrm{true}$ and (ii) the word on the lookup tape neither contains $\mathrm{enc}(u)\,\mathrm{enc}(adoc(u))\,\sharp$ nor $\mathrm{enc}(u)\,\sharp$. If such $t$ and $u$ exist, call subroutine *lookup* for $u$; otherwise halt the computation.
6: **end for**
```

to note that such a machine looks up no more than one URI per iteration (cf. line 5 in Algorithm 4.2). Hence, in contrast to a 2P machine, an ER machine prioritizes result construction over data retrieval. On the other hand, similar to a 2P machine, the data retrieval strategy performed by an ER machine traverses the reachable subweb of the queried Web of Linked Data in a breadth-first manner (because each scan of the lookup tape at line 5 in Algorithm 4.2 starts from the leftmost position). Due to these two properties we may show that for each solution in a query result there exists an iteration during which the ER machine computes this solution:

Lemma 4.6. *Let M be the ER machine for a* monotonic *SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ whose reachability criterion c does* not *ensure finiteness; and let W be a Web of Linked Data encoded on the Web input tape of M. For each solution $\mu \in \mathcal{Q}_c^{P,S}(W)$ there exists a $j_\mu \in \{1, 2, \dots\}$ such that during the execution of Algorithm 4.2 by M it holds that $\mu \in [\![P]\!]_{T_j}$ for all $j \in \{j_\mu, j_\mu{+}1, \dots\}$.*

We prove Lemma 4.6 in Section E.8 (cf. page 226ff). The proof resembles our proofs of the corresponding lemmas for full-Web machines and for 2P machines (cf. Lemma 3.2 on page 54 and Lemma 4.2 on page 88, respectively).

So far our results verify that (i) the set of query solutions computed after any iteration of the loop in Algorithm 4.2 is sound, and (ii) eventually, this set covers the query result completely. The following lemma shows that each iteration requires a finite number of computation steps only (for the proof of this lemma refer to Section E.9, page 228f).

Lemma 4.7. *Let M be the ER machine for a SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ and let W be a Web of Linked Data encoded on the Web input tape of M. During the execution of Algorithm 4.2, M completes each iteration of the loop in Algorithm 4.2 after a finite number of computation steps.*

Altogether, Lemmas 4.5 to 4.7 conclude the discussion of case (2) of our proof of Theorem 4.3. That is, based on these lemmas it is easy to verify that the ER machine for our SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_{c^*}^{P^*,S^*}$ satisfies the requirements in our definition of eventually computable Linked Data queries (cf. Definition 2.11, page 29) and, thus, query $\mathcal{Q}_{c^*}^{P^*,S^*}$ is eventually computable by an LD machine.

Case (3): Suppose query $\mathcal{Q}_{c^*}^{P^*,S^*}$ is *not* monotonic, and there exists a Web of Linked Data W such that (i) the (S^*, c^*, P^*)-reachable subweb of W is infinite and (ii) $\mathcal{Q}_{c^*}^{P^*,S^*}(W) \neq \emptyset$. Let W^* be such a Web of Linked Data.

To show that $\mathcal{Q}_{c^*}^{P^*,S^*}$ is not even eventually computable by an LD machine we use the same argument as used for the corresponding discussion of non-monotonic SPARQL$_{\mathsf{LD}}$ queries (see the proof of Theorem 3.2 on page 53). That is, we show a contradiction by assuming that query $\mathcal{Q}_{c^*}^{P^*,S^*}$ is (at least) eventually computable by an LD machine. Hence, for the proof we assume an LD machine M (which is not necessarily an ER machine, nor a 2P machine) whose computation of $\mathcal{Q}_{c^*}^{P^*,S^*}$ on any Web of Linked Data (encoded on the Web input tape of M) has the two properties given in our definition of eventually computable Linked Data queries (cf. Definition 2.11, page 29).

Let W^* be encoded on the Web input tape of M and let μ be an arbitrary solution for $\mathcal{Q}_{c^*}^{P^*,S^*}$ in W^*; i.e., $\mu \in \mathcal{Q}_{c^*}^{P^*,S^*}(W^*)$. Based on our assumption, machine M writes enc(μ) to its output tape after a finite number of computation steps (cf. property 2 in Definition 2.11). We argue that this is impossible: Since $\mathcal{Q}_{c^*}^{P^*,S^*}$ is not monotonic, M may not add enc(μ) to the output before M has accessed all LD documents that are (c^*, P^*)-reachable from S^* in W^*. However, due to the infiniteness of the (S^*, c^*, P^*)-reachable subweb of W^*, there exists an infinite number of such documents. Therefore, accessing all these documents is a nonterminating process and, thus, M cannot write enc(μ) to its output after a finite number of computation steps. As a consequence, the computation of $\mathcal{Q}_{c^*}^{P^*,S^*}$ (over W^*) by M does not have the properties given in Definition 2.11, which contradicts our initial assumption. Due to this contradiction we conclude that $\mathcal{Q}_{c^*}^{P^*,S^*}$ is not eventually computable by an LD machine.

Case (4): Suppose (i) query $\mathcal{Q}_{c^*}^{P^*,S^*}$ is *not* monotonic, (ii) for *each* Web of Linked Data W for which the (S^*, c^*, P^*)-reachable subweb of W is infinite it holds that $\mathcal{Q}_{c^*}^{P^*,S^*}(W) = \emptyset$, and (iii) there exists at least one such Web of Linked Data (in which the (S^*, c^*, P^*)-reachable subweb is infinite). To show that $\mathcal{Q}_{c^*}^{P^*,S^*}$ is eventually computable by an LD machine we use the 2P machine for $\mathcal{Q}_{c^*}^{P^*,S^*}$ (cf. Definition 4.11, page 88). We write M to denote this machine. To consider all possible Webs of Linked Data we distinguish the following two cases:

First, we consider Webs of Linked Data whose (S^*, c^*, P^*)-reachable subweb is *finite*. Let W be such a Web of Linked Data. Lemmas 4.2 to 4.4 (cf. Section 4.4.2, page 86ff) verify that machine M, with enc(W) on its Web input tape, halts after a finite number of computation steps with a possible encoding of query result $\mathcal{Q}_{c^*}^{P^*,S^*}(W)$ on its output tape. Hence, for W, the computation of M has the properties required for eventually computable Linked Data queries (cf. Definition 2.11, page 29). In fact, the computation of M even has the more restrictive properties required for finitely computable Linked Data queries (cf. Definition 2.10, page 29).

Second, we consider Webs of Linked Data whose (S^*, c^*, P^*)-reachable subweb is *infinite*. Recall that the result of query $\mathcal{Q}_{c^*}^{P^*,S^*}$ in those Webs is empty. Let W be such a Web of Linked Data. Since the (S^*, c^*, P^*)-reachable subweb of W is infinite, the computation of M on Web input W does not halt. More precisely, the data retrieval phase in Algorithm 4.1 never terminates (cf. page 88). Nonetheless, due to $\mathcal{Q}_{c^*}^{P^*,S^*}(W) = \emptyset$, the computation of M on Web input W has the properties required by Definition 2.11. ■

4.5. Differences between SPARQL$_{\mathsf{LD}}$ and SPARQL$_{\mathsf{LD(R)}}$

We conclude the analysis of properties of SPARQL$_{\mathsf{LD(R)}}$ queries by comparing our results to the corresponding results for SPARQL$_{\mathsf{LD}}$ queries established in the previous chapter (cf. Section 3.3, page 42ff). In particular, we focus on the main differences between SPARQL$_{\mathsf{LD}}$ and SPARQL$_{\mathsf{LD(R)}}$ w.r.t. the analyzed properties.

By comparing Theorem 4.3 (cf. page 91) and Theorem 3.2 (cf. page 53) we notice that the relationship between computability and monotonicity is not as definite for (satisfiable) SPARQL$_{\mathsf{LD(R)}}$ queries (whose reachability criterion does not ensure finiteness) as it is for (satisfiable) SPARQL$_{\mathsf{LD}}$ queries. The reason why Theorem 4.3 does not present a more distinctive statement is the possibility of the four different cases discussed in our proof of the theorem. As a result, there exist satisfiable SPARQL$_{\mathsf{LD(R)}}$ queries that feature the same limited computability as their SPARQL$_{\mathsf{LD}}$ counterparts. However, the reasons for such limitation in each of the two cases differ significantly: In case of SPARQL$_{\mathsf{LD}}$ the limitation can be attributed to the infiniteness of the set of all URIs, whereas, for SPARQL$_{\mathsf{LD(R)}}$, the limitation is a consequence of the possibility to query an infinitely large Web of Linked Data.

In addition to our main computability-related result in Theorem 4.3, we also identified a class of SPARQL$_{\mathsf{LD(R)}}$ queries that present a special case w.r.t. (LD-machine-based) computability: Any SPARQL$_{\mathsf{LD(R)}}$ query whose reachability criterion ensures finiteness is finitely computable by an LD machine (cf. Proposition 4.12, page 91). Hence, for these queries, computational feasibility is independent of monotonicity. Such an independence does not exist for any (satisfiable) SPARQL$_{\mathsf{LD}}$ query (cf. Theorem 3.2). Table 4.1 summarizes our computability-related results.

In addition to differences w.r.t. (LD-machine-based) computability, we identified further differences between SPARQL$_{\mathsf{LD}}$ and SPARQL$_{\mathsf{LD(R)}}$:

- LD-machine-based computation of any unboundedly satisfiable SPARQL$_{\mathsf{LD}}$ query cannot terminate with a guarantee for complete query results (cf. Proposition 3.2, page 48). For an unboundedly satisfiable SPARQL$_{\mathsf{LD(R)}}$ query, in contrast, an LD-machine-based computation over some Webs of Linked Data may terminate (with a complete query result), even if the query is not finitely computable by an LD machine; this includes all finite Webs of Linked Data but also some infinite Webs (cf. Corollary 4.1, page 65, and Proposition 4.10, page 87).

- While the monotonicity of any SPARQL$_{\mathsf{LD}}$ query is correlated with the monotonicity of the SPARQL expression used by the query, such a relationship does not exist for all SPARQL$_{\mathsf{LD(R)}}$ queries (compare Proposition 3.1, page 43, and Proposition 4.7, page 77). Instead, there exist SPARQL$_{\mathsf{LD(R)}}$ queries that are monotonic regardless of whether their SPARQL expression is monotonic or non-monotonic (cf. Proposition 4.8, page 80). However, for SPARQL$_{\mathsf{LD(R)}}$ queries whose reachability criterion does not ensure finiteness, we have the same correlation as we have for all SPARQL$_{\mathsf{LD}}$ queries (cf. Proposition 4.9, page 81). Remarkably, the SPARQL$_{\mathsf{LD(R)}}$ queries covered by this result are exactly those queries for which monotonicity has an impact on LD-machine-based computability (cf. Table 4.1).

	finitely computable	**event. computable**	**not even event. comp.**	**Corresponding result**
SPARQL$_{\mathsf{LD}}$				
- unsatisfiable	all			Proposition 2.3, p.29
- satisfiable, monotonic		all		Theorem 3.2, p.53
- non-monotonic			all	Theorem 3.2, p.53
SPARQL$_{\mathsf{LD(R)}}$ *(reachability criterion does not ensure finiteness)*				
- unsatisfiable	all			Proposition 2.3, p.29
- satisfiable, monotonic	some	some		Theorem 4.3, p.91
- non-monotonic	some	some	some	Theorem 4.3, p.91
SPARQL$_{\mathsf{LD(R)}}$ *(reachability criterion ensures finiteness)*				
- unsatisfiable	all			Proposition 2.3, p.29
- satisfiable, monotonic	all			Proposition 4.12, p.91
- non-monotonic	all			Proposition 4.12, p.91

Table 4.1.: Correlation between (LD-machine-based) computability and basic properties for all SPARQL-based Linked Data queries considered in this dissertation.

Part II.

Execution of Queries over a Web of Linked Data

5. Overview of Query Execution Techniques

In the previous chapters we discuss theoretical foundations of queries over a Web of Linked Data. In the second part of this dissertation we now focus on approaches to execute such queries. More precisely, in this chapter we provide a comprehensive, informal overview on techniques that might be used for developing a Linked Data query execution approach. The following chapters then focus on a particular query execution strategy (cf. Chapter 6) and a concrete approach to implement this strategy (cf. Chapter 7).

To discuss possible query execution techniques, we briefly recall the challenges of querying Linked Data on the WWW: Usually, queries are executed over a finite structure of data (e.g., a relational database or an RDF dataset) that is assumed to be fully available to the execution system. However, in this dissertation we focus on queries over a Web of Linked Data that might be infinite and that is—at best—partially known at the beginning of a query execution process. A query execution system might obtain data only by looking up URIs and parsing the documents retrieved by such a lookup. However, whether the lookup of a given URI actually results in the retrieval of a document is unknown beforehand. Furthermore, in a puristic implementation of a Web of Linked Data (as we assume in this dissertation), Web servers do not provide query processing interfaces. Hence, a query execution system cannot distribute the execution of (sub)queries to remote data sources. Instead, such a system has to retrieve data for local processing.

Multiple approaches to address these challenges have been proposed in the literature [67, 72, 79, 99, 100, 115, 126, 139, 156, 159, 162]. The basis of each of these approaches is a number of specific (and often complementary) query execution techniques. Some of the techniques presented for different approaches implement the same abstract idea; other techniques are conceptually different or serve different purposes.

The goal of this chapter is to provide a systematic overview of these Linked Data query execution techniques. To this end, we categorize these techniques along three orthogonal dimensions: (i) data source selection (cf. Section 5.1), (ii) data source ranking (cf. Section 5.2), and (iii) integration of data retrieval and result construction (cf. Section 5.3). For each of these dimensions, we provide a comprehensive conceptual comparison of the techniques in that dimension. Thereafter, we discuss so called traversal-based query execution strategies which combine particular types of techniques from each of the dimensions (cf. Section 5.4). These strategies are of particular interest because the query execution strategy analyzed in the following chapters presents a specific example of these traversal-based strategies. A classification consisting of this analyzed strategy and all existing Linked Data query execution approaches concludes this chapter (Section 5.5).

5.1. Data Source Selection

For the execution of Linked Data queries it is necessary to retrieve data by looking up URIs. There exist three classes of approaches for selecting the URIs that a query execution system looks up during the execution of a given query: (i) live exploration approaches, (ii) index-based approaches, and (iii) hybrid approaches. In the following we discuss each of these types.

5.1.1. Live Exploration Approaches

Live exploration approaches make use of the characteristics of Webs of Linked Data, in particular, the existence of data links. In order to execute a given Linked Data query, live exploration systems perform a recursive URI lookup process during which they incrementally discover further URIs that also qualify for lookup. Thus, such a system explores the queried Web by traversing data links at query execution time. While the data retrieved during such an exploration allows for a discovery of more URIs to look up, it also provides the basis for constructing the query result.

Live exploration systems may not need to look up all URIs discovered. Instead, certain live exploration approaches may (directly or indirectly) introduce criteria to decide which of the discovered URIs are scheduled for lookup. Such a lookup criterion may resemble a particular reachability criterion. In such a case it may be shown that the given live exploration approach is sound and complete for queries under the corresponding reachability-based query semantics. For instance, we shall see that the query execution strategy that we study in this dissertation supports c_{Match}-semantics.

We notice that query execution based on live exploration is similar to focused crawling as studied in the context of search engines for the WWW [31, 12]. However, in focused crawling a (discovered) URI qualifies for lookup because of a high relevance for a specific topic; in live exploration approaches the relevance is more closely related to the task of answering the query at hand. Furthermore, the purpose of retrieving Web content is slightly different in both cases: Focused crawling, or Web crawling in general, is a pre-runtime (or background) process during which a system populates a search index or a local database; then, the runtime component of such a system provides query access to the populated data structure. By contrast, live exploration approaches are used to retrieve data for answering a particular query; in these approaches, traversal-based data retrieval is an essential part of the query execution process itself. Nonetheless, implementation techniques used for focused crawling, such as context graphs [40], may be applied in a live exploration approach for Linked Data query execution.

The most important characteristic of live exploration approaches is the possibility to use data from initially unknown data sources. This characteristic allows for serendipitous discovery and, thus, enables applications that tap the full potential of a Web of Linked Data such as the WWW. Another characteristic is that live exploration approaches might be used to develop query execution systems that do not require any a-priori information about the queried Web. Consequently, such a system might readily be used without having to wait for the completion of an initial data load phase or any other type of preprocessing. Hence, live exploration approaches are most suitable for an "on-demand" querying scenario. However, data access times inherently add up due to the recursive nature of the lookup process. Possibilities for parallelizing data retrieval are limited because relevant URIs become available only incrementally. Furthermore, from Example 4.3 (cf. page 65) we know that the recursive link discovery may be infinite, even if the expected query result is finite. Another limitation of live exploration approaches is their inherent dependency on the structure of the link graph as well as on the number of links in the queried Web of Linked Data. In a Web sparsely populated with links, chances are low to discover relevant data. While such a limitation is not an issue for queries under a reachability-based query semantics, systems that aim to support full-Web semantics might report more complete results for certain queries if they use other source selection approaches.

In its purest form, live exploration approaches assume query execution systems that do not have any a-priori information about the queried Web. This assumption also holds for the approach that we study in this dissertation. It is also possible, however, that a query execution system reuses data retrieved during the execution of a query as a basis for executing subsequent queries. In [71] we demonstrate that such a reuse is beneficial for two reasons: 1) it can improve query performance because it reduces the need to retrieve data multiple times; 2) assuming full-Web semantics, it can provide for more complete query results, calculated based on data from data sources that would not be discovered by a live exploration with an initially empty query-local dataset. However, since reusing the query-local dataset for the execution of multiple queries is a form of data caching, it requires suitable caching strategies. In particular, any system that keeps previously retrieved data has to apply an appropriate invalidation strategy; otherwise it could lose the advantage of up-to-date query results. As an alternative to caching retrieved data it is also possible to keep only a summary of the data or certain statistics about it. Such information may then be used to guide the execution of later queries (as in the case of index-based source selection approaches which we discuss in the following).

5.1.2. Index-Based Approaches

Index-based approaches ignore the existence of data links during the query execution process. Instead, these approaches use a pre-populated index to determine a set of URIs for lookup during query execution time. Hence, in contrast to index structures that store the data itself (such as the original B-tree [13] or existing approaches for indexing RDF data [65, 123, 164]), the index-based approaches discussed here use data structures that index URIs as pointers to data; each of these URIs may appear multiple times in such an index because the data that can be retrieved using such a URI may be associated with multiple index keys.

A typical example for such a data structure uses triple patterns as index keys [99]. Given such a pattern, the corresponding index entry is a set of URIs such that looking up each of these URIs provides us with some data that contains a matching triple for the pattern. To enable data source ranking (discussed in the following Section 5.2), index entries may additionally encode the cardinality of matching triples for each indexed

URI [67, 99, 159]. Thus, such an index presents a summary of the data available from all indexed URIs.

Source selection using such an index is based on a notion of relevance: A URI is *relevant* for a given query if the data retrieved by looking up the URI contributes to the query result [99, 156]. However, the existence of a triple that matches a triple pattern from the query is not sufficient to make the corresponding URI relevant; only if such a matching triple can be used to construct a solution of the query result, the URI is relevant.

Given that data from irrelevant URIs is not required to compute a query result, avoiding the lookup of such URIs reduces the cost of query executions significantly [67, 159, 99, 126]. Consequently, the focus of research in this context is to identify a subset of all (indexed) URIs that contains all relevant URIs and as few irrelevant ones as possible. While simpler approaches consider any triple pattern of a given query separately [126], more sophisticated approaches achieve a higher reduction of irrelevant URIs by taking into account joins between triple patterns [67, 159, 99, 156].

We note that these index-based approaches are closer in spirit to traditional query processing techniques than live exploration approaches. Existing data summarization and indexing techniques may be adapted to develop an index-based approach for Linked Data query execution. For instance, Umbrich et al. adopt multidimensional histograms (originally proposed to estimate selectivity of multidimensional queries [120]) as a data summary for index-based Linked Data query execution [159]. Similarly, the QTree that Harth et al. use as a summary of Linked Data [67] is a combination of a histogram and an R-tree (the latter was originally proposed to index data about spatial objects [61]).

Further index structures for index-based Linked Data query execution are proposed in the literature: In contrast to the aforementioned approach of using triple patterns as index keys, Tian et al. extract frequently used combinations of triple patterns from a given query workload and use unique encodings of these combinations as index keys [156]. For a query workload that is similar to the workload used for building an index, the authors show that their approach can prune more irrelevant URIs than the baseline approach of using triple patterns as index keys. An *inverted URI index* is another, very simple index structure [159]. In this case the index keys are URIs, namely, the URIs mentioned in the data that can be retrieved by looking up the indexed URIs. In another approach the index keys are properties and classes from ontologies used for the data [126]. Umbrich et al. refer to this approach as *schema-level indexing* [159]. In their work the authors compare index-based approaches that use an inverted URI index, schema-level indexing, the aforementioned QTree, and a multidimensional histogram [159].

Existing work on index-based Linked Data query execution usually assumes that the set of URIs to be indexed is given. To build the index for such a set it is necessary to retrieve the data for any given URI. Instead of populating the index based on a given set of URIs it is also possible to build such an index using the output of a Web crawler for Linked Data. For a comprehensive discussion of crawling Linked Data we refer to Hogan et al. [88]. Alternatively, (partially) populated indexes may also be a by-product of executing queries using a live exploration approach. However, in all these cases an initial lookup of all indexed URIs is required.

After populating an initial version of an index it is necessary to maintain such an index. Maintenance may include adding additionally discovered URIs and keeping the index up to date. The latter is necessary because what data can be retrieved from indexed URIs might change over time. While Umbrich et al. address this topic briefly [159], no work exists that discusses index maintenance for index-based Linked Data query execution in detail. We also do not elaborate on this topic further because index-based approaches are not the focus of this dissertation. However, we point out that the topic is related to index maintenance in information retrieval (e.g., [30, 105, 107]), index maintenance for (traditional) Web search engines (e.g., [27, 108]), Web caching (e.g., [39, 116, 163]), maintenance of data(base) caches (e.g., [25, 37]), and view maintenance in databases and data warehouses (e.g., [59, 147, 168]).

The most important characteristic of index-based approaches is the ability to determine at the beginning of a query execution all URIs that need to be looked up. This ability enables query execution systems to parallelize data retrieval. Such a parallelization might reduce data retrieval time for executing a query. As a consequence, an efficiently implemented index-based system might answer a Linked Data query faster than a live exploration system (assuming both systems look up the same set of URIs during the execution).

On the other hand, a live exploration system is ready for use immediately, whereas an index-based system can be used only after initializing its index. Such an initialization may take a significant amount of time assuming that the system has to retrieve the data for all indexed URIs first. In the aforementioned publications only Paret et al. take the initial retrieval time into account for the evaluation of their approaches [126]. Unfortunately, the actual setup of Paret et al.'s experiments is not clear; in particular, missing information about response times of the dedicated Web servers used for the experiment and about the number of URIs looked up, prohibit drawing conclusions from the reported measurements. However, for systems that use crawling to populate their index, we may get an idea of the initial data retrieval time by looking into related work. In particular, in their work on a search engine for Linked Data, Hogan et al. report the following measurements [88]: For crawling 1,000 URIs (resp. 100,000 URIs) with 64 threads on a single machine they report an overall crawl time of about 9 minutes (resp. 360 minutes); in a distributed setting, 8 machines with 64 threads each, crawl 100,000 URIs in about 63 minutes.

Another advantage of index-based approaches claimed in the literature is the ability to report query results that are more complete when compared to live exploration approaches [67]. However, the authors' understanding of completeness remains unclear, because they do not provide a precise definition of query semantics for the Linked Data queries executed by their approach (the same holds for any of the aforementioned index-based approaches; i.e., [67, 99, 126, 156, 159]). However, if we assume full-Web semantics, it is indeed possible that an index-based approach computes some solutions of a query result which a live exploration approach cannot compute; this is the case if (some) data necessary for computing these solutions cannot be discovered by link traversal. On the other hand, a live exploration approach may discover URIs that are not indexed and the data retrieved by looking up these URIs may allow for the computation of some query solutions. In such a case the corresponding index-based execution cannot compute these solutions. Hence, a general statement about the superiority of an index-based approach over a live exploration approach (or vice versa) w.r.t. result completeness is not possible in the context of full-Web semantics.

Finally, we also emphasize that the aforementioned notion of relevance of URIs should not be carried over directly to live exploration approaches (or used in a comparison of both types of approaches). For a live exploration system the retrieval of data is not only necessary to obtain matching triples that contribute to the query result; instead, such data may also allow the system to discover (and, thus, traverse) data links, through which the system may eventually obtain additional matching triples.

5.1.3. Hybrid Approaches

Hybrid source selection approaches combine an index-based approach with a live exploration approach and, thus, aim to achieve the advantages of both approaches without inheriting their respective shortcomings. For instance, a hybrid approach may use an index to determine a suitable set of seed URIs as input for a subsequent live exploration process. This process may than feed back information for updating, for expanding, or for reorganizing the index. An alternative idea is a hybrid approach that uses an index only to prioritize discovered data links and, thus, to control a live exploration process.

To the best of our knowledge, the only query execution strategy that implements such a hybrid approach is the “mixed strategy” proposed by Ladwig and Tran [99]. This strategy is based on a ranked list of URIs that need to be looked up. To obtain an initial version of this list for a given query, the approach exploits an index. Additional URIs discovered during query execution are then integrated into the list.

5.2. Data Source Ranking

Instead of merely selecting URIs to look up, execution strategies for Linked Data queries may rank the resulting set of URIs such that the ranking represents a priority for looking up each (selected) URI. Such a data source ranking may allow a query execution system to (i) report solutions of a query result as early as possible, (ii) to reduce the time for computing the first k solutions, or (iii) to compute the maximum number of solutions in a given amount of time [99]. Although, the query execution strategy studied in this dissertation does not implement an approach for source ranking, we briefly summarize related work on source ranking for Linked Data query execution.

Harth et al. complement their (QTree) index-based source selection approach with source ranking [67]. Selected URIs are ranked using an estimate for the number of query solutions that the data of each URI contributes to. The basis for estimating these numbers are cardinalities recorded in the QTree entries for each (indexed) URI. Then, by accessing such a QTree, a query execution system might obtain an estimate of how many matching triples for a given triple pattern are available from each URI. Based on these triple-pattern-specific estimates, Harth et al.’s approach computes the estimates for the whole query recursively. For this computation, the authors take join cardinality estimations into account.

Ladwig and Tran introduce a ranking approach that includes multiple scores [99]:

1. *Triple frequency - inverse source frequency (TF-ISF).* TF-ISF is an adaptation of the well-known TF-IDF measure used in information retrieval [135]. Ladwig and Tran define TF-ISF for a pair of an indexed URI and a triple pattern (from the query). The computation of such a TF-ISF measure is based on the triple-pattern-specific cardinality of the corresponding URI (that is, how many matching triples for the pattern can be obtained by looking up the URI). Similar to Harth et al. [67], Ladwig and Tran's approach obtains these cardinalities from an index. However, due to a different index structure used by Ladwig and Tran, the obtained cardinalities are accurate (in Harth et al. they are estimates only, because the QTree is an approximate index structure [67]).

2. *(URI-specific) join cardinality estimates.* This score represents an estimation of the number of query solutions that can be computed using only the data from a given URI. While this score is similar to the estimates that Harth et al. use for ranking, it neglects joins based on data from multiple URIs. For the computation of such a (URI-specific) join cardinality estimate, Ladwig and Tran propose an approach that, again, uses the triple pattern cardinalities recorded in their index, as well as partial query solutions (generated using a random sample of data already retrieved during the query execution).

3. *In-link scores.* This score is calculated based on incoming data links (i.e., references to a given URI in the data available from other URIs). Interlinkage information about a set of URIs may be recorded in an additional data structure that complements the main index used for index-based source selection. However, Ladwig and Tran propose to obtain such information during query execution. As a result, the in-link score can also be used to rank URIs selected by live exploration approaches.

Ladwig and Tran's ranking approach aggregates these scores using a weighted summation [99]. This approach is suitable for hybrid source selection because it supports both, URIs selected from an index as well as URIs discovered by live exploration. However, the ranks for indexed URIs are more accurate because for those URIs all scores are available, whereas newly discovered URIs can only be ranked based on their in-link score.

Some of the input for calculating the aforementioned scores may be refined based on information that becomes available during the query execution process. For instance, the data used to obtain samples for join cardinality estimation grows; similarly, additional interlinkage information becomes available. As a consequence, Ladwig and Tran propose a periodic recalculation of scores and resulting ranks. Based on certain thresholds the authors study the trade-off between the benefits of a more accurate ranking that can be achieved by recalculating scores more frequently and the higher costs incurred by a more frequent recalculation [99].

The source ranking approaches as summarized above are designed to achieve the objectives mentioned in the beginning of this section. We note that similar objectives characterize the problem of top-k query processing where only the k top-ranked result elements need to be computed. Wagner et al. study top-k processing for Linked Data queries [162]. In particular, the authors propose a top-k approach that builds on the index-based source selection strategy. Although this approach is about ranking (intermediate) query solutions, it enforces an indirect ranking of the URIs to look up. The additional information that is necessary for ranking is, again, assumed to be recorded in the pre-populated index.

5.3. Integration of Data Retrieval and Result Construction

The actual process of executing a Linked Data query may consist of two separate phases: During the first phase a query execution system selects URIs and uses them to retrieve data from the queried Web (as discussed before); during a subsequent, second phase the system generates the query result using the data retrieved in the first phase. Instead of separating these two phases it is also possible to integrate the retrieval of data and the result construction process. Hereafter, we use the term *integrated execution approaches* to refer to Linked Data query execution approaches that apply the latter idea. Analogously, *separated execution approaches* separate data retrieval from result construction by two consecutive phases. We emphasize that separating or integrating data retrieval and result construction is a design decision that is orthogonal to what source selection approach (and source ranking approach) is used for developing a Linked Data query execution approach. In the following we discuss both types of approaches, separated execution and integrated execution.

5.3.1. Separated Execution Approaches

Due to the clear separation of data retrieval and result construction, separated approaches are straightforward to implement. In particular, index-based source selection lends itself naturally to such an implementation [67, 159, 126]. However, it is also easy to develop a separated execution approach based on live exploration. For instance, we may derive such an approach directly from the definitions for reachability-based query semantics as introduced in the previous chapter (cf. Section 4.1, page 61ff): During the first phase a query execution system retrieves data by traversing recursively all data links that qualify according to the reachability criterion specified in the query; during the second phase the system generates the query result. The 2P machine that we use for the proofs in Section 4.4.2 implements such an approach (cf. Algorithm 4.1, page 88).

The downside of separated execution approaches is that the query execution system can report first solutions only after it has completed the data retrieval phase. Looking up a large set of selected URIs or retrieving the complete set of reachable data may exceed the resources of an execution system or it may take a prohibitively long time; for queries that are not finitely computable by an LD machine, it is even possible that the data retrieval process does not terminate at all (this was the reason why we could not use a 2P machine for the second case in our proof of Theorem 4.3; cf. page 91). We note that the application of data source ranking may address these problems (for the price of missing some query solutions).

5.3.2. Integrated Execution Approaches

Integrated approaches may allow a query execution system to report first solutions for a (monotonic) query early, that is, before data retrieval has been completed. Furthermore, integrated approaches have the potential to require significantly less query-local memory than any separated execution approach; this holds in particular for integrated approaches that process retrieved data in a streaming manner and, thus, do not require to store all retrieved data until the end of a query execution process.

As for separated approaches it is possible to use any type of source selection as a basis for an integrated execution approach. A manifold of combinations are conceivable; in particular, live exploration may be combined with an integrated approach in a multitude of ways. We refer to query executions that present such a combination (i.e., live exploration with an integrated execution approach) as *traversal-based query executions.*

5.4. Traversal-Based Query Execution

For a simple example of a traversal-based query execution strategy we recall the ER machine that we use in our proof of Theorem 4.3 (cf. Definition 4.12, page 92). The query execution strategy of this machine combines the idea of an integrated execution with source selection by live exploration: The machine alternates between link traversal phases and result computation phases. Each link traversal phase consists of traversing all those relevant data links that the machine finds in the data retrieved during the previous link traversal phase. Hence, with each link traversal phase the machine expands its information about (the reachable subweb of) the queried Web of Linked Data. After each link traversal phase the machine generates a (potentially incomplete) query result; from such a result the machine reports those solutions that did not appear in the previously generated result. While this execution strategy is sufficient for proving Theorem 4.3, the frequent recomputation of partial query results is not efficient and, thus, the strategy may not be suitable in practice.

Schmedding proposes a traversal-based query execution strategy that addresses this problem [139]. The idea of this strategy is to adjust an intermediate (and potentially incomplete) query result after each link traversal phase. Schmedding's main contribution is an extension of the SPARQL algebra operators that makes the differences between query results computed on different input data explicit; based on the extended algebra the intermediate query result can be adjusted by using only the data retrieved during the directly preceding link traversal phase (instead of recomputing the intermediate query result from scratch as done by the ER machine).

An alternative approach to traversal-based query execution is the strategy that we study in the following chapters. This strategy is based on a result construction process that generates any single solution of a query result incrementally (as opposed to generating incrementally the query result as a whole). We shall see that this strategy achieves an even tighter integration of data retrieval and result construction than the two aforementioned approaches.

The idea of integrating the traversal of data links into the application logic has first been proposed by Berners-Lee et al. [16]. The authors outline an algorithm that traverses data links in order to obtain more data about the entities presented in the Tabulator Linked Data browser. Shinavier's functional scripting language Ripple is based on the same idea [146]: While Ripple programs operate on Linked Data, the automatic lookup of recursively discovered URIs is an integral feature of the language. Therefore, it is not necessary to add explicit URI lookup commands to such a program. Instead, during runtime the Ripple interpreter traverses data links and retrieves all Linked Data required for the execution incrementally. The earliest integration of link traversal into an execution of Linked Data queries was implemented in the Semantic Web Client Library [20].

The first research publication on Linked Data query execution describes the idea of traversal-based query execution and introduces an efficient implementation of this idea using a synchronized pipeline of iterators [79]. We follow up on this implementation approach in [72], where we propose a heuristics-based approach for query planning. These two publications provide the basis for Chapter 7 in this dissertation. In a more recent publication we introduce a general, implementation independent formalization of a traversal-based execution strategy [75]; this formalization is the basis for the query execution model that we present in the following chapter (cf. Section 6.3, page 115ff).

While our work on implementing traversal-based query execution focuses on iterators, other authors introduce alternative implementation approaches:

- Ladwig and Tran propose an implementation approach that uses symmetric hash join operators which are connected via an asynchronous, push-based pipeline [99]. In later work, the authors extend this approach and introduce the *symmetric index hash join* operator. This operator allows a query execution system to incorporate a query-local RDF data set into the query execution [100].

- Miranker et al. introduce another push-based implementation [115]. The authors implement traversal-based query execution using Forgy's Rete match algorithm (originally introduced in [51]).

Since traversal-based query execution approaches combine an integrated execution and live exploration, they inherit the advantages and limitations of these two strategies (as discussed in Sections 5.1.1 and 5.3.2). That is, like all live exploration systems, traversal-based query execution systems are able to make use of data from initially unknown data sources and can readily be used without first populating and maintaining supporting data structures. Furthermore, a traversal-based query execution system can be built to report first solutions early. On the downside, data retrieval may not be parallelized as effectively as is possible with index-based source selection; moreover, a sparsity of data links reduces the chances for discovering potentially relevant data and may thus result in missing a larger number of solutions for queries under full-Web semantics.

5.5. Summary

We conclude our discussion of query execution techniques for Linked Data queries by classifying existing approaches in Table 5.1. For the classification we use the three dimensions as introduced in this chapter.

Publication	**Source Selection**	**Source Ranking**	**Integr. Exec.**
Harth et al. [67, 159]	index-based	yes	no
Ladwig and Tran [99] (“bottom up”)*	live exploration	yes	yes
Ladwig and Tran [99] (“top down”)	index-based	yes	yes
Ladwig and Tran [99] (mixed strategy)*	hybrid	yes	yes
Ladwig and Tran [100]*	live exploration	no	yes
Miranker et al. [115]*	live exploration	no	yes
Paret et al. [126]	index-based	no	no
Schmedding [139]*	live exploration	no	yes
Tian et al. [156]	index-based	no	n/a
Umbrich et al. [159] (multidim. histograms)	index-based	yes	no
Umbrich et al. [159] (schema-level index)	index-based	no	no
Umbrich et al. [159] (inverted URI index)	index-based	no	no
Wagner et al. [162]	index-based	yes	yes
*our work**	live exploration	no	yes

Table 5.1.: Classification of existing work on Linked Data query execution along the dimensions of (i) data source selection, (ii) data source ranking, and (iii) integration of data retrieval and result construction. Approaches marked with an asterisk (*) are traversal-based query execution approaches.

6. A Traversal-Based Strategy

Traversal-based query execution (as described in Section 5.4, page 107f) is among the most prevalent approaches for which query execution techniques have been studied in existing work on Linked Data query processing (cf. Table 5.1, page 109). This research interest may perhaps be attributed to the fact that traversal-based query execution presents a novel query execution paradigm. The ability to discover data from unknown data sources is its most distinguishing advantage over traditional query execution approaches which assume a fixed set of potentially relevant data sources that is known beforehand. This ability poses new practical challenges to query planning and optimization, which are the focus of existing work. On the other hand, a systematic discussion of fundamental properties such as termination or soundness and completeness of proposed approaches is missing from existing work. Therefore, in this and the following chapter, we provide such a more fundamental analysis.

For this analysis we focus on a specific traversal-based query execution strategy that integrates tightly data retrieval and query result construction. In this chapter we introduce and discuss this strategy independent from particular implementation techniques. The following chapter then focuses on a specific implementation approach. By clearly distinguishing the general strategy from this implementation approach, we can study the general characteristics separate from implementation-specific peculiarities.

To allow us to focus on the essence of the strategy we restrict ourselves to a discussion of conjunctive queries. We emphasize, however, that the concepts introduced in this part of the dissertation can easily be extended to support more expressive queries (as long as such queries are monotonic).

This chapter is organized as follows: As a preliminary for a well-founded discussion, Section 6.1 defines a notion of conjunctive Linked Data queries. The remaining sections focus on our traversal-based strategy: Section 6.2 introduces the strategy informally using an example. Section 6.3 presents a query execution model that defines the strategy formally. Based on this model we analyze the strategy. In particular, Section 6.4 shows that the strategy allows for a sound and complete execution of any conjunctive Linked Data query under the (reachability-based) c_{Match}-semantics that has been introduced in Section 4.1 (cf. page 61ff). We conclude the chapter with a brief summary in Section 6.5.

6.1. Conjunctive Linked Data Queries

As mentioned above, we discuss our traversal-based execution strategy for conjunctive queries only. Such a focus on conjunctive queries allows us to simplify the presentation of the main idea without getting distracted by details necessary to support more expressive Linked Data queries. However, as a preliminary for discussing execution of conjunctive

Linked Data queries we need a precise definition of these queries. This section provides such a definition.

Basically, the notion of conjunctive queries that we focus on are $\text{SPARQL}_{\text{LD(R)}}$ queries that use SPARQL expressions consisting only of triple patterns and AND. Since the AND operator is associative and commutative [128, 140], the order of the triple patterns that occur in such a conjunctive expression is irrelevant w.r.t. query semantics. Therefore, we may use (finite) sets of triple patterns as a more compact representation of these expressions. Consequently, for the sake of a more concise formalism we define our notion of conjunctive queries based on sets of triple patterns (instead of using the corresponding, conjunctive SPARQL expressions). Such a finite set of triple patterns is commonly referred to as a *basic graph pattern (BGP)* [63, 127]; we adopt this term in the following.

For any BGP B we write $\text{vars}(B)$ to denote the (finite) set of all variables in triple patterns of B, i.e., $\text{vars}(B) := \{?v \in \text{vars}(tp) \,|\, tp \in B\}$. Furthermore, the application of a valuation μ to a BGP B, denoted by $\mu[B]$, is defined as follows: $\mu[B] := \{\mu[tp] \,|\, tp \in B\}$.

To define conjunctive Linked Data queries using BGPs, we first introduce the usual set semantics for BGPs as specified in Pérez et al. [127]: Let $B = \{tp_1, tp_2, \dots, tp_n\}$ be a BGP and let G be a (potentially infinite but countable) set of RDF triples. The *evaluation of B over G*, denoted by $[\![B]\!]_G$, is defined as follows:

$$[\![B]\!]_G := \{\mu \,|\, \mu \text{ is a valuation such that } \text{dom}(\mu) = \text{vars}(B) \text{ and } \mu[B] \subseteq G\}\,.$$

The following proposition shows that any conjunctive SPARQL expression is semantically equivalent to the corresponding BGP.

Proposition 6.1. *Let P be a SPARQL expression that consists only of triple patterns and* AND*; let $B = \{tp_1, tp_2, \dots, tp_n\}$ be the BGP that consists of all triple patterns in P. For any set of RDF triples G it holds that $[\![P]\!]_G = [\![B]\!]_G$.*

Proof. Proposition 6.1 follows trivially from the results of Pérez et al. [127] (see in particular [127, Proposition 3.13] and [127, Note 3.14]). ■

To use our definitions and results from the previous chapter as a foundation for a BGP-based formalism, we assume a mapping exp that maps any nonempty BGP $B = \{tp_1, tp_2, \dots, tp_n\}$ to a SPARQL expression of the form $((\dots(tp_1 \text{ AND } tp_2) \text{ AND } \dots) \text{ AND } tp_n)$ where $n = |B|$ and $tp_i \in B$ for all $i \in \{1, \dots, n\}$; each singleton BGP $B = \{tp\}$ is mapped to its triple pattern tp (a single triple pattern is also a SPARQL expression, cf. Section 3.2.1).

Based on mapping exp we overload some of the notation and terminology introduced in the previous chapter: For any reachability criterion c, RDF triple t, URI u, and nonempty BGP B, we write $c(t, u, B)$ to denote the result of $c(t, u, \text{exp}(B))$. Then, given a reachability criterion c and a nonempty BGP B, an LD document $d \in D$ is (c, B)-reachable from a set of URIs $S \subseteq \mathcal{U}$ in a Web of Linked Data $W = (D, data, adoc)$ if and only if d is $(c, \text{exp}(B))$-reachable from S in W; the (S, c, B)-reachable subweb of W is the same as the $(S, c, \text{exp}(B))$-reachable subweb of W.

We are now ready to introduce a BGP-based notion of conjunctive Linked Data queries (under reachability-based query semantics).

Definition 6.1 ($\mathbf{C}_{\mathsf{LD(R)}}$ query). Let $S \subseteq \mathcal{U}$ be a finite set of URIs; let c be a reachability criterion; and let B be a nonempty BGP. The *conjunctive Linked Data query ($C_{\mathsf{LD}(R)}$ query)* that uses B, S, and c, denoted by $\mathcal{Q}_c^{B,S}$, is a Linked Data query that, for any Web of Linked Data W, is defined by $\mathcal{Q}_c^{B,S}(W) := [\![B]\!]_{\mathsf{AllData}(R)}$ where R denotes the (S, c, B)-reachable subweb of W. □

From the definitions it follows trivially that any $\mathrm{C}_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{B,S}$ is semantically equivalent to the corresponding $\mathrm{SPARQL}_{\mathsf{LD(R)}}$ query that uses expression $\mathsf{exp}(B)$. Formally:

Proposition 6.2. *For any $C_{\mathsf{LD}(R)}$ query $\mathcal{Q}_c^{B,S}$ and any Web of Linked Data W it holds that $\mathcal{Q}_c^{B,S}(W) = \mathcal{Q}_c^{P,S}(W)$ where $\mathcal{Q}_c^{P,S}$ is the SPARQL$_{\mathsf{LD}(R)}$ query that uses SPARQL expression $P = \mathsf{exp}(B)$ (and the same S and c as $\mathcal{Q}_c^{B,S}$).*

Proof. Proposition 6.2 follows directly from Definition 6.1, Proposition 6.1, and Definition 4.4 (cf. page 63). ■

Based on the semantic equivalence shown in Proposition 6.2, our results for (conjunctive) $\mathrm{SPARQL}_{\mathsf{LD(R)}}$ queries carry over directly to $\mathrm{C}_{\mathsf{LD(R)}}$ queries:

Corollary 6.1. *Let $\mathcal{Q}_c^{B,S}$ be a $C_{\mathsf{LD}(R)}$ query whose set of URIs S is not empty.*

1. *$\mathcal{Q}_c^{B,S}$ is satisfiable and monotonic.*
2. *If reachability criterion c ensures finiteness, then $\mathcal{Q}_c^{B,S}$ is finitely computable by an LD machine.*
3. *If reachability criterion c does not ensure finiteness, then $\mathcal{Q}_c^{B,S}$ is either finitely computable by an LD machine or it is eventually computable by an LD machine.*

Proof. *Claim 1* follows trivially from: (i) Proposition 6.2, (ii) the relationships shown in Proposition 4.7 (cf. page 77), and (iii) the fact that conjunctive SPARQL expressions (with triple patterns and AND only) are satisfiable (cf. Proposition C.2, page 197) and monotonic (cf. Proposition C.6, page 202).

Claim 2 follows trivially from: (i) Proposition 6.2 and (ii) Proposition 4.12 (which shows that any $\mathrm{SPARQL}_{\mathsf{LD(R)}}$ query whose reachability criterion ensures finiteness is finitely computable by an LD machine; cf. page 91).

Claim 3 follows trivially from: (i) Proposition 6.2, (ii) the satisfiability and the monotonicity of $\mathcal{Q}_c^{B,S}$ (see the previously shown Claim 1), and (iii) the fact that any satisfiable, monotonic $\mathrm{SPARQL}_{\mathsf{LD(R)}}$ query whose reachability criterion does not ensure finiteness is either finitely computable or eventually computable by an LD machine (cf. Theorem 4.3, page 91). ■

After defining our notion of conjunctive Linked Data queries formally, we are now ready to discuss our traversal-based strategy for executing such queries.

6.2. Informal Description

We recall from Section 5.4 (cf. page 107f) that traversal-based query execution approaches combine a live exploration approach to source selection (as discussed in Section 5.1.1, page 100f) with the idea of integrating data retrieval into the query-local result construction process (discussed in Section 5.3.2, page 107). The specific strategy of traversal-based query execution that we aim to discuss in this dissertation proposes a particularly tight integration of data retrieval approach into a result construction process that generates solutions of query results incrementally. The following example outlines this strategy.

Example 6.1. Let $\mathcal{Q}^{B_{\mathsf{ex}},S_{\mathsf{ex}}}_{c_{\mathsf{Match}}}$ be a $\mathrm{C}_{\mathsf{LD(R)}}$ query under c_{Match}-semantics. Suppose the BGP of this query is $B_{\mathsf{ex}} = \{tp_{\mathsf{ex1}}, tp_{\mathsf{ex2}}, tp_{\mathsf{ex3}}, tp_{\mathsf{ex4}}\}$ with the following four triple patterns:

$$tp_{\mathsf{ex1}} = (?product, \mathsf{producedBy}, \mathsf{producer1}), \qquad tp_{\mathsf{ex2}} = (?product, \mathsf{name}, ?productName),$$
$$tp_{\mathsf{ex3}} = (?offer, \mathsf{offeredProduct}, ?product), \text{ and } \quad tp_{\mathsf{ex4}} = (?offer, \mathsf{offeredBy}, \mathsf{vendor1}).$$

The query asks for all products from producer 1 for which we find offers from vendor 1. Let the seed URIs of this query include the URIs of all entities mentioned in B_{ex}, that is, $S_{\mathsf{ex}} = \{\mathsf{producer1}, \mathsf{vendor1}\}$. In what follows we describe a traversal-based execution of this query over our example Web W_{ex} (cf. Example 2.1, page 18).

Traversal-based query execution usually starts with an empty, query-local dataset. We obtain some seed data by looking up the seed URIs mentioned in the query. For our example query with the seed URIs producer1 and vendor1, we retrieve two sets of RDF triples, $data_{\mathsf{ex}}(d_{\mathsf{Pr1}})$ and $data_{\mathsf{ex}}(d_{\mathsf{V1}})$. We add these triples to the local dataset.

Now, we begin an iterative process which is based on those RDF triples in the query-local dataset that are a matching triple for any of the triple patterns in our query. During each step of the process we use such a triple (i) to construct a valuation and (ii) to retrieve more data by looking up the URIs mentioned in the triple. Looking up these URIs presents a traversal of data links based on which we may augment the query-local dataset. Such an augmentation may allow us to discover more matching triples and, thus, to perform further steps. The valuation that we construct during such a step may either be based solely on the current matching triple and, thus, cover only the corresponding triple pattern; or the valuation extends a previously constructed valuation such that it covers multiple triple patterns from the query. Ultimately, we are interested in the valuations that cover all triple patterns (because these valuations are the solutions of the query result). The whole process continues until it reaches a fixed point. We note that such a fixed point may not exist for queries that are not finitely computable by an LD machine.

For triple pattern tp_{ex1} in our example query the local dataset contains matching triple $(\mathsf{product2}, \mathsf{producedBy}, \mathsf{producer1})$, originating from seed document d_{Pr1}. By using this triple, we construct a valuation $\mu_{\mathsf{ex1}} = \{?product \rightarrow \mathsf{product2}\}$ and look up the URIs mentioned in the triple. While a second lookup of URI producer1 is not necessary and we do not retrieve any data by looking up URI producedBy, the lookup of URI product2 allows us to augment the query-local dataset with $data_{\mathsf{ex}}(d_{\mathsf{p2}})$.

In the augmented dataset we now find a matching triple for triple pattern tp_{ex2}, that is, triple $(\mathsf{product2}, \mathsf{name}, \text{"Product 2"}) \in data_{\mathsf{ex}}(d_{\mathsf{p2}})$. Given this triple we can extend valuation μ_{ex1} by adding a binding for variable *?productName*. We obtain valuation $\mu_{\mathsf{ex2}} = \{?product \to \mathsf{product2}, ?productName \to \text{"Product 2"}\}$, which already covers the first two triple patterns of our query. Notice, constructing μ_{ex2} is only possible because we retrieved $data_{\mathsf{ex}}(d_{\mathsf{p2}})$. However, before we discovered the first matching triple and looked up the URI product2, we neither knew about the existence of LD document d_{p2} nor about the matching triple that allows us to construct μ_{ex2}. Hence, the traversal of data links enables us to answer queries based on data from initially unknown sources.

We proceed with our execution strategy as follows: The data from the other seed document, d_{V1}, contains a matching triple for triple pattern tp_{ex4}. Based on this triple we construct a new valuation $\mu_{\mathsf{ex3}} = \{?offer \to \mathsf{offer1.1}\}$ (μ_{ex3} covers triple pattern tp_{ex4}) and augment our local dataset with all RDF triples from $data(d_{\mathsf{off1.1}})$, which we retrieve by looking up URI offer1.1. After this augmentation we may eventually construct valuation $\mu_{\mathsf{ex4}} = \{?product \to \mathsf{product2}, ?productName \to \text{"Product 2"}, ?offer \to \mathsf{offer1.1}\}$, which covers the whole BGP B_{ex}. Hence, μ_{ex4} can be reported as a solution for $\mathcal{Q}^{B_{\mathsf{ex}},S_{\mathsf{ex}}}_{c_{\mathsf{Match}}}$ over W_{ex}. □

The example illustrates how our traversal-based query execution strategy integrates an incremental construction of query solutions with a live exploration approach to data retrieval. This strategy presents an even tighter integration of data retrieval and result construction than the two strategies outlined in the beginning of Section 5.4 (cf. page 107f). That is, instead of performing multiple result computation phases that always compute a whole query result (from scratch as in an ER machine or incrementally as proposed by Schmedding [139]), our traversal-based strategy executes a single result construction process only; this process computes the solutions of a query result by augmenting valuations incrementally. For such an augmentation the process uses matching triples from data retrieved via link traversal. At the same time, the process uses these matching triples for the next link traversal step. Hence, this process deeply intertwines result construction and (traversal-based) data retrieval.

We emphasize that the strategy presents a general approach rather than a concrete algorithm. Hence, the strategy can be implemented using different techniques. Possible implementation approaches may use a Rete network as proposed by Miranker et al. [115], or Ladwig and Tran's asynchronous pipeline of symmetric hash join operators [99]. We study an iterator-based implementation in the following chapter. Further implementation approaches are conceivable. As a general foundation for such implementation approaches we now provide a formal definition of our traversal-based execution strategy.

6.3. Query Execution Model

This section defines a query execution model that captures the traversal-based execution strategy outlined in the previous section. This model does not only present a precise, unambiguous definition of the strategy, it also allows us to analyze the strategy formally (and independent of implementation-specific peculiarities).

Before we define our model we give an informal overview thereof.

6.3.1. Overview

The query execution strategy that we aim to capture with our execution model constructs solutions for a query incrementally. We call the intermediate results of such a construction *partial solutions* (a formal definition follows shortly). In addition to the construction of solutions, the strategy discovers the queried Web of Linked Data in an incremental manner. Therefore, we model query execution by a sequence of states, where each *state* is characterized by (i) the set of the partial solutions that have been constructed already and (ii) the currently *discovered subweb* of the queried Web of Linked Data.

In addition to the basic structural elements (partial solutions, discovered subwebs, and states) our model introduces operations over these elements. These operations include an *augmentation* operation for partial solutions and an *expansion* operation for the discovered subweb. Both of these operations are based on a matching triple. Since our traversal-based execution approach always uses such a matching triple for performing an augmentation and the corresponding expansion at the same time, we also introduce a combination of these operations. By performing such a *combined operation* an execution in our model enters its next state.

In the following we first focus on partial solutions and the corresponding augmentation operation. Second, we define our notion of discovered subwebs of a queried Web of Linked Data and formalize how query execution expands such a discovered subweb. After discussing augmentation and expansion separately, we introduce the concepts necessary to intertwine the incremental construction of (partial) solutions with the incremental expansion of the discovered subweb. Finally, we combine these concepts into an abstract procedure which represents the query execution that is possible in our model.

6.3.2. Partial Solutions

We now define the concept of partial solutions. This concept shall allow us to represent all valuations computed during query execution. However, as outlined in Example 6.1 (cf. page 114), some of these valuations cover only a part of the queries that we want to answer. In our definition of partial solutions we make the part covered explicit:

Definition 6.2 (Partial Solution). Let $\mathcal{Q}_c^{B,S}$ be a $\mathrm{C}_{\mathsf{LD(R)}}$ query; let W be a Web of Linked Data; let R be the (S,c,B)-reachable subweb of W. A *partial solution* for $\mathcal{Q}_c^{B,S}$ in W is a pair $\sigma = (E,\mu)$ with $E \subseteq B$ and $\mu \in [\![E]\!]_{\mathsf{AllData}(R)}$. To denote the set of all partial solutions for $\mathrm{C}_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{B,S}$ in Web of Linked Data W we write $\boldsymbol{\sigma}(\mathcal{Q}_c^{B,S}, W)$. □

Example 6.2. During the query execution in Example 6.1 (cf. page 114) we construct valuation μ_{ex1} using RDF triple $(\mathsf{product2}, \mathsf{producedBy}, \mathsf{producer1}) \in data_{\mathsf{ex}}(d_{\mathsf{Pr1}})$, which is a matching triple for triple pattern tp_{ex1} (the first triple pattern of our example query). Thus, the corresponding partial solution is the pair $\sigma_{\mathsf{ex1}} = (\{tp_{\mathsf{ex1}}\}, \mu_{\mathsf{ex1}})$. Partial solutions that correspond to the other valuations mentioned in Example 6.1 are:

$$\sigma_{\mathsf{ex2}} = (\{tp_{\mathsf{ex1}}, tp_{\mathsf{ex2}}\}, \mu_{\mathsf{ex2}}) \qquad \sigma_{\mathsf{ex3}} = (\{tp_{\mathsf{ex4}}\}, \mu_{\mathsf{ex3}})$$
$$\sigma_{\mathsf{ex4}} = (\{tp_{\mathsf{ex1}}, tp_{\mathsf{ex2}}, tp_{\mathsf{ex4}}\}, \mu_{\mathsf{ex4}}) \qquad \sigma_{\mathsf{ex5}} = (\{tp_{\mathsf{ex1}}, tp_{\mathsf{ex2}}, tp_{\mathsf{ex3}}, tp_{\mathsf{ex4}}\}, \mu_{\mathsf{ex4}})\,.$$ □

Given a partial solution $\sigma = (E, \mu)$, we say that this partial solution *covers* the triple patterns in E. For partial solutions of a query $\mathcal{Q}_c^{B,S}$ that cover the BGP B completely, it is trivial to show that the valuations in these partial solutions are solutions for the query:

Proposition 6.3. *Let $\sigma = (E, \mu)$ be a partial solution for a $\mathrm{C}_{\mathsf{LD}(R)}$ query $\mathcal{Q}_c^{B,S}$ in a Web of Linked Data W. If $E = B$, then $\mu \in \mathcal{Q}_c^{B,S}(W)$.*

Proof. Proposition 6.3 follows trivially from Definitions 6.2 and 6.1 (cf. page 113). ■

A special partial solution that exists for any $\mathrm{C}_{\mathsf{LD(R)}}$ query (in any Web of Linked Data) is the *empty partial solution* $\sigma_\emptyset := (B_\emptyset, \mu_\emptyset)$. This partial solution covers the empty BGP $B_\emptyset := \emptyset$ (recall, $\mu_\emptyset$ denotes the empty valuation for which $\mathrm{dom}(\mu_\emptyset) = \emptyset$ holds). Every query execution in our execution model starts with the empty partial solution.

6.3.3. Constructing (Partial) Solutions

During query execution our model uses matching triples to extend valuations incrementally such that the resulting, extended valuations cover larger parts of the corresponding BGP. We define such an extension as an operation over partial solutions:

Definition 6.3 (Augmentation). Let $\sigma = (E, \mu)$ be a partial solution for a $\mathrm{C}_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{B,S}$ in a Web of Linked Data W. Given a triple pattern $tp \in B \setminus E$ and an RDF triple t such that t is a matching triple for triple pattern $tp' = \mu[tp]$, the *(t, tp)-augmentation of σ*, denoted by $\mathsf{AUG}(\sigma, t, tp)$, is a pair (E', μ') where $E' := E \cup \{tp\}$ and μ' is a valuation that extends μ as follows: (i) $\mathrm{dom}(\mu') := \mathrm{vars}(E')$ and (ii) $\mu'[E'] := \mu[E] \cup \{t\}$. □

Example 6.3. Recall that RDF triple $(\mathsf{product2}, \mathsf{producedBy}, \mathsf{producer1})$ is a matching triple for triple pattern $tp_{\mathsf{ex1}} = (?product, \mathsf{producedBy}, \mathsf{producer1})$ of the query in Example 6.1 (cf. page 114). Let us write t_{ex1} to denote this triple, then the $(t_{\mathsf{ex1}}, tp_{\mathsf{ex1}})$-augmentation of the empty partial solution $\sigma_\emptyset$ is $\mathsf{AUG}(\sigma_\emptyset, t_{\mathsf{ex1}}, tp_{\mathsf{ex1}}) = (\{tp_{\mathsf{ex1}}\}, \mu_{\mathsf{ex1}})$ where $\mu_{\mathsf{ex1}} = \{?product \rightarrow \mathsf{product2}\}$. We emphasize that this augmentation of $\sigma_\emptyset$ is the partial solution σ_{ex1} that we introduce in Example 6.2. Similarly, partial solution σ_{ex2} in Example 6.2 is the result of $\mathsf{AUG}(\sigma_{\mathsf{ex1}}, t_{\mathsf{ex2}}, tp_{\mathsf{ex2}})$ if we assume t_{ex2} is the second matching triple that we use during the example execution (i.e., the triple that matches triple pattern tp_{ex2}). □

The following proposition shows that, if the matching triple used for augmenting a partial solution is available in the reachable subweb of the queried Web, then the result of the augmentation is again a partial solution (as in Example 6.3).

Proposition 6.4. *Let $\sigma = (E, \mu)$ be a partial solution for a $\mathrm{C}_{\mathsf{LD}(R)}$ query $\mathcal{Q}_c^{B,S}$ in a Web of Linked Data W; let R denote the (S, c, B)-reachable subweb of W; let $tp \in B \setminus E$; and let t be a matching triple for triple pattern $tp' = \mu[tp]$. If $t \in \mathsf{AllData}(R)$, then the (t, tp)-augmentation of σ is a partial solution for $\mathcal{Q}_c^{B,S}$ in W.*

Proof. Suppose $t \in \mathsf{AllData}(R)$. To show that $(E', \mu') = \mathsf{AUG}(\sigma, t, tp)$ is a partial solution for $\mathcal{Q}_c^{B,S}$ in W, we have to show: (1) $E' \subseteq B$ and (2) $\mu' \in [\![E']\!]_{\mathsf{AllData}(R)}$ (cf. Definition 6.2, page 116).

(1) holds because (i) $E' = E \cup \{tp\}$ (cf. Definition 6.3, page 117), (ii) $tp \in B \setminus E$, and (iii) $E \subseteq B$ (because σ is a partial solution for $\mathcal{Q}_c^{B,S}$ in W).

To show (2) we distinguish two cases: $E = \emptyset$ and $E \neq \emptyset$. We note, in the former case, partial solution σ is the empty partial solution $\sigma_\emptyset$, whereas, in the latter case, $\sigma \neq \sigma_\emptyset$.

We first discuss the former case. In this case, $E' = \{tp\}$ holds. From Definition 6.3 we know $\mathrm{dom}(\mu') = \mathrm{vars}(E')$ and $\mu'[E'] = \{t\}$ (to see the latter, note that $\mu[E] = \emptyset$ for $E = \emptyset$). From $t \in \mathsf{AllData}(R)$ we have $\mu'[E'] \subseteq \mathsf{AllData}(R)$ and, thus, $\mu' \in [\![E']\!]_{\mathsf{AllData}(R)}$.

We now discuss the second case, $E \neq \emptyset$. In this case, $\mu' \in [\![E']\!]_{\mathsf{AllData}(R)}$ follows trivially from the definitions and the fact that $t \in \mathsf{AllData}(R)$. ■

6.3.4. Discovered Subwebs of the Queried Web

In addition to a set of partial solutions that have already been constructed, any point in a query execution process is characterized by the information that has already been discovered about the queried Web of Linked Data.

Example 6.4. At the begin of the query execution in Example 6.1 we look up the seed URIs producer1 and vendor1. As a result we obtain partial knowledge of the queried example Web $W_{\mathsf{ex}} = (D_{\mathsf{ex}}, data_{\mathsf{ex}}, adoc_{\mathsf{ex}})$. More precisely, we learn that $adoc_{\mathsf{ex}}(\mathsf{producer1}) = d_{\mathsf{Pr1}}$ and $adoc_{\mathsf{ex}}(\mathsf{vendor1}) = d_{\mathsf{V1}}$, and, thus, $\{d_{\mathsf{Pr1}}, d_{\mathsf{V1}}\} \subseteq D_{\mathsf{ex}}$; we also discover the RDF triples in $data_{\mathsf{ex}}(d_{\mathsf{Pr1}})$ and in $data_{\mathsf{ex}}(d_{\mathsf{V1}})$. □

The information that is available about a queried Web of Linked Data (at any particular point during query execution) determines the set of all possible next steps for the execution. Therefore, our execution model captures this information formally:

Definition 6.4 (Discovered Subweb). Let $\mathcal{T} = (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ be the infinite set of all possible RDF triples and let $\perp$ denote the nonexistent LD document (cf. Definition 2.1, page 16). A *discovered subweb* of a Web of Linked Data $W = (D, data, adoc)$ is a tuple $W_{\mathfrak{D}} = (D_{\mathfrak{D}}, data_{\mathfrak{D}}, adoc_{\mathfrak{D}})$ with the following three elements:

1. $D_{\mathfrak{D}}$ is a *finite* subset of D; i.e., $D_{\mathfrak{D}} \subseteq D$.
2. $data_{\mathfrak{D}}$ is a total mapping $data_{\mathfrak{D}} : D_{\mathfrak{D}} \to 2^{\mathcal{T}}$ such that $data_{\mathfrak{D}}(d) = data(d)$ for all LD documents $d \in D_{\mathfrak{D}}$.
3. $adoc_{\mathfrak{D}}$ is a *partial* mapping $adoc_{\mathfrak{D}} : \mathcal{U} \to D_{\mathfrak{D}} \cup \{\perp\}$ such that $adoc_{\mathfrak{D}}$ satisfies the following conditions for any URI $u \in \mathcal{U}$:
   a) If $adoc(u) \in D_{\mathfrak{D}}$, then $adoc_{\mathfrak{D}}(u) = adoc(u)$ or $u \notin \mathrm{dom}(adoc_{\mathfrak{D}})$.
   b) If $adoc(u) \in D \setminus D_{\mathfrak{D}}$, then $u \notin \mathrm{dom}(adoc_{\mathfrak{D}})$.
   c) If $adoc(u) = \perp$, then $adoc_{\mathfrak{D}}(u) = \perp$ or $u \notin \mathrm{dom}(adoc_{\mathfrak{D}})$. □

Definition 6.4 assumes finiteness for the set of LD documents in any discovered subweb of a Web of Linked Data. This assumption captures the fact that we obtain information about a queried Web of Linked Data incrementally; thus, at any point in a query execution process we only know a finite part of such a Web, even if this Web is infinite. Furthermore, we emphasize that mapping $adoc$ of any Web of Linked Data $W = (D, data, adoc)$ is total (cf. Definition 2.1, page 16), whereas the corresponding mapping $adoc_{\mathfrak{D}}$ of a discovered subweb $W_{\mathfrak{D}} = (D_{\mathfrak{D}}, data_{\mathfrak{D}}, adoc_{\mathfrak{D}})$ is only partial. This relaxation is necessary because up to any point in a query execution process we can only look up a particular (finite) subset of all URIs. Due to this relaxation, discovered subwebs of Webs of Linked Data are not necessarily Webs of Linked Data themselves and, thus, strictly speaking, they are not subwebs as per Definition 2.3 (cf. page 18).

To denote the set of *all RDF triples* in a discovered subweb we overload function $\mathsf{AllData}$. That is, for any discovered subweb $W_{\mathfrak{D}} = (D_{\mathfrak{D}}, data_{\mathfrak{D}}, adoc_{\mathfrak{D}})$ (of some Web of Linked Data), we define:

$$\mathsf{AllData}(W_{\mathfrak{D}}) := \bigcup_{d \in D_{\mathfrak{D}}} data_{\mathfrak{D}}(d)\,.$$

Remark 6.1. For a discovered subweb $W_{\mathfrak{D}} = (D_{\mathfrak{D}}, data_{\mathfrak{D}}, adoc_{\mathfrak{D}})$ of a Web of Linked Data $W = (D, data, adoc)$ it holds that mapping $data_{\mathfrak{D}}$ depends only on set $D_{\mathfrak{D}}$ and mapping $data$. Therefore, we may identify any discovered subweb $W_{\mathfrak{D}} = (D_{\mathfrak{D}}, data_{\mathfrak{D}}, adoc_{\mathfrak{D}})$ of a given Web of Linked Data $W = (D, data, adoc)$, by specifying only $D_{\mathfrak{D}}$ and $adoc_{\mathfrak{D}}$.

Query execution in our model starts by looking up the seed URIs given in the query. The result is an *initially discovered subweb* which contains information about all seed URIs and about all LD documents that can be retrieved using the seed URIs.

Example 6.5. Let $(D_{\mathsf{ex0}}, data_{\mathsf{ex0}}, adoc_{\mathsf{ex0}})$ denote the initially discovered subweb of our example Web W_{ex} after looking up seed URIs $S_{\mathsf{ex}} = \{\mathsf{producer1}, \mathsf{vendor1}\}$ at the begin of our example execution (cf. Example 6.1, page 114). Since $\mathsf{producer1}$ and $\mathsf{vendor1}$ are the only URIs looked up at this point, mapping $adoc_{\mathsf{ex0}}$ is defined only for these two URIs; i.e., $\mathrm{dom}(adoc_{\mathsf{ex0}}) = \{\mathsf{producer1}, \mathsf{vendor1}\}$ with $adoc_{\mathsf{ex0}}(\mathsf{producer1}) = d_{\mathsf{Pr1}}$ and $adoc_{\mathsf{ex0}}(\mathsf{vendor1}) = d_{\mathsf{V1}}$. Furthermore, we have $D_{\mathsf{ex0}} = \{d_{\mathsf{Pr1}}, d_{\mathsf{V1}}\}$ and, thus, $\mathrm{dom}(data_{\mathsf{ex0}}) = \{d_{\mathsf{Pr1}}, d_{\mathsf{V1}}\}$ with $data_{\mathsf{ex0}}(d_{\mathsf{Pr1}}) = data_{\mathsf{ex}}(d_{\mathsf{Pr1}})$ and $data_{\mathsf{ex0}}(d_{\mathsf{V1}}) = data_{\mathsf{ex}}(d_{\mathsf{V1}})$. □

In general, we define the initially discovered subweb of a queried Web as follows:

Definition 6.5 (Seed Subweb). Let $W = (D, data, adoc)$ be a Web of Linked Data and let $S \subseteq \mathcal{U}$ be a finite set of URIs. The S*-seed subweb* of W, denoted by $\mathfrak{D}_{\mathsf{init}(S,W)}$, is the discovered subweb of W that is defined by $\mathfrak{D}_{\mathsf{init}(S,W)} := (D_0, data_0, adoc_0)$ such that:

1. $D_0 = \{adoc(u) \in D \,|\, u \in S \text{ and } adoc(u) \neq \bot\}$, and
2. $\mathrm{dom}(adoc_0) = S$ and $adoc_0(u) = adoc(u)$ for all $u \in \mathrm{dom}(adoc_0)$. □

6.3.5. Traversing Data Links

During traversal-based query execution we traverse data links to retrieve additional RDF triples. These triples may allow us to compute partial solutions and to discover further data links. In the query execution strategy that we model we traverse data links by looking up the URIs mentioned in each matching triple that we use for generating partial solutions (cf. Example 6.1, page 114).

In terms of our execution model the lookup of URIs from a matching triple is an operation that expands the discovered subweb of the queried Web of Linked Data. Formally:

Definition 6.6 (Expansion). Let $W_{\mathfrak{D}} = (D_{\mathfrak{D}}, data_{\mathfrak{D}}, adoc_{\mathfrak{D}})$ be a discovered subweb of a Web of Linked Data $W = (D, data, adoc)$. Given an RDF triple t, the *t-expansion* of $W_{\mathfrak{D}}$ in W, denoted by $\mathsf{EXP}(W_{\mathfrak{D}}, t, W)$, is a tuple $(D'_{\mathfrak{D}}, data'_{\mathfrak{D}}, adoc'_{\mathfrak{D}})$ whose three elements are defined as follows:

1. $D'_{\mathfrak{D}} := D_{\mathfrak{D}} \cup \Delta^W(t)$ where $\Delta^W(t) := \{ adoc(u) \in D \,|\, u \in \mathrm{uris}(t) \text{ and } adoc(u) \neq \bot \}$.
2. $data'_{\mathfrak{D}}$ is a total mapping $data'_{\mathfrak{D}} \colon D'_{\mathfrak{D}} \rightarrow 2^{\mathcal{T}}$ such that $data'_{\mathfrak{D}}(d) := data(d)$ for all LD documents $d \in D'_{\mathfrak{D}}$.
3. $adoc'_{\mathfrak{D}}$ is a partial mapping $adoc'_{\mathfrak{D}} \colon \mathcal{U} \rightarrow D'_{\mathfrak{D}} \cup \{\bot\}$ such that

$$\mathrm{dom}(adoc'_{\mathfrak{D}}) := \mathrm{dom}(adoc_{\mathfrak{D}}) \cup \mathrm{uris}(t)$$

and, for each URI $u \in \mathrm{dom}(adoc'_{\mathfrak{D}})$,

$$adoc'_{\mathfrak{D}}(u) := \begin{cases} adoc(u) & \text{if } u \in \mathrm{uris}(t), \\ adoc_{\mathfrak{D}}(u) & \text{else.} \end{cases}$$ □

Although expansion operations use the queried Web of Linked Data W as an input, it is important to note that Definition 6.6 accounts for the limited data access capabilities in an implementation of a Web of Linked Data such as the WWW. That is, query execution systems can perform expansion operations without having complete knowledge of W: By looking up all URIs $u \in \mathrm{uris}(t)$, such a system can obtain all information necessary to generate the t-expansion of a given discovered subweb. Similarly, in more abstract terms, an LD machine can obtain this information by performing its expand procedure for all URIs in $\mathrm{uris}(t)$.

Example 6.6. Recall that the first matching triple that we consider during our example execution is $t_{\mathsf{ex1}} = (\mathsf{product2}, \mathsf{producedBy}, \mathsf{producer1})$ (cf. Example 6.1 on page 114). By looking up the URIs from t_{ex1} in the queried example Web W_{ex}, we perform the t_{ex1}-expansion of $\mathfrak{D}_{\mathsf{init}(S_{\mathsf{ex}}, W_{\mathsf{ex}})}$ in W_{ex}, where $\mathfrak{D}_{\mathsf{init}(S_{\mathsf{ex}}, W_{\mathsf{ex}})} = (D_{\mathsf{ex0}}, data_{\mathsf{ex0}}, adoc_{\mathsf{ex0}})$ is the (initially discovered) S_{ex}-seed subweb of W_{ex} (cf. Example 6.5, page 119). As a result we have $\mathsf{EXP}(\mathfrak{D}_{\mathsf{init}(S_{\mathsf{ex}}, W_{\mathsf{ex}})}, t_{\mathsf{ex1}}, W_{\mathsf{ex}}) = (D_{\mathsf{ex1}}, data_{\mathsf{ex1}}, adoc_{\mathsf{ex1}})$ where $D_{\mathsf{ex1}} = D_{\mathsf{ex0}} \cup \{d_{\mathsf{p2}}\}$ and $\mathrm{dom}(adoc_{\mathsf{ex1}}) = \mathrm{dom}(adoc_{\mathsf{ex0}}) \cup \{\mathsf{product2}, \mathsf{producedBy}\}$ with $adoc_{\mathsf{ex1}}(\mathsf{product2}) = d_{\mathsf{p2}}$ and $adoc_{\mathsf{ex1}}(\mathsf{producedBy}) = \bot$. □

The following proposition shows that the set of all possible discovered subwebs of a Web of Linked Data is closed under the expansion operation.

Proposition 6.5. *Let $W_{\mathfrak{D}}$ be a discovered subweb of a Web of Linked Data W and let t be an arbitrary RDF triple, then $\mathsf{EXP}(W_{\mathfrak{D}}, t, W)$ is also a discovered subweb of W.*

Proof. Let $W = (D, data, adoc)$, $W_{\mathfrak{D}} = (D_{\mathfrak{D}}, data_{\mathfrak{D}}, adoc_{\mathfrak{D}})$, and $\mathsf{EXP}(W_{\mathfrak{D}}, t, W) = (D'_{\mathfrak{D}}, data'_{\mathfrak{D}}, adoc'_{\mathfrak{D}})$. To prove that $\mathsf{EXP}(W_{\mathfrak{D}}, t, W)$ is a discovered subweb of W we have to show that $(D'_{\mathfrak{D}}, data'_{\mathfrak{D}}, adoc'_{\mathfrak{D}})$ is a tuple that has the three properties given in Definition 6.4 (cf. page 118). Property 2, however, holds by definition.

Property 1 requires that $D'_{\mathfrak{D}} = D_{\mathfrak{D}} \cup \Delta^W(t)$ is a finite subset of D. Since $W_{\mathfrak{D}}$ is a discovered subweb of W, $D_{\mathfrak{D}}$ is a finite subset of D (cf. Definition 6.4). Hence, we only have to show (i) $\Delta^W(t) \subseteq D$ and (ii) $\Delta^W(t)$ is finite (cf. Definition 6.6). The former holds by definition and the latter follows from the finiteness of $\mathrm{uris}(t)$.

W.r.t. Property 3 we first note that $adoc'_{\mathfrak{D}}$ is a partial mapping $adoc'_{\mathfrak{D}} : \mathcal{U} \to D'_{\mathfrak{D}} \cup \{\bot\}$. Hence, we have to show that $adoc'_{\mathfrak{D}}$ satisfies the three requirements of Property 3.

a) Let $u \in \mathcal{U}$ be a URI such that $adoc(u) \in D'_{\mathfrak{D}}$. If $u \in \mathrm{dom}(adoc'_{\mathfrak{D}})$, then $adoc'_{\mathfrak{D}}(u) = adoc(u)$ holds by Definition 6.6 (recall, $W_{\mathfrak{D}}$ is a discovered subweb of W).

b) Let $u \in \mathcal{U}$ be a URI such that $adoc(u) \in D \setminus D'_{\mathfrak{D}}$, and show $u \notin \mathrm{dom}(adoc'_{\mathfrak{D}})$. Since $\mathrm{dom}(adoc'_{\mathfrak{D}}) = \mathrm{dom}(adoc_{\mathfrak{D}}) \cup \mathrm{uris}(t)$ (cf. Definition 6.6) we have to show $u \notin \mathrm{dom}(adoc_{\mathfrak{D}})$ and $u \notin \mathrm{uris}(t)$. Since $adoc(u) \notin D'_{\mathfrak{D}}$ and $D'_{\mathfrak{D}} = D_{\mathfrak{D}} \cup \Delta^W(t)$ we have $adoc(u) \notin D_{\mathfrak{D}}$ and $adoc(u) \notin \Delta^W(t)$. From $adoc(u) \notin D_{\mathfrak{D}}$ and the fact that $W_{\mathfrak{D}}$ is a discovered subweb of W we have $u \notin \mathrm{dom}(D_{\mathfrak{D}})$. Similarly, from $adoc(u) \notin \Delta^W(t)$ and the fact that $adoc(u) \in D$ (and, thus, $adoc(u) \neq \bot$) we have $u \notin \mathrm{uris}(\mu)$.

c) Let $u \in \mathcal{U}$ be a URI such that $adoc(u) = \bot$. If $u \in \mathrm{dom}(adoc'_{\mathfrak{D}})$, then $adoc'_{\mathfrak{D}}(u) = \bot$ holds by Definition 6.6 and the fact that $W_{\mathfrak{D}}$ is a discovered subweb of W. ■

6.3.6. Combining Construction and Traversal

Expanding the discovered subweb and augmenting partial solutions may be understood as separate processes. However, the idea of the execution strategy that we model is to intertwine these two processes. More precisely, the strategy combines each augmentation of a partial solution with an expansion operation that uses the same matching triple as used for the augmentation. To capture this idea in our execution model we formalize query execution as a sequence of states such the transition from a state to a subsequent state is the combined performance of an augmentation operation and a corresponding expansion operation. We note that each state of a such a query execution is characterized sufficiently by specifying (i) the set of partial solutions that have already been constructed and (ii) the currently discovered subweb of the queried Web. Consequently, we define a *query execution state*, or *QE state* for short, as follows:

Definition 6.7 (QE State). Let $\mathcal{Q}^{B,S}_c$ be a $\mathrm{C}_{\mathsf{LD(R)}}$ query; let W be a Web of Linked Data; and let R denote the (S, c, B)-reachable subweb of W. A *QE state* for $\mathcal{Q}^{B,S}_c$ over W is a pair $st = (\sigma, R_{\mathfrak{D}})$ where:

1. $\sigma \subseteq \sigma(\mathcal{Q}^{B,S}_c, W)$ is a finite set of partial solutions for $\mathcal{Q}^{B,S}_c$ in W, and
2. $R_{\mathfrak{D}}$ is a discovered subweb of R. □

Instead of simply prescribing that $R_{\mathfrak{D}}$ of any QE state $st = (\mathcal{O}, R_{\mathfrak{D}})$ is a discovered subweb of the queried Web of Linked Data, Definition 6.7 requires that $R_{\mathfrak{D}}$ must be contained in the corresponding reachable subweb (of the queried Web). This constraint is necessary to ensure the soundness of our execution model: Recall, Proposition 6.4 guarantees that the augmentation of partial solutions is sound if the matching triple that we use for the augmentation is contained in the corresponding *reachable* subweb of the Web (cf. page 117).

We now focus on possible transitions from a QE state $st = (\mathcal{O}, R_{\mathfrak{D}})$ to a subsequent QE state. As mentioned before, such a transition presents a combined performance of augmenting a partial solution $\sigma \in \mathcal{O}$ and expanding the discovered subweb $R_{\mathfrak{D}}$ (using the same matching triple for both operations). To capture such a combination formally, we introduce the concept of an *augmentation & expansion task (AE task)* and define the operation of performing such a task as an operation over QE states.

We characterize AE tasks by the elements that are necessary for a combined performance of an augmentation and the corresponding expansion:

Definition 6.8 (AE Task). Let $\mathcal{Q}_c^{B,S}$ be a $\mathrm{C}_{\mathsf{LD(R)}}$ query; let W be a Web of Linked Data; let R be the (S,c,B)-reachable subweb of W. A tuple $(\sigma, t, tp) \in \mathcal{O}(\mathcal{Q}_c^{B,S}, W) \times \mathsf{AllData}(R) \times B$, where $\sigma = (E, \mu)$, is an *AE task* for $\mathcal{Q}_c^{B,S}$ over W if the following two properties hold:

1. Triple pattern tp is not covered by partial solution σ; i.e., $tp \notin E$.

2. RDF triple t is a matching triple for triple pattern $tp' = \mu[tp]$. □

Example 6.7. The first AE task of our example query execution in Example 6.1 (cf. page 114) is $\tau_{\mathsf{ex1}} = (\sigma_\emptyset, t_{\mathsf{ex1}}, tp_{\mathsf{ex1}})$ where $\sigma_\emptyset = (B_\emptyset, \mu_\emptyset)$ is the empty partial solution (cf. page 117), $t_{\mathsf{ex1}} = (\mathsf{product2}, \mathsf{producedBy}, \mathsf{producer1})$ is the first matching triple that we consider during the execution, and $tp_{\mathsf{ex1}} \in B_{\mathsf{ex}}$ (cf. Example 6.1). It holds $tp_{\mathsf{ex1}} \notin B_\emptyset$ because $B_\emptyset = \emptyset$, and RDF triple t_{ex1} matches triple pattern $tp'_{\mathsf{ex1}} = \mu_\emptyset[tp_{\mathsf{ex1}}]$. Note, $\mu_\emptyset[tp_{\mathsf{ex1}}] = tp_{\mathsf{ex1}}$ because $\mathrm{dom}(\mu_\emptyset) = \emptyset$.

The QE state, denoted by st_{ex0}, in which the execution performs this AE task τ_{ex1} is the initial state after looking up the seed URIs. At this point we only know the empty partial solution $\sigma_\emptyset$ and the initially discovered subweb $\mathfrak{D}_{\mathsf{init}(S_{\mathsf{ex}}, W_{\mathsf{ex}})}$ of the queried Web of Linked Data W_{ex} ($\mathfrak{D}_{\mathsf{init}(S_{\mathsf{ex}}, W_{\mathsf{ex}})}$ is given in Example 6.5, page 119). Hence, we have $st_{\mathsf{ex0}} = (\{\sigma_\emptyset\}, \mathfrak{D}_{\mathsf{init}(S_{\mathsf{ex}}, W_{\mathsf{ex}})})$.

Performing AE task τ_{ex1} comprises (i) computing the $(t_{\mathsf{ex1}}, tp_{\mathsf{ex1}})$-augmentation of $\sigma_\emptyset$ and (ii) executing the t_{ex1}-expansion of $\mathfrak{D}_{\mathsf{init}(S_{\mathsf{ex}}, W_{\mathsf{ex}})}$ in W_{ex}. As a result, the next QE state is $st_{\mathsf{ex1}} = (\{\sigma_\emptyset, \sigma_{\mathsf{ex1}}\}, \mathfrak{D}_{\mathsf{ex1}})$ where $\sigma_{\mathsf{ex1}} = \mathsf{AUG}(\sigma_\emptyset, t_{\mathsf{ex1}}, tp_{\mathsf{ex1}})$ (cf. Example 6.3, page 117) and $\mathfrak{D}_{\mathsf{ex1}} = \mathsf{EXP}(\mathfrak{D}_{\mathsf{init}(S_{\mathsf{ex}}, W_{\mathsf{ex}})}, t_{\mathsf{ex1}}, W_{\mathsf{ex}})$ (cf. Example 6.6, page 120). □

We now define the operation of performing an AE task formally:

Definition 6.9 (Performance of an AE task). Let $\tau = (\sigma, t, tp)$ be an AE task for a $\mathrm{C}_{\mathsf{LD(R)}}$ query over a Web of Linked Data W and let $st = (\mathcal{O}, R_{\mathfrak{D}})$ be a QE state for the same query over the same Web W. The *performance of* τ *in* st, denoted by $\tau[st]$, is a pair $(\mathcal{O}', R'_{\mathfrak{D}})$ where $\mathcal{O}' := \mathcal{O} \cup \{\mathsf{AUG}(\sigma, t, tp)\}$ and $R'_{\mathfrak{D}} := \mathsf{EXP}(R_{\mathfrak{D}}, t, W)$. □

We emphasize that the requirements for the elements σ, t and tp in our definition of AE tasks (Definition 6.8) are the same as the requirements for augmentation operations (cf. Definition 6.6, page 120). This equivalence allows us to define the performance of AE tasks as given in Definition 6.9.

However, Definition 6.9 per se does not guarantee that the result of performing an AE task τ in a QE state st is again a possible QE state for the corresponding query execution. Instead, we have to show that the resulting pair $\tau[st] = (\mathcal{O}', R'_{\mathfrak{D}})$ satisfies our definition of QE states (Definition 6.7, page 121). Unfortunately, showing that $R'_{\mathfrak{D}}$ is a discovered subweb of the corresponding reachable subweb R turns out to be tricky: Although we know that $R'_{\mathfrak{D}}$ is a discovered subweb of the complete queried Web (cf. Proposition 6.5, page 120), we may not have a guarantee that $R'_{\mathfrak{D}}$ is fully contained in the reachable subweb. We need such a guarantee to ensure soundness of subsequent augmentation operations (as discussed in the context of Definition 6.7). To resolve this dilemma, we restrict our model to c_{Match}-semantics; in this case we have the necessary guarantee as the following lemma shows (for the proof refer to Section E.10, page 229f).

Lemma 6.1. *Let $\mathcal{Q}^{B,S}_{c_{\mathsf{Match}}}$ be a $C_{\mathsf{LD(R)}}$ query (under c_{Match}-semantics); let R denote the $(S, c_{\mathsf{Match}}, B)$-reachable subweb of a Web of Linked Data W; and let $R_{\mathfrak{D}}$ be a discovered subweb of R. For any RDF triple t with (i) $t \in \mathsf{AllData}(R_{\mathfrak{D}})$ and (ii) t is a matching triple for a triple pattern $tp \in B$, it holds that $\mathsf{EXP}(R_{\mathfrak{D}}, t, W)$ is a discovered subweb of R.*

We explain the restriction to c_{Match}-semantics in Lemma 6.1 as follows: The query execution strategy that we model expands the discovered subweb of the queried Web only by looking up URIs from RDF triples that match a triple pattern in the query (as we demonstrate in Example 6.1, page 114). Therefore, this strategy enforces query-based reachability (cf. Section 4.1.1, page 62f). As a result, the strategy only supports $\mathrm{C}_{\mathsf{LD(R)}}$ queries under c_{Match}-semantics, and so does our execution model.

For the sake of conciseness, in the remainder of this dissertation we refer to these queries as *conjunctive c_{Match}-queries* (*$C_{\mathsf{LD(M)}}$ queries* for short) and omit the index "c_{Match}" in formulas.

Definition 6.10 ($\mathbf{C}_{\mathsf{LD(M)}}$ query). Let $S \subseteq \mathcal{U}$ be a finite set of URIs and let B be a nonempty BGP. The *$C_{\mathsf{LD(M)}}$ query* that uses B and S, denoted by $\mathcal{Q}^{B,S}$, is the $\mathrm{C}_{\mathsf{LD(R)}}$ query $\mathcal{Q}^{B,S}_{c_{\mathsf{Match}}}$ that uses the same B and S (and reachability criterion c_{Match}). □

Remark 6.2. Due to Definition 6.10 we have $\mathcal{Q}^{B,S}(W) = \mathcal{Q}^{B,S}_{c_{\mathsf{Match}}}(W)$ for any $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ and any Web of Linked Data W.

For $\mathrm{C}_{\mathsf{LD(M)}}$ queries we can now show the soundness of performing AE tasks:

Proposition 6.6. *Let $\mathcal{Q}^{B,S}$ be a $C_{\mathsf{LD(M)}}$ query and let W be a Web of Linked Data. If τ and st are an AE task and a QE state for $\mathcal{Q}^{B,S}$ over W, respectively, then $\tau[st]$ is also a QE state for $\mathcal{Q}^{B,S}$ over W.*

Proof. Let $st = (\mathcal{O}, R_{\mathfrak{D}})$ and $\tau[st] = (\mathcal{O}', R'_{\mathfrak{D}})$. To show that $\tau[st]$ is a QE state for $\mathcal{Q}^{B,S}$ over W, we have to prove the following two claims (cf. Definition 6.7, page 121):

(i) $\mathcal{O}'$ is a finite set of partial solutions for $\mathcal{Q}^{B,S}$ in W, and (ii) $R'_{\mathfrak{D}}$ is a discovered subweb of the $(S, c_{\mathsf{Match}}, B)$-reachable subweb of W. The first claim follows from Definition 6.9 (cf. page 122), Proposition 6.4 (cf. page 117), and the fact that $\mathcal{O}$ is a finite set of partial solutions for $\mathcal{Q}^{B,S}$ in W (cf. Definition 6.7). Similarly, the second claim follows from Definition 6.9, Lemma 6.1 (cf. page 123), and the fact that $R_{\mathfrak{D}}$ is a discovered subweb of the $(S, c_{\mathsf{Match}}, B)$-reachable subweb of W (cf. Definition 6.7). ■

In the following proposition we also show that the order in which a query execution performs AE tasks is irrelevant w.r.t. the resulting QE state. Furthermore, performing the same AE task multiple times does not affect the resulting QE state.

Proposition 6.7. *If τ_1 and τ_2 are AE tasks for a $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ over a Web of Linked Data W and st is a QE state for $\mathcal{Q}^{B,S}$ over W, then (i) $\tau_1[\tau_2[st]] = \tau_2[\tau_1[st]]$ and (ii) $\tau_1[\tau_1[st]] = \tau_1[st]$.*

Proof. Let $st = (\mathcal{O}, R_{\mathfrak{D}})$ and $R_{\mathfrak{D}} = (D_{\mathfrak{D}}, data_{\mathfrak{D}}, adoc_{\mathfrak{D}})$. Then, Proposition 6.7 follows trivially from the definitions of augmentation (Definition 6.3, page 117) and of expansion (Definition 6.6, page 120), and the fact that $\mathcal{O}$, $D_{\mathfrak{D}}$, and $\mathrm{dom}(adoc_{\mathfrak{D}})$ are sets. ■

For our discussion so far we implicitly take the AE tasks that a query execution can perform as given. Although we can enumerate the set of all AE tasks for a query execution completely, such an enumeration requires the availability of all partial solutions for the query and of all RDF triples from any reachable LD document (see Definition 6.8 of AE tasks on page 122). Since this information is only available partially in most QE states, certain AE tasks are "hidden" in such a state. We shall ensure that query execution in our model may not perform such a task as long as it is hidden. As a basis we provide a formal definition of *hidden AE tasks*.

Definition 6.11 (Hidden AE Task). Let $\mathcal{Q}^{B,S}$ be a $\mathrm{C}_{\mathsf{LD(M)}}$ query; let W be a Web of Linked Data; let $st = (\mathcal{O}, R_{\mathfrak{D}})$ be a QE state for $\mathcal{Q}^{B,S}$ over W. An AE task $\tau = (\sigma, t, tp)$ for $\mathcal{Q}^{B,S}$ over W is *hidden* in st if $\sigma \notin \mathcal{O}$ or $t \notin \mathsf{AllData}(R_{\mathfrak{D}})$. □

Remark 6.3. Given Definition 6.11 and Definition 6.9 (cf. page 122), it is easy to see that any AE task that is *not* hidden in a particular QE state st_x is also not hidden in all QE states that may result from performing an arbitrary AE task in st_x.

Example 6.8. Recall the first example AE task, $\tau_{\mathsf{ex1}} = (\sigma_\emptyset, t_{\mathsf{ex1}}, tp_{\mathsf{ex1}})$, which our example execution performs in the initial QE state $st_{\mathsf{ex0}} = (\{\sigma_\emptyset\}, \mathfrak{D}_{\mathsf{init}(S_{\mathsf{ex}}, W_{\mathsf{ex}})})$ (cf. Example 6.7, page 122). This task is *not* hidden in state st_{ex0}. It is also not hidden in the second QE state $st_{\mathsf{ex1}} = \tau_{\mathsf{ex1}}[st_{\mathsf{ex0}}]$ (or in any other, subsequent state of the execution).

The second AE task that our example execution performs is $\tau_{\mathsf{ex2}} = (\sigma_{\mathsf{ex1}}, t_{\mathsf{ex2}}, tp_{\mathsf{ex2}})$ where $\sigma_{\mathsf{ex1}} = (\{tp_{\mathsf{ex1}}\}, \mu_{\mathsf{ex1}})$ and RDF triple $t_{\mathsf{ex2}} = (\mathsf{product2}, \mathsf{name}, \text{"Product2"})$ matches triple pattern $\mu_{\mathsf{ex1}}[tp_{\mathsf{ex2}}] = (\mathsf{product2}, \mathsf{name}, ?productName)$. This task is hidden in the initial state st_{ex0} because $\sigma_{\mathsf{ex1}} \notin \{\sigma_\emptyset\}$ and also $t_{\mathsf{ex2}} \notin \mathsf{AllData}(\mathfrak{D}_{\mathsf{init}(S_{\mathsf{ex}}, W_{\mathsf{ex}})})$. However, in the second QE state st_{ex1}—in which our execution performs this task—the task is not hidden (anymore): Recall from Example 6.7 that $st_{\mathsf{ex1}} = (\{\sigma_\emptyset, \sigma_{\mathsf{ex1}}\}, \mathfrak{D}_{\mathsf{ex1}})$ and, thus,

$\sigma_{\mathsf{ex1}} \in \{\sigma_\emptyset, \sigma_{\mathsf{ex1}}\}$ and $t_{\mathsf{ex2}} \in \mathsf{AllData}(\mathfrak{D}_{\mathsf{ex1}})$. The latter holds because t_{ex2} is contained in the data of LD document d_{p2} and this document is added to the discovered subweb by performing the previous AE task τ_{ex1} (which executes the t_{ex1}-expansion of $\mathfrak{D}_{\mathsf{init}(S_{\mathsf{ex}},W_{\mathsf{ex}})}$ in W_{ex}, as discussed in Example 6.7). □

We aim to ensure that a query execution in our model only performs AE tasks that are not hidden in the particular QE state in which the execution performs them. On the other hand, we note that the exist cases in which query execution may not make progress by performing certain AE tasks in certain QE states, even if these tasks are not hidden (case (ii) of Proposition 6.7 is a trivial example, cf. page 124). To identify AE tasks that guarantee progress in a given QE state we introduce the concept of *open AE tasks*:

Definition 6.12 (Open AE Task). Let $\mathcal{Q}^{B,S}$ be a $\mathrm{C}_{\mathsf{LD(M)}}$ query; let W be a Web of Linked Data; let *st* be a QE state for $\mathcal{Q}^{B,S}$ over W. An AE task τ for $\mathcal{Q}^{B,S}$ over W is *open* in *st* if (i) τ is not hidden in *st* and (ii) $st \neq \tau[st]$. To denote the set of all AE tasks (for $\mathcal{Q}^{B,S}$ over W) that are open in a QE state $st = (\mathfrak{S}, R_{\mathfrak{D}})$ we write $\mathrm{Open}(\mathfrak{S}, R_{\mathfrak{D}})$. □

6.3.7. Abstract Query Execution Procedure

We now use the introduced concepts to define an abstract procedure with which we formalize a query execution that applies the execution strategy demonstrated in Example 6.1 (cf. page 114). Algorithm 6.1 illustrates this abstract procedure, which we call *tbExec*. The input for *tbExec* is a nonempty BGP B, a finite set of (seed) URIs S, and a Web of Linked Data W. Hence, executions of this procedure compute $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ over W. The remainder of this section describes this procedure briefly.

In Algorithm 6.1 we denote the incrementally progressing QE state of *tbExec* executions by $(\mathfrak{P}, \mathfrak{D})$. That is, at any point during an execution of *tbExec*(B,S,W), $\mathfrak{P}$ is the (finite) set of all currently constructed partial solutions (for $\mathcal{Q}^{B,S}$ in W), and $\mathfrak{D}$ is the currently discovered subweb of W.

After initializing $\mathfrak{P}$ and $\mathfrak{D}$ (cf. lines 1 and 2 in Algorithm 6.1), the procedure amounts to a continuous execution of open AE tasks. We represent this continuous process by a loop (lines 3 to 9). Each iteration of this loop performs an open AE task (lines 5 to 7) and checks whether the newly constructed partial solution (E', μ') covers the whole BGP of the executed query; if this is the case, the valuation μ' in (E', μ') must be reported as a solution for the query (line 8). We emphasize that the set of all open AE tasks, $\mathrm{Open}(\mathfrak{P}, \mathfrak{D})$, changes when the query execution performs such a task. The loop terminates when no more open AE tasks exist for the current QE state.

If, however, the $(S, c_{\mathsf{Match}}, B)$-reachable subweb of the queried Web of Linked Data W is infinite, then the set of all AE tasks for $\mathcal{Q}^{B,S}$ over W is infinite as well. In such a case, there always exist open AE tasks during any execution of *tbExec*(B,S,W) and, thus, any such execution continues ad infinitum (as long as it is not stopped externally). Such a non-terminating execution is consistent with Proposition 4.10 (cf. page 87).

We emphasize the abstract nature of *tbExec*. The fact that we model query execution as a single loop that performs (open) AE tasks sequentially does not imply that our execution model has to be implemented in such a strictly sequential form. Instead, different

Algorithm 6.1 *tbExec*(B, S, W) – Compute query result $\mathcal{Q}^{B,S}(W)$.

```
1: $\mathfrak{P} := \{\sigma_\emptyset\}$            // $\sigma_\emptyset$ is the empty partial solution (cf. page 117)
2: $\mathfrak{D} := \mathfrak{D}_{\mathsf{init}(S,W)}$       // $\mathfrak{D}_{\mathsf{init}(S,W)}$ is the $S$-seed subweb of $W$ (cf. Definition 6.5, page 119)
3: while Open$(\mathfrak{P}, \mathfrak{D}) \neq \emptyset$ do
4:     Choose open AE task $(\sigma, t, tp) \in$ Open$(\mathfrak{P}, \mathfrak{D})$
5:     $(E', \mu') := \mathsf{AUG}(\sigma, t, tp)$      // Perform the selected AE task in the
6:     $\mathfrak{P} := \mathfrak{P} \cup \{(E', \mu')\}$        // current QE state; this performance
7:     $\mathfrak{D} := \mathsf{EXP}(\mathfrak{D}, t, W)$         // indirectly changes Open$(\mathfrak{P}, \mathfrak{D})$.
8:     if $E' = B$ then report $\mu'$ endif
9: end while
```

implementation approaches are possible, including implementations that perform multiple open AE tasks in parallel. The nondeterministic selection of open AE tasks in line 4 of Algorithm 6.1 provides the flexibility for interpreting our model in terms of diverse implementation techniques. In contrast to the concrete (implementable) algorithms of an actual implementation approach (such as the iterator-based approach discussed in the following chapter), we understand *tbExec* as an instrument for presenting and for studying the general idea of the traversal-based query execution strategy outlined in Example 6.1 (cf. page 114).

6.4. Soundness and Completeness

We now show that the traversal-based query execution strategy captured by our execution model is sound and complete.

Theorem 6.1. *Let W be a Web of Linked Data and let $\mathcal{Q}^{B,S}$ be a $C_{\mathsf{LD}(M)}$ query.*

- *Soundness: For any valuation μ reported by an execution of tbExec(B, S, W) it holds that $\mu \in \mathcal{Q}^{B,S}(W)$.*
- *Completeness: There exist executions of tbExec(B, S, W) that eventually report all solutions $\mu \in \mathcal{Q}^{B,S}(W)$.*

As a basis for proving the soundness we use the following lemma.

Lemma 6.2. *Let W be a Web of Linked Data and let $\mathcal{Q}^{B,S}$ be a $C_{\mathsf{LD}(M)}$ query. At any point during an (arbitrary) execution of tbExec(B, S, W) it holds that (i) each $\sigma \in \mathfrak{P}$ is a partial solution for $\mathcal{Q}^{B,S}$ in W and (ii) $\mathfrak{D}$ is a discovered subweb of the $(S, c_{\mathsf{Match}}, B)$-reachable subweb of W.*

Our proof of Lemma 6.2 is based on Proposition 6.6 (cf. page 123) and can be found in Section E.11 (cf. page 230).

The following lemma provides a basis for proving completeness.

Lemma 6.3. *Let $W = (D, data, adoc)$ be a Web of Linked Data and let $\mathcal{Q}^{B,S}$ be a $C_{\mathsf{LD(M)}}$ query. There exist executions of tbExec(B, S, W) that have the following two properties:*

(1.) For each LD document $d \in D$ that is (c_{Match}, B)-reachable from S in W there exists an iteration (of the loop in tbExec) after which d is part of $\mathfrak{D}$.

(2.) For each partial solution $\sigma \in \mathcal{O}(\mathcal{Q}^{B,S}, W)$ there exists an iteration after which $\sigma \in \mathfrak{P}$.

We prove Lemma 6.3 in Section E.12 (cf. page 231ff) by using the notion of *FIFO-based executions* of *tbExec*, that are, executions of *tbExec* that use a FIFO strategy to choose an open AE task at line 4 of Algorithm 6.1. More precisely, such an execution always chooses an open AE task $\tau \in \text{Open}(\mathfrak{P}, \mathfrak{D})$ for which there does *not* exist another $\tau' \in \text{Open}(\mathfrak{P}, \mathfrak{D})$ such that (i) τ was hidden in an earlier QE state of the execution and (ii) τ' was not hidden in that state. Informally, we note that such an execution resembles a breadth-first search over the link graph of the corresponding reachable subweb of W.

Although we assume FIFO-based executions to prove Lemma 6.3, there may be other executions of *tbExec*(B, S, W) that also have both properties as stated in Lemma 6.3. On the other hand, there are executions of *tbExec* that do not have these properties. As a trivial example consider executions that use a LIFO strategy for choosing the next open AE task. If such an execution starts traversing along an infinite path in an (infinite) reachable subweb of a queried Web, then the execution never discovers LD documents that are not on this path; moreover, this execution does not terminate.

We claim that such incomplete and non-terminating executions are only possible in cases where the corresponding reachable subweb of the queried Web is infinite. For the sake of brevity we omit a formal verification of this claim; in the context of this dissertation it is sufficient to know that incomplete, non-terminating executions are possible at all (as shown by the aforementioned LIFO example). Furthermore, recall that the antecedent of the claim (that is, whether the corresponding reachable subweb is finite or infinite) is not LD machine decidable (cf. Theorem 4.1, page 68). We also note that even complete executions of *tbExec* (that have the two properties given in Lemma 6.3) do not terminate in the (not LD machine decidable) case that the corresponding reachable subweb is infinite (cf. Proposition 4.10, page 87).

We now use Lemmas 6.2 and 6.3 to prove Theorem 6.1 (cf. page 126).

Proof of Theorem 6.1. For the proof of *soundness* let μ be a valuation that an arbitrary execution of *tbExec*(B, S, W) reports in some iteration it_j. We show that $\mu \in \mathcal{Q}^{B,S}(W)$. Valuation μ originates from the pair (E, μ) that the execution of *tbExec*(B, S, W) constructs and adds to $\mathfrak{P}$ in iteration it_j. Since (E, μ) is a partial solution for $\mathcal{Q}^{B,S}$ in W (cf. Lemma 6.2, page 126) and *tbExec* reports μ only if $E = B$ (cf. line 8 in Algorithm 6.1), μ is a solution for $\mathcal{Q}^{B,S}$ in W (cf. Proposition 6.3, page 117); i.e., $\mu \in \mathcal{Q}^{B,S}(W)$.

For the proof of *completeness* let μ be an arbitrary solution for $\mathcal{Q}^{B,S}$ in W; i.e., $\mu \in \mathcal{Q}^{B,S}(W)$. We show that there exist executions of *tbExec*(B, S, W) that report μ. There exists a partial solution $\sigma = (E, \mu) \in \mathcal{O}(\mathcal{Q}^{B,S}, W)$ such that $E = B$. Due to

Lemma 6.3 there exists an execution of $tbExec(B, S, W)$ with the following property: During this execution there exists an iteration (of the loop in *tbExec*) that constructs partial solution σ and adds σ to $\mathfrak{P}$. This iteration reports μ because $E = B$ (cf. line 8 in Algorithm 6.1). ■

Theorem 6.1 verifies the applicability of our traversal-based query execution strategy for answering $\mathrm{C}_{\mathsf{LD(M)}}$ queries over a Web of Linked Data. Any implementation of our execution model is guaranteed to report query results that are sound. However, our discussion also shows that completeness of reported query results cannot be guaranteed by any implementation.

6.5. Summary

This chapter is dedicated to a general strategy for executing (conjunctive) Linked Data queries. This strategy intertwines traversal-based retrieval of data with a result construction process that generates solutions of a query result incrementally. Our main contribution in this context is a query execution model that provides a formal definition of the strategy. While the strategy may be implemented in various ways, our execution model is independent from specific implementation approaches and, thus, enables us to analyze the strategy in general.

We prove that our execution model allows for a sound and complete execution of any conjunctive Linked Data query under (reachability-based) c_{Match}-semantics. From our results in Chapter 4 we know that a complete execution of such a query does not terminate in cases where the corresponding reachable subweb of the queried Web of Linked Data is infinite. However, our discussion in this chapter also reveals that some implementations of our execution model may neither guarantee termination nor completeness in such a case (even if the expected query result is finite).

7. An Iterator-Based Implementation

Our execution model as presented in the previous chapter defines a traversal-based execution strategy for conjunctive Linked Data queries (under reachability-based c_{Match}-semantics). A query execution system that applies the strategy captured by this model, requires a concrete approach for implementing the execution model. In this chapter we focus on using the well-known iterator model [56] for such a purpose.

The iterator model introduces the concept of an *iterator* as a particular implementation of an operator that allows a consumer to get the results of the operation separately, one at a time. An iterator provides three functions: `Open`, `GetNext`, and `Close`. `Open` initializes the data structures needed to perform the operation; `GetNext` returns the next result of the operation; and `Close` ends the iteration and releases allocated resources.

Many DBMSs employ the iterator model for query execution [85], that is, query execution plans are implemented as a tree of iterators. Such an iterator tree computes a query result in a pull fashion: During execution the `GetNext` function of each iterator calls `GetNext` on its child iterator(s) and uses the input obtained by these calls for producing the next result(s). Employing iterators for query execution simplifies *"the code [that is] responsible for coordinating the execution of a plan"* [133, page 408].

Another advantage of the iterator model is that iterators naturally support an implementation of *pipelined execution plans* [133]. Such a plan consists of *non-blocking operators* that may work on partial input from preceding operators instead of having to consume (and materialize) all input results for producing output [151, 166]. The merits of using pipelined execution plans over non-pipelined plans are threefold:

1. Pipelining allows for a space-efficient query execution because *"only a small buffer is needed to exchange data between pipelined operations instead of storage for large temporary relations."* [151]

2. *"The disk I/O cost is significantly reduced since the intermediate relations [...] need not be [materialized and] written back to disks."* [34]

3. Pipelining reduces *"the perceived response time by an end user [because] the first tuples of the resulting relation [...] can be produced earlier [...]."* [34]

While the characteristics of using iterators for executing queries over a fixed set of data are widely known, we aim to achieve an understanding of the suitability of iterators for a traversal-based query execution. To this end, we investigate an approach that employs a pipeline of iterators for implementing the execution model introduced in the previous chapter. A distinguishing feature of the iterators used in this approach is that calling their `GetNext` function (to obtain intermediate results for computing a given query) has the desired side effect of a dynamic, traversal-based expansion of the query-local dataset.

After defining the iterator-based implementation approach we analyze the approach formally and experimentally. This analysis reveals a major drawback of the approach: The (computed) query result that we obtain by executing a $\mathrm{C}_{\mathsf{LD(M)}}$ query using the approach may only be a subset of the expected query result. Hence, as per our classification in Section 2.1.2 (cf. page 22ff), the approach is sound but not complete.

The chapter is structured as follows: Section 7.1 introduces the implementation approach. Section 7.2 and Section 7.3 present our formal and experimental analysis of this approach, respectively. Finally, Section 7.4 summarizes our results.

7.1. Definition

This section introduces our approach to implement traversal-based query execution using a pipeline of iterators. Essentially, such a pipeline evaluates the BGP of a $\mathrm{C}_{\mathsf{LD(M)}}$ query over a query-local dataset that the iterators augment continuously with data from the queried Web of Linked Data. To present the approach we first introduce the static case. That is, we first describe how pipelined iterators may be employed for evaluating BGPs over a fixed set of RDF triples. Our description introduces basic concepts for querying RDF data with iterators and, thus, lays the foundations for presenting the traversal-based implementation afterwards.

7.1.1. Iterators for Query Execution over RDF Data

A logical plan for evaluating a BGP is an operator tree with as many leaf nodes as there exist triple patterns in the BGP. Each of these leaf nodes represents a data access operation for such a triple pattern. The internal nodes (including the root node) represent join operations. In query execution systems that employ iterator-based pipelining to evaluate BGPs, such a logical plan usually takes the form of a *left-deep tree*, that is, a tree in which the right child of each internal node is a leaf node. Since the join is a commutative and associative operation [128, 140], multiple of these left-deep operator trees are possible for any given BGP (with at least two triple patterns). In fact, for a BGP with a cardinality of n, there exist $n!$ different left-deep trees. However, these trees differ only in the order in which the triple patterns of the BGP are associated with leaf nodes. Therefore, specifying such an order is sufficient to represent logical plans for evaluating BGPs.

Definition 7.1 (Logical Plan). Let B be a BGP. A *logical plan* for B is a pair $(B, \triangleleft)$ where $\triangleleft$ is a strict total order on the triple patterns in B. □

Given a logical plan $(B, \triangleleft)$ for a BGP B, we write $\mathrm{sub}_k^{\triangleleft}(B)$ to denote the subset of B that consists of the first k triple patterns according to the plan; i.e.,

$$\mathrm{sub}_k^{\triangleleft}(B) := \Big\{ tp \in B \,\Big|\, k > \big|\{ tp' \in B \,|\, tp' \triangleleft tp \}\big| \Big\} \qquad \text{for all } k \in \{0, \dots, |B|\}.$$

We emphasize that for $k = 0$, $\mathrm{sub}_0^{\triangleleft}(B) = \emptyset$ holds, and for $k = |B|$, $\mathrm{sub}_{|B|}^{\triangleleft}(B) = B$.

Listing 7.1 `Open`, `GetNext`, and `Close` functions of the root iterator I_0 used for evaluating BGPs over a fixed, a-priori defined set of RDF triples G.

```
FUNCTION Open
  $ready$ := true

FUNCTION GetNext
  if $ready$ = true then
      $ready$ := false
      return $\mu_\emptyset$ // empty solution; dom($\mu_\emptyset$) = $\emptyset$
  else
      return EndOfFile
  end if

FUNCTION Close
  // nothing to do
```

As mentioned before, for a BGP B with a cardinality of $n = |B|$, we have $n!$ possible logical plans. Selecting one of these plans for execution is a query optimization problem that is out of scope of this dissertation (nonetheless, our experimental analysis in Section 7.3 addresses the question of "good" logical plans partially). Therefore, for the following description of the iterator implementation, we assume an arbitrary plan is given.

For an actual query execution the (selected) logical plan must be transformed into a physical query execution plan. In our context such a *physical query execution plan* (for short *physical plan*) is a pipeline of iterators whose functions `Open`, `GetNext`, and `Close` are given in Listing 7.1 and Listing 7.2. The pipeline is constructed as follows.

Assume a BGP B of size n and a corresponding logical plan $(B, \triangleleft)$. The physical plan for $(B, \triangleleft)$ consists of $n{+}1$ iterators, denoted by I_0 to I_n, each of which returns a set of valuations as output. These iterators are organized in a pipeline such that for each $k \in \{1, \dots, n\}$, iterator I_k consumes the output of I_{k-1}. Except for I_0, each iterator is responsible for a particular triple pattern from B; more precisely, for each $k \in \{1, \dots, n\}$, iterator I_k is responsible for the triple pattern $tp \in \mathrm{sub}_k^{\triangleleft}(B) \setminus \mathrm{sub}_{k-1}^{\triangleleft}(B)$. To denote the triple pattern of iterator I_k we write $\mathrm{tp}(I_k)$.

Iterator I_0 is a special iterator; it always only provides a single empty valuation $\mu_\emptyset$ (i.e., $\mathrm{dom}(\mu_\emptyset) = \emptyset$). Listing 7.1 specifies the functions `Open`, `GetNext`, and `Close` for I_0. Since such an iterator is always the first iterator in our physical plans, we call this iterator the *root iterator*.

Iterators I_1 to I_n perform the same `Open`, `GetNext`, and `Close` function. Listing 7.2 specifies these functions. We briefly describe the operation implemented by the `GetNext` function: The valuations that the `GetNext` function of an iterator I_k (with $k \in \{1, \dots, n\}$) returns are solutions for BGP $E_k = \mathrm{sub}_k^{\triangleleft}(B)$. To produce these solutions iterator I_k executes the following three steps repeatedly: First, I_k consumes a valuation μ_{input} from its direct predecessor I_{k-1} (cf. line 4 in Listing 7.2) and applies this valuation to its triple

Listing 7.2 `Open`, `GetNext`, and `Close` functions of an iterator used for evaluating BGPs over a fixed, a-priori defined set of RDF triples G.

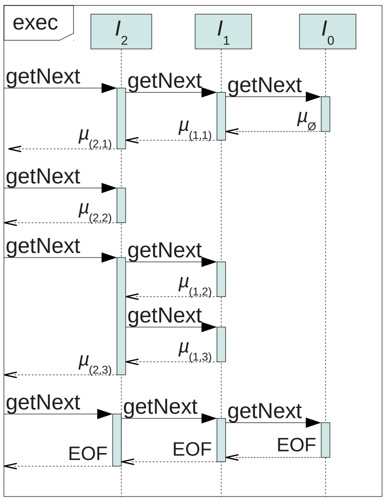
```
Require:
    $tp$ – a triple pattern
    $I_{\mathsf{pred}}$ – a predecessor iterator
    $G$ – the queried set of RDF triples

FUNCTION Open
 1: $I_{\mathsf{pred}}$.Open // initialize the input iterator
 2: $\Omega_{\mathsf{tmp}} := \emptyset$ // used for holding (precomputed) valuations between calls of the GetNext function

FUNCTION GetNext
 3: while $\Omega_{\mathsf{tmp}} = \emptyset$ do
 4:     $\mu_{\mathsf{input}} := I_{\mathsf{pred}}$.GetNext // consume valuation from the input iterator
 5:     if $\mu_{\mathsf{input}} = \textsc{EndOfFile}$ then
 6:         return EndOfFile
 7:     end if
 8:     $tp' := \mu_{\mathsf{input}}[tp]$
 9:     $T := \{ t \in G \,|\, t \text{ is a matching triple for } tp' \}$
10:     $\Omega_{\mathsf{tmp}} := \{ \mu_{\mathsf{input}} \cup \mu' \,|\, \mu' \text{ is a valuation with } \mathrm{dom}(\mu') = \mathrm{vars}(tp') \text{ and } \mu'[tp'] \in T \}$
11: end while
12: $\mu :=$ an element in $\Omega_{\mathsf{tmp}}$
13: $\Omega_{\mathsf{tmp}} := \Omega_{\mathsf{tmp}} \setminus \{\mu\}$
14: return $\mu$

FUNCTION Close
15: $I_{\mathsf{pred}}$.Close // close the input iterator
```

pattern $tp_k = \mathrm{tp}(I_k)$, resulting in a triple pattern $tp'_k = \mu_{\mathsf{input}}[tp_k]$ (cf. line 8); second, I_k (pre)computes a set of solutions by finding matching triples for tp'_k in the queried dataset (cf. lines 9 and 10); and, third, I_k returns each of the precomputed solutions, one after another (cf. lines 12 to 14).

Example 7.1. Let $B_{\mathsf{ex}} = \{tp_1, tp_2\}$ be a BGP with the following two triple patterns:

$$tp_1 = (?p, \mathsf{producedBy}, \mathsf{producer1}) \,, \qquad tp_2 = (?o, \mathsf{offeredProduct}, ?p) \,.$$

For an evaluation of B_{ex} we assume a logical plan $(B_{\mathsf{ex}}, \lhd_{\mathsf{ex}})$ such that $tp_1 \lhd_{\mathsf{ex}} tp_2$. Hence, in the corresponding physical plan, iterator I_1 is responsible for triple pattern tp_1 and I_2 for tp_2 (i.e., $\mathrm{tp}(I_1) = tp_1$ and $\mathrm{tp}(I_2) = tp_2$). The sequence diagram in Figure 7.1(a) illustrates an execution of this plan over a set of RDF triples G_{ex} that is given as follows:



(a) Sequence diagram that illustrates the interaction between iterators during the example execution.

$$\begin{aligned}
\Omega_{\mathsf{tmp}(1)} &= \{\mu_{(1,1)}, \mu_{(1,2)}, \mu_{(1,3)}\} \\
\Omega_{\mathsf{tmp}(2,1)} &= \{\mu_{(2,1)}, \mu_{(2,2)}\} \\
\Omega_{\mathsf{tmp}(2,2)} &= \emptyset \\
\Omega_{\mathsf{tmp}(2,3)} &= \{\mu_{(2,3)}\}
\end{aligned}$$

$$\begin{aligned}
\mu_{(1,1)} &= \{?p \to \mathsf{product1}\} \\
\mu_{(2,1)} &= \{?p \to \mathsf{product1}, ?o \to \mathsf{offer1.1}\} \\
\mu_{(2,2)} &= \{?p \to \mathsf{product1}, ?o \to \mathsf{offer2.1}\} \\
\mu_{(1,2)} &= \{?p \to \mathsf{product2}\} \\
\mu_{(1,3)} &= \{?p \to \mathsf{product3}\} \\
\mu_{(2,3)} &= \{?p \to \mathsf{product3}, ?o \to \mathsf{offer1.2}\}
\end{aligned}$$

(b) All sets of (intermediate) solutions that the iterators in the example execution (pre)compute.

Figure 7.1.: A particular, iterator-based execution of the example BGP over the set of RDF triples as discussed in Example 7.1.

$$\begin{aligned}
G_{\mathsf{ex}} = \big\{\ & (\mathsf{product1}, \mathsf{producedBy}, \mathsf{producer1}), (\mathsf{offer1.1}, \mathsf{offeredProduct}, \mathsf{product1}), \\
& (\mathsf{product2}, \mathsf{producedBy}, \mathsf{producer1}), (\mathsf{offer1.2}, \mathsf{offeredProduct}, \mathsf{product3}), \\
& (\mathsf{product3}, \mathsf{producedBy}, \mathsf{producer1}), (\mathsf{offer2.1}, \mathsf{offeredProduct}, \mathsf{product1})\ \big\}\,.
\end{aligned}$$

As it can be seen from the diagram, the first execution of the `GetNext` function of iterator I_1 begins with consuming the empty valuation $\mu_\emptyset$ from root iterator I_0. This valuation corresponds to μ_{input} in Listing 7.2 (cf. line 4). Based on $\mu_\emptyset$, iterator I_1 precomputes a set $\Omega_{\mathsf{tmp}(1)}$ of solutions for triple pattern $tp'_1 = \mu_\emptyset[tp_1]$ (cf. lines 8 to 10 in Listing 7.2). Note, $tp'_1 = tp_1$ because $\mathrm{dom}(\mu_\emptyset) = \emptyset$. Since the queried data contains three RDF triples that match tp'_1, it holds that $\Omega_{\mathsf{tmp}(1)} = \{\mu_{(1,1)}, \mu_{(1,2)}, \mu_{(1,3)}\}$ where $\mu_{(1,1)} = \{?p \to \mathsf{product1}\}$, $\mu_{(1,2)} = \{?p \to \mathsf{product2}\}$, and $\mu_{(1,3)} = \{?p \to \mathsf{product3}\}$.

After precomputing $\Omega_{\mathsf{tmp}(1)}$, iterator I_1 removes an (arbitrary) valuation from this precomputed set and returns this valuation as the first result of its operation (cf. lines 12 to 14). In our example execution the returned valuation is $\mu_{(1,1)}$ (cf. Figure 7.1(a)).

Using valuation $\mu_{(1,1)}$ as input, iterator I_2 computes its set Ω_{tmp} for triple pattern $tp'_2 = \mu_{(1,1)}[tp_2] = (?o, \mathsf{offeredProduct}, \mathsf{product1})$. To denote this particular version of Ω_{tmp} we write $\Omega_{\mathsf{tmp}(2,1)}$. Since two RDF triples in G_{ex} match the triple pattern tp'_2, the iterator computes $\Omega_{\mathsf{tmp}(2,1)} = \{\mu_{(2,1)}, \mu_{(2,2)}\}$ with $\mu_{(2,1)} = \mu_{(1,1)} \cup \{?o \to \mathsf{offer1.1}\}$ and $\mu_{(2,2)} = \mu_{(1,1)} \cup \{?o \to \mathsf{offer2.1}\}$. It is easy to see that each of the two valuations in

$\Omega_{\mathsf{tmp}(2,1)}$ is a solution for BGP $\mathrm{sub}_2^{\lhd_{\mathsf{ex}}}(B_{\mathsf{ex}}) = \{tp_1, tp_2\}$. Iterator I_2 concludes the first execution of its `GetNext` function by reporting $\mu_{(2,1)}$ (after removing it from Ω_{tmp}).

In response to the second call of its `GetNext` function, iterator I_2 returns the other precomputed solution, that is, $\mu_{(2,2)}$. As a consequence, Ω_{tmp} is empty when the query execution system requests another solution from I_2 (by calling the `GetNext` function of I_2 a third time). Hence, at the begin of the third execution of `GetNext`, I_2 consumes the next valuation from its predecessor I_1. Let this valuation be $\mu_{(1,2)}$. The set Ω_{tmp} that I_2 may construct based on $\mu_{(1,2)}$, denoted by $\Omega_{\mathsf{tmp}(2,2)}$, is empty because G_{ex} does not contain a matching triple for triple pattern $\mu_{(1,2)}[tp_2] = (?o, \mathsf{offeredProduct}, \mathsf{product2})$. Therefore, I_2 consumes another valuation from I_1.

In the remaining steps of our example query execution, the iterators proceed as illustrated in Figure 7.1(a); Figure 7.1(b) enumerates all (intermediate) solutions that the iterators report and consume, as well as all versions of the respective set Ω_{tmp} that the iterators precompute. □

7.1.2. Iterators for Traversal-Based Query Execution

We now adapt the iterators to define an implementation approach for our traversal-based query execution strategy. Hence, this approaches focuses on executing $\mathrm{C}_{\mathsf{LD(M)}}$ queries over a Web of Linked Data. As in the static case discussed before, logical (query execution) plans specify an order over the BGP of the $\mathrm{C}_{\mathsf{LD(M)}}$ query that has to be executed and a corresponding physical plan is a pipeline of iterators. However, to implement a traversal-based query execution we require a different kind of iterators. In this section we introduce these iterators which we call *link traversing iterators*.

All link traversing iterators in a pipelined execution plan share a data structure that represents the (currently) discovered subweb of the queried Web of Linked Data. As in the abstract procedure *tbExec* of our query execution model (cf. Section 6.3.7, page 125f), we denote this discovered subweb by $\mathfrak{D}$. During query execution $\mathfrak{D}$ grows monotonically.

To perform the initialization of $\mathfrak{D}$ at the beginning of iterator-based query executions we extend the `Open` function of root iterator I_0. Listing 7.3 specifies the adjusted function

Listing 7.3 `Open` function for the root iterator I_0 in our iterator implementation of traversal-based query execution (Functions `GetNext` and `Close` for I_0 are the same as in Listing 7.1 on page 131).

```
Require:
    $S$ – a finite set of seed URIs ($S \subset \mathcal{U}$)
    $W$ – the queried Web of Linked Data
    $\mathfrak{D}$ – the currently discovered part of $W$ (note, all iterators have access to $\mathfrak{D}$)

FUNCTION Open
1: $\mathfrak{D} := \mathfrak{D}_{\mathsf{init}(S,W)}$ // $\mathfrak{D}_{\mathsf{init}(S,W)}$ is the $S$-seed part of $W$ (cf. Definition 6.5, page 119)
2: $ready$ := true
```

Listing 7.4 `GetNext` function for an iterator in our iterator implementation of traversal-based query execution (Functions `Open` and `Close` are the same as in Listing 7.2 on page 132).

```
Require:
    $tp$ – a triple pattern
    $I_{\mathsf{pred}}$ – a predecessor iterator
    $W$ – the queried Web of Linked Data
    $\mathfrak{D}$ – the currently discovered part of $W$ (all iterators in the pipeline access this $\mathfrak{D}$)
    $\Omega_{\mathsf{tmp}}$ – a set that allows the iterator to keep (precomputed) partial solutions between
          calls of this GetNext function; $\Omega_{\mathsf{tmp}}$ is empty initially (cf. Listing 7.2)

 1: while $\Omega_{\mathsf{tmp}} = \emptyset$ do
 2:    $\mu_{\mathsf{input}} := I_{\mathsf{pred}}$.GetNext  // consume valuation from the input iterator
 3:    if $\mu_{\mathsf{input}}$ = EndOfFile then
 4:       return EndOfFile
 5:    end if

 6:    $tp' := \mu_{\mathsf{input}}[tp]$
 7:    $G_{\mathsf{snap}} := \mathsf{AllData}(\mathfrak{D})$
 8:    $T := \{ t \in G_{\mathsf{snap}} \,|\, t$ is a matching triple for $tp' \}$
 9:    $\Omega_{\mathsf{tmp}} := \{ \mu_{\mathsf{input}} \cup \mu' \,|\, \mu'$ is a valuation with $\mathrm{dom}(\mu') = \mathrm{vars}(tp')$ and $\mu'[tp'] \in T \}$

10:    for all $t \in T$ do
11:       $\mathfrak{D} := \mathsf{EXP}(\mathfrak{D}, t, W)$  // $\mathsf{EXP}(\mathfrak{D}, t, W)$ denotes the $t$-expansion of $\mathfrak{D}$ in $W$
12:    end for                    // (cf. Definition 6.6, page 120)

13: end while

14: $\mu :=$ an element in $\Omega_{\mathsf{tmp}}$
15: $\Omega_{\mathsf{tmp}} := \Omega_{\mathsf{tmp}} \setminus \{\mu\}$
16: return $\mu$
```

for the link traversing version of I_0. Functions `GetNext` and `Close` of this iterator do not require adjustments; hence, they are the same as in Listing 7.1 (cf. page 131).

For the link traversing version of the iterators that consume and report valuations we also extend the functionality of their static counterparts. However, in this case, we have to adjust the `GetNext` function (`Open` and `Close` remain the same as in Listing 7.2 on page 132). Listing 7.4 specifies the adjusted `GetNext` function. Differences between this `GetNext` function and the `GetNext` function for the static case are highlighted in the listing. These differences are twofold:

1. While iterators for the static case compute valuations over a fixed set of RDF triples G, link traversing iterators use the set of all RDF triples in (a snapshot of) $\mathfrak{D}$ for computing valuations (compare line 9 in Listing 7.2 to line 8 in Listing 7.4).

2. In addition to computing (and reporting) valuations, link traversing iterators also perform the incremental expansion of $\mathfrak{D}$ that is characteristic for traversal-based

query execution (and not necessary in the static case). In particular, a link traversing iterator performs expand operations (per Definition 6.6, page 120) each time it precomputes the next version of its set Ω_{tmp}. For these operations the iterator uses the same set of matching triples that it uses for generating the valuations in Ω_{tmp} (cf. lines 10 to 12 in Listing 7.4).

Example 7.2. Let $\mathcal{Q}^{B_{\mathsf{ex}},S_{\mathsf{ex}}}$ be a $\mathrm{C}_{\mathsf{LD(M)}}$ query with a set $S_{\mathsf{ex}} = \{\mathsf{producer1}\}$ of seed URIs and a BGP $B_{\mathsf{ex}} = \{tp_1, tp_2\}$ that consists of the following two triple patterns:

$$tp_1 = (?product, \mathsf{producedBy}, \mathsf{producer1})\,, \qquad tp_2 = (?previous, \mathsf{oldVersionOf}, ?product)\,.$$

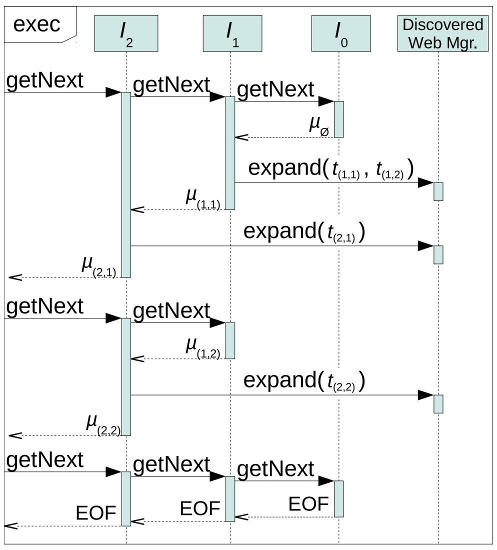
For a traversal-based execution of this query over our example Web W_{ex} (cf. Example 2.1, page 18) we assume a physical plan of link traversing iterators I_0 to I_2 such that I_0 is the root iterator and iterators I_1 and I_2 are responsible for triple patterns tp_1 and tp_2, respectively. The sequence diagram in Figure 7.2(a) illustrates an execution of this plan; Figure 7.2(b) enumerates all valuations that the iterators report and consume during this execution, as well as all versions of the set Ω_{tmp} precomputed by each iterator.

In contrast to the sequence diagram that illustrates our example execution for the static case (cf. Figure 7.1(a), page 133), the diagram in Figure 7.2(a) contains an additional lifeline. This lifeline represents a (hypothetical) component that manages the currently discovered subweb $\mathfrak{D}$ of the queried Web of Linked Data. We emphasize that we do not assume (or require) an explicit existence of such a management component in any traversal-based query execution system. Instead, the purpose of this lifeline is to illustrate the points at which iterators attempt to expand $\mathfrak{D}$.

The example execution begins with the initialization of $\mathfrak{D}$ by root iterator I_0. Hence, when iterator I_1 executes its `GetNext` function for the first time, $\mathfrak{D}$ consists of a single LD document, namely $d_{\mathsf{Pr1}} = adoc_{\mathsf{ex}}(\mathsf{producer1})$. The data in this document contains two RDF triples that match triple pattern $tp_1 = \mathrm{tp}(I_1)$. We denote these triples by $t_{(1,1)}$ and $t_{(1,2)}$ (cf. Figure 7.2(b)). Based on these matching triples, iterator I_1 precomputes two valuations: $\mu_{(1,1)} = \{?product \to \mathsf{product2}\}$ and $\mu_{(1,2)} = \{?product \to \mathsf{product3}\}$. Instead of immediately reporting one of these valuations to its predecessor I_2 (as it would happen in the static case), iterator I_1 first uses the two matching triples to expand $\mathfrak{D}$. As a result, $\mathfrak{D}$ consists of the LD documents d_{Pr1}, $d_{\mathsf{p2}} = adoc_{\mathsf{ex}}(\mathsf{product2})$, and $d_{\mathsf{p3}} = adoc_{\mathsf{ex}}(\mathsf{product3})$ when I_1 reports $\mu_{(1,1)}$ to I_2.

Based on a matching triple in $data_{\mathsf{ex}}(d_{\mathsf{p2}})$, iterator I_2 now precomputes valuation $\mu_{(2,1)}$ as an augmentation of $\mu_{(1,1)}$ (cf. Figure 7.2). Thus, I_2 benefits from the previous expansion of $\mathfrak{D}$ that led to the discovery of LD document d_{p2}. The execution proceeds as illustrated in Figure 7.2(a). □

As can be seen in the example, calling the `GetNext` function of a link traversing iterator may have the side effect of expanding $\mathfrak{D}$ (as desired for an implementation of traversal-based query execution). Since all iterators share the same data structure for $\mathfrak{D}$, the next iterator that computes valuations may benefit immediately from this expansion (and from all previous expansions). For instance, in the previous example, the expand operations performed by iterator I_1 enable iterator I_2 to compute valuations.



(a) Sequence diagram that illustrates the interaction between the link traversing iterators during the example execution.

$$t_{(1,1)} = (\mathsf{product2}, \mathsf{producedBy}, \mathsf{producer1})$$
$$t_{(1,2)} = (\mathsf{product3}, \mathsf{producedBy}, \mathsf{producer1})$$
$$t_{(2,1)} = (\mathsf{product1}, \mathsf{oldVersionOf}, \mathsf{product2})$$
$$t_{(2,2)} = (\mathsf{product1}, \mathsf{oldVersionOf}, \mathsf{product3})$$

$$\Omega_{\mathsf{tmp}(1)} = \{\mu_{(1,1)}, \mu_{(1,2)}\}$$
$$\Omega_{\mathsf{tmp}(2,1)} = \{\mu_{(2,1)}\}$$
$$\Omega_{\mathsf{tmp}(2,2)} = \{\mu_{(2,2)}\}$$

$$\mu_{(1,1)} = \{?product \rightarrow \mathsf{product2}\}$$
$$\mu_{(2,1)} = \{?previous \rightarrow \mathsf{product1}\} \cup \mu_{(1,1)}$$
$$\mu_{(1,2)} = \{?product \rightarrow \mathsf{product3}\}$$
$$\mu_{(2,2)} = \{?previous \rightarrow \mathsf{product1}\} \cup \mu_{(1,2)}$$

(b) Matching triples and all sets of (intermediate) solutions that the link traversing iterators (pre)compute during the example execution.

Figure 7.2.: A particular, iterator-based execution of the example $\mathrm{C}_{\mathsf{LD(M)}}$ query as discussed in Example 7.2.

We conclude our introduction of the iterator approach by summarizing the overall process for executing a $\mathrm{C}_{\mathsf{LD(M)}}$ query in Algorithm 7.5 (cf. page 138).

7.2. Formal Analysis

We now analyze the implementation approach formally and show the following property:

Theorem 7.1. *Any execution of Algorithm 7.5 for a $C_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ over a Web of Linked Data W reports a finite (possibly nonproper) subset of $\mathcal{Q}^{B,S}(W)$.*

Theorem 7.1 verifies that the implementation approach is sound and, thus, can be used for answering $\mathrm{C}_{\mathsf{LD(M)}}$ queries. However, the theorem also shows that the approach cannot guarantee answers that are complete query results.

We first discuss examples for iterator-based executions that provide an incomplete query result. Afterwards, we interpret the implementation approach in terms of our execution model; based on the results of this discussion we prove Theorem 7.1 (for the proof itself refer to page 144) and explain the incompleteness of the approach.

7.2.1. Examples for Incompleteness

Example 7.2 (cf. page 136) demonstrates a particular execution of Algorithm 7.5 for a given $\mathrm{C}_{\mathsf{LD(M)}}$ query. This execution reports two valuations. These two valuations

Algorithm 7.5 Overall process for an iterator-based execution of $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ over a Web of Linked Data W.

```
1: $n := |B|$
2: Select a logical plan $(B, \triangleleft)$ for $B$.

3: Create a data structure $\mathfrak{D}$ for the currently discovered subweb of $W$.
4: Use $S$, $W$, and $\mathfrak{D}$ to create root iterator $I_0$ (as defined in Listing 7.3, page 134).
5: for $k := 1$ to $n$ do
6:    Assume $tp_k \in B$ is the $k$-th triple pattern in $B$ according to order $\triangleleft$. Use $tp_k$,
      $I_{k-1}$, $W$, and $\mathfrak{D}$ to create link traversing iterator $I_k$ (as defined in Listing 7.4,
      page 135).
7: end for

8: $I_n$.Open // initialize all iterators recursively

9: repeat
10:    $\mu := I_n$.GetNext
11:    if $\mu \neq$ EndOfFile then report $\mu$ endif
12: until $\mu =$ EndOfFile

13: $I_n$.Close // close all iterators recursively
```

make up the complete (expected) query result for the example query over the example Web. However, other executions of the algorithm are also possible for the same example query (over the same example Web). Some of these (alternative) executions provide an incomplete query result only. In this section we discuss two of these executions, both of which reveal different characteristics of the iterator-based implementation approach.

We first focus on an execution that uses an alternative logical plan:

Example 7.3. For the query execution in Example 7.2 we assume a logical plan such that, in the corresponding physical plan, iterator I_1 is responsible for triple pattern $tp_1 = (?product, \mathsf{producedBy}, \mathsf{producer1})$ and I_2 for $tp_2 = (?previous, \mathsf{oldVersionOf}, ?product)$; i.e., $\mathrm{tp}(I_1) = tp_1$ and $\mathrm{tp}(I_2) = tp_2$. Let us now use the alternative logical plan $(B_{\mathsf{ex}}, \triangleleft'_{\mathsf{ex}})$ with $tp_2 \triangleleft'_{\mathsf{ex}} tp_1$. Hence, for the pipeline of iterators I'_0, I'_1, I'_2 that constitutes the corresponding (alternative) physical plan, it holds that $\mathrm{tp}(I'_1) = tp_2$ and $\mathrm{tp}(I'_2) = tp_1$.

The initialization of $\mathfrak{D}$ by the root iterator is the same for all (iterator-based) executions of the example query $\mathcal{Q}^{B_{\mathsf{ex}},S_{\mathsf{ex}}}$. Hence, after the initialization, $\mathfrak{D}$ consists of LD document $d_{\mathsf{Pr1}} = adoc_{\mathsf{ex}}(\mathsf{producer1})$, because the corresponding set of seed URIs is $S_{\mathsf{ex}} = \{\mathsf{producer1}\}$ (cf. Example 7.2). The example execution proceeds as follows: Algorithm 7.5 calls the `GetNext` function of I'_2, this function calls `GetNext` of I'_1, and I'_1 consumes the empty valuation $\mu_\emptyset$ from I'_0 (by calling `GetNext` of I'_0). Then, I'_1 tries to find matching triples for triple pattern $\mu_\emptyset[\mathrm{tp}(I'_1)] = (?previous, \mathsf{oldVersionOf}, ?product)$ in the current snapshot of $\mathfrak{D}$ (cf. line 8 in Listing 7.4, page 135). However, $data_{\mathsf{ex}}(d_{\mathsf{Pr1}})$ does not contain such a triple (cf. Figure 2.1, page 19). Thus, I'_1 cannot return an intermediate solution to its successor I'_2. As a result, the query execution process terminates with an incomplete (empty) query result. □

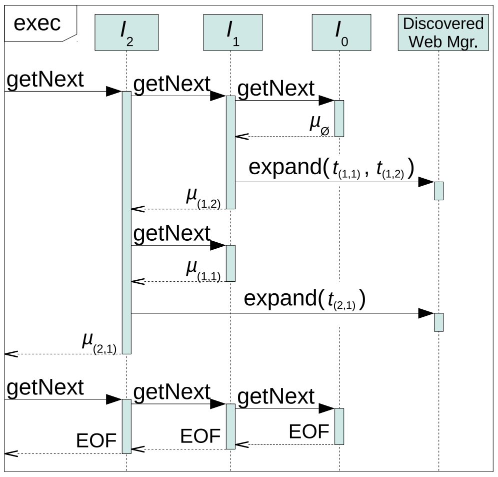


Figure 7.3.: Sequence diagram for the query execution outlined in Example 7.4. Valuations and RDF triples mentioned in the diagram are the same as given in Figure 7.2(b) (cf. page 137).

The example verifies our claim that the iterator implementation of traversal-based query execution may produce an incomplete query result. Our experiments shall even show that there exist queries for which *none* of all possible logical plans produces the complete query result (cf. Section 7.3.3, page 158ff). Moreover, by comparing Example 7.3 and Example 7.2, we also note that for different logical plans, the approach may produce different subsets of the (complete) query result. Thus, in contrast to traditional query execution scenarios, alternative logical plans for this approach are not guaranteed to be semantically equivalent (nonetheless, each plan is sound). We come back to this limitation after interpreting the implementation approach in terms of our query execution model in the next section.

While the previous example uses a different logical plan, executions of Algorithm 7.5 may even differ without varying the logical (and, thus, physical) plan: We emphasize the arbitrary order in which iterators may report precomputed valuations (cf. lines 14 to 16 in Listing 7.4, page 135). Due to this flexibility we may observe different, nonidentical query execution processes for the same physical plan. This type of nondeterministic behavior is not unusual and has no consequences in the traditional, static case. It may however have an impact in the dynamic case as the following example demonstrates.

Example 7.4. Let the pipeline of iterators I_0, I_1, I_2 be the same physical plan as in Example 7.2 (cf. page 136). Hence, it holds that $\mathrm{tp}(I_1) = tp_1$ and $\mathrm{tp}(I_2) = tp_2$. However, we now outline another possible execution of this physical plan. More precisely, we assume that iterator I_1 reports the precomputed valuations $\Omega_{\mathsf{tmp}(1)} = \{\mu_{(1,1)}, \mu_{(1,2)}\}$ in a different order than in Example 7.2. The sequence diagram in Figure 7.3 illustrates the execution process that we may observe in such a case.

Up to the point at which iterator I_1 completes its expansion of $\mathfrak{D}$ (at line 12 in Listing 7.4, page 135), the execution is identical to the execution in Example 7.2. Hence, at this point the currently discovered subweb $\mathfrak{D}$ of the queried example Web W_{ex} consists of the LD documents d_{Pr1}, d_{p2}, and d_{p3}. Now, iterator I_1 returns precomputed valuation $\mu_{(1,2)}$ to its successor I_2 (in contrast to valuation $\mu_{(1,1)}$ returned in Example 7.2). Based on this valuation, iterator I_2 uses the current snapshot of $\mathfrak{D}$ to find matching triples for triple pattern $\mu_{(1,2)}[\mathrm{tp}(I_2)] = (?\mathit{previous}, \mathsf{oldVersionOf}, \mathsf{product3})$. Although such a matching triple exists in LD document d_{p1} in the queried Web (cf. Example 2.1, page 18), this document has not been discovered and retrieved at this point. Therefore, I_2 cannot construct valuation $\mu_{(2,2)}$ as an augmentation of $\mu_{(1,2)}$ (as was possible during the execution in Example 7.2). Instead, I_2 discards $\mu_{(1,2)}$ and consumes the next valuation from I_1.

The second valuation consumed from I_1 is $\mu_{(1,1)}$. Based on $\mu_{(1,1)}$, iterator I_2 tries to find matching triples for triple pattern $\mu_{(1,1)}[\mathrm{tp}(I_2)] = (?\mathit{previous}, \mathsf{oldVersionOf}, \mathsf{product2})$. For this triple pattern the currently discovered subweb $\mathfrak{D}$ already contains matching triple $(\mathsf{product1}, \mathsf{oldVersionOf}, \mathsf{product2})$ (originating from LD document d_{p2}). Hence, I_2 constructs (and returns) the corresponding valuation $\mu_{(2,1)}$.

After that, the iterators return no more valuations and, thus, the execution outlined reports only valuation $\mu_{(2,1)}$ as a solution of the query result $\mathcal{Q}^{B_{\mathsf{ex}},S_{\mathsf{ex}}}(W_{\mathsf{ex}}) = \{\mu_{(2,1)}, \mu_{(2,2)}\}$. Although this answer is not empty (as the answer produced by the execution in Example 7.3), it is also not complete (in contrast to the answer in Example 7.2). □

The example demonstrates that the nondeterministic order in which link traversing iterators report precomputed valuations may have an impact on the query results provided by a pipeline of such iterators. We emphasize, however, that the alternative report order used in Example 7.4 (compared to the "original" order assumed in Example 7.2) is not the main reason for the incomplete query result reported by the execution. Instead, this alternative order merely allows us to observe the effect demonstrated in the example.

The actual reason why iterator I_2 cannot compute valuation $\mu_{(2,2)}$ in the example is that the iterators discard each input valuation after using it. While this "use and forget" strategy is typical for pipelined iterators, it presents a major limitation for our use case where it may lead to incomplete answers. On the other hand, due to this strategy the iterator implementation of traversal-based query execution has the following advantageous property as we shall see in the next section: It guarantees termination of query executions (even for queries that are not finitely computable by an LD machine).

Nonetheless, we may prevent the nondeterministic behavior of link traversing iterators by prescribing a particular (artificial) order in which iterators must report precomputed valuations. While such an adjustment does not address the issue of potentially incomplete query results, it ensures repeatability. Consequently, for most of our experiments we shall use such an adjustment. However, for our formal results in the following section we take into account that multiple executions are possible for a physical plan.

7.2.2. Alignment of the Implementation with the Execution Model

We now show formally that (and how) Algorithm 7.5 implements our traversal-based query execution model (defined in Section 6.3, page 115ff). Thereafter, we use the resulting propositions to prove Theorem 7.1 (cf. page 137). Furthermore, such an alignment shall allow us to provide a more informed explanation of the effects observed in the previous section. In the following, we first focus on the concept of partial solutions and then discuss the performance of open AE tasks by link traversing iterators.

In terms of our execution model, we may interpret each valuation computed and reported by a link traversing iterator as a partial solution (as introduced in Definition 6.2, page 116). The following proposition verifies this claim:

Proposition 7.1. *Let $\mathcal{Q}^{B,S}$ be a $C_{\mathsf{LD(M)}}$ query that uses a BGP B of size $n = |B|$; let $(B, \triangleleft)$ be a logical plan for executing $\mathcal{Q}^{B,S}$ over a Web of Linked Data W; let the pipeline of link traversing iterators $I_0, \ldots, I_n$ be the corresponding physical plan; and let exec denote an execution of this physical plan. For any $k \in \{0, \ldots, n\}$ and any valuation μ that iterator I_k computes and reports during exec it holds that the pair $\big(\mathrm{sub}_k^{\triangleleft}(B), \mu\big)$ is a partial solution for $\mathcal{Q}^{B,S}$ in W.*

Proof. Let $\Omega_{\mathsf{all}}^{exec}$ denote the set of all valuations that iterators I_0 to I_n compute and report during *exec*. We prove Proposition 7.1 by induction on the sequence in which iterators I_0 to I_n generate these valuations.

Base case: The first valuation generated by any of the iterators is the empty valuation $\mu_\emptyset$ that root iterator I_0 generates. Since $\mathrm{sub}_0^{\triangleleft}(B) = \emptyset$, the pair $\big(\mathrm{sub}_0^{\triangleleft}(B), \mu_\emptyset\big)$ is a partial solution for $\mathcal{Q}^{B,S}$ in W. In fact, this pair is the empty partial solution introduced on page 117 in Section 6.3.2.

Induction step: Let $\mu_{(k,i)} \in \Omega_{\mathsf{all}}^{exec} \setminus \{\mu_\emptyset\}$ be an arbitrary valuation computed during *exec* after I_0 generated $\mu_\emptyset$. Furthermore, let I_k be the iterator that computes $\mu_{(k,i)}$. We note that I_k cannot be the root iterator I_0 because I_0 generates no more valuations after generating $\mu_\emptyset$ (as can be seen in Listings 7.1 and 7.3). Let Ω denote the set of all valuations that all iterators I_0 to I_n precompute (during *exec*) before I_k computes $\mu_{(k,i)}$. By induction we assume that for each valuation $\mu_{(k',i')} \in \Omega$ (where $k' \in \{0, \ldots, n\}$ and $\mu_{(k',i')}$ is computed by iterator $I_{k'}$), the pair $\big(\mathrm{sub}_{k'}^{\triangleleft}(B), \mu_{(k',i')}\big)$ is a partial solution for $\mathcal{Q}^{B,S}$ in W. Based on this hypothesis we show that the pair $\big(\mathrm{sub}_k^{\triangleleft}(B), \mu_{(k,i)}\big)$ is also a partial solution for $\mathcal{Q}^{B,S}$ in W.

Let $\mu_{(k-1,j)}$ be the valuation that iterator I_k consumes from its predecessor I_{k-1} during the particular execution of its `GetNext` function in which I_k (pre)computes $\mu_{(k,i)}$. By induction, the pair $\big(\mathrm{sub}_k^{\triangleleft}(B), \mu_{(k-1,j)}\big)$ is a partial solution for $\mathcal{Q}^{B,S}$ in W. Hereafter, we write $\sigma_{(k-1,j)}$ to denote this partial solution.

Let $\mathfrak{D}_{(k-1,j)}$ be the particular snapshot of $\mathfrak{D}$ that I_k uses for generating $\mu_{(k,i)}$. Since I_k computes $\mu_{(k,i)}$ based on $\mu_{(k-1,j)}$, there must exist an RDF triple t' and a valuation μ' such that (i) $t' \in \mathsf{AllData}(\mathfrak{D}_{(k-1,j)})$, (ii) $\mu_{(k,i)} = \mu_{(k-1,j)} \cup \mu'$, (iii) $\mathrm{dom}(\mu') = \mathrm{vars}(tp'_{(k,j)})$, and (iv) $t' = \mu'[tp'_{(k,j)}]$, where $tp'_{(k,j)} = \mu_{(k-1,j)}[\mathrm{tp}(I_k)]$ (cf. line 9 in Listing 7.4, page 135). Thus, for RDF triple t' it also holds that $t' = \mu_{(k,i)}[\mathrm{tp}(I_k)]$.

Then, in terms of our execution model, the pair $\big(\mathrm{sub}_k^{\triangleleft}(B), \mu_{(k,i)}\big)$ is the $(t', \mathrm{tp}(I_k))$-augmentation of partial solution $\sigma_{(k-1,j)}$ (cf. Definition 6.3, page 117). By Proposition 6.4 (cf. page 117), this augmentation is a partial solution for $\mathcal{Q}^{B,S}$ in W if (i) $\sigma_{(k-1,j)}$ is a partial solution for $\mathcal{Q}^{B,S}$ in W, and (ii) $t' \in \mathsf{AllData}(R)$ where R is the $(S, c_{\mathsf{Match}}, B)$-reachable subweb of W. Since the former holds by induction, it remains to prove the latter. Since $t' \in \mathsf{AllData}(\mathfrak{D}_{(k-1,j)})$, we prove $t' \in \mathsf{AllData}(R)$ by showing that $\mathfrak{D}_{(k-1,j)}$ is a discovered subweb of R (in which case we have $\mathsf{AllData}(\mathfrak{D}_{(k-1,j)}) \subseteq \mathsf{AllData}(R)$).

After iterator I_0 initialized $\mathfrak{D}$ with $\mathfrak{D}_{\mathsf{init}(S,W)}$, each (other) iterator from the pipeline may expand $\mathfrak{D}$ only based on RDF triples that match the triple pattern for which the iterator is responsible (cf. lines 10 to 12 in Listing 7.4). Hence, snapshot $\mathfrak{D}_{(k-1,j)}$ is the result of expanding $\mathfrak{D}_{\mathsf{init}(S,W)}$ using triples that match a triple pattern from BGP B. Since $\mathfrak{D}_{\mathsf{init}(S,W)}$ is a discovered subweb of the $(S, c_{\mathsf{Match}}, B)$-reachable subweb of W (cf. Definition 6.5, page 119), we use Lemma 6.1 (cf. page 123) recursively to show that $\mathfrak{D}_{(k-1,j)}$ is also a discovered subweb of the $(S, c_{\mathsf{Match}}, B)$-reachable subweb of W. ■

Example 7.5. Revisiting the query execution discussed in Example 7.2 (cf. page 136), we may say that iterator I_1 implicitly generates partial solutions $\sigma_{(1,1)} = (\{tp_1\}, \mu_{(1,1)})$ and $\sigma_{(1,2)} = (\{tp_1\}, \mu_{(1,2)})$. Similarly, iterator I_2 (the other iterator in Example 7.2) generates partial solutions $\sigma_{(2,1)} = (\{tp_1, tp_2\}, \mu_{(2,1)})$ and $\sigma_{(2,2)} = (\{tp_1, tp_2\}, \mu_{(2,2)})$. □

Primarily, Proposition 7.1 shows that link traversing iterators (implicitly) generate partial solutions. However, the proposition also specifies these partial solutions. Thus, as a direct consequence of Proposition 7.1 we highlight the following property:

Remark 7.1. Let $I_0, \dots, I_n$ be a pipeline of link traversing iterators that executes logical plan $(B, \triangleleft)$ for $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ (where $n = |B|$). Then, for each $k \in \{0, \dots, n\}$, iterator I_k is the only iterator from the pipeline that may (implicitly) generate a partial solution $(E, \mu) \in \sigma(\mathcal{Q}^{B,S}, W)$ for which it holds that $E = \mathrm{sub}_k^{\triangleleft}(B)$ (where $\sigma(\mathcal{Q}^{B,S}, W)$ denotes the set of all partial solutions for $\mathcal{Q}^{B,S}$ over a queried Web of Linked Data W, as introduced in Definition 6.2, page 116).

Remark 7.2. We also note that an iterator-based execution of a logical plan $(B, \triangleleft)$ for a $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ cannot compute any partial solution $(E, \mu) \in \sigma(\mathcal{Q}^{B,S}, W)$ for which there does not exist a $k \in \{0, \dots, |B|\}$ such that $E = \mathrm{sub}_k^{\triangleleft}(B)$.

While Proposition 7.1 shows that link traversing iterators (implicitly) compute partial solutions, the following proposition shows that for each such iterator the overall number of these partial solutions is finite and there are not duplicates.

Proposition 7.2. *Let $(B, \triangleleft)$ be a logical plan for executing a $C_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ (over a Web of Linked Data), and let the pipeline of link traversing iterators $I_0, \dots, I_n$ be the corresponding physical plan (where $n = |B|$). Any iterator in the physical plan computes and reports a finite number of valuations during any execution of the physical plan; each of these valuations is not compatible with any other valuation that the iterator computes during the execution. (We recall that two valuations μ and μ' are* not compatible*, denoted by $\mu \not\sim \mu'$, if there exists a variable $?v \in \mathrm{dom}(\mu) \cap \mathrm{dom}(\mu')$ such that $\mu(?v) \neq \mu'(?v)$.)*

Proof. To prove Proposition 7.2 we assume (without loss of generality) a particular, arbitrary execution of the physical plan. The proof is by induction on $k \in \{0, \dots, n\}$.

Base case ($k = 0$): Root iterator I_0 reports a single valuation only (namely, the empty valuation $\mu_\emptyset$); this happens only once (during any execution of the physical plan).

Induction step ($1 \leq k \leq n$): Suppose iterator I_{k-1} reports a finite number of valuations, each of which is not compatible with any other valuation that I_{k-1} reports. If I_{k-1} reports no valuations, then I_k computes no valuations and, thus, satisfies Proposition 7.2. Hence, in the remainder of this proof we assume I_{k-1} reports at least a single valuation.

Iterator I_k performs line 9 in Listing 7.4 (cf. page 135) as many times as iterator I_{k-1} reports a valuation via its `GetNext` function. Each of these performances generates a new set Ω_{tmp}. To denote the particular version of Ω_{tmp} generated using the i-th input valuation (consumed from I_{k-1}) we write $\Omega_{\mathsf{tmp}(k,i)}$. By induction, there exists an upper bound $m \in \mathbb{N}^+$ for i (i.e., $0 < i \leq m$). Thus, to show that I_k computes a finite number of valuations only, it suffices to prove that $\Omega_{\mathsf{tmp}(k,i)}$ is finite for all $i \in \{1, \dots, m\}$: I_k precomputes $\Omega_{\mathsf{tmp}(k,i)}$ using a snapshot of the currently discovered subweb of the queried Web of Linked Data. Since the set of LD documents in any discovered subweb is finite (cf. Definition 6.4, page 118), and the set of RDF triples in any LD document is also finite, we conclude that iterator I_k may only use a finite number of RDF triples to precompute $\Omega_{\mathsf{tmp}(k,i)}$. Thus, $\Omega_{\mathsf{tmp}(k,i)}$ is guaranteed to be finite. Removing any valuation μ from the (current version of) set Ω_{tmp}, before reporting this valuation, guarantees that the number of reported valuations is finite as well (cf. lines 14 to 16 in Listing 7.4).

We now show that the valuations computed by I_k are not compatible with one another. We need to distinguish the following two cases:

1. We first focus on the difference of all valuations within any particular set $\Omega_{\mathsf{tmp}(k,i)}$ (for all $i \in \{1, \dots, m\}$): W.l.o.g., let $\Omega_{\mathsf{tmp}(k,i)}$ be such a set and let $\mu_{(k,x)} \in \Omega_{\mathsf{tmp}(k,i)}$ and $\mu_{(k,y)} \in \Omega_{\mathsf{tmp}(k,i)}$ be two of the valuations in this set; i.e., $\mu_{(k,x)} \neq \mu_{(k,y)}$. Furthermore, let tp'_k be the particular triple pattern that iterator I_k uses to precompute $\Omega_{\mathsf{tmp}(k,i)}$; i.e., $tp'_k = \mu_{(k-1,i)}[\mathrm{tp}(I_k)]$ where $\mu_{(k-1,i)}$ is the i-th input valuation (that I_k consumes from I_{k-1}). Then, there exist valuations $\mu'_{(k,x)}$ and $\mu'_{(k,y)}$ such that $\mu_{(k,x)} = \mu_{(k-1,i)} \cup \mu'_{(k,x)}$ and $\mu_{(k,y)} = \mu_{(k-1,i)} \cup \mu'_{(k,y)}$ (cf. line 9 in Listing 7.4). These two valuations, $\mu'_{(k,x)}$ and $\mu'_{(k,y)}$, are computed from different matching triples (i.e., $\mu'_{(k,x)}[tp'_k] \neq \mu'_{(k,y)}[tp'_k]$), because $\mu_{(k,x)} \neq \mu_{(k,y)}$. Therefore, $\mu'_{(k,x)} \not\sim \mu'_{(k,y)}$ and, thus, $\mu_{(k,x)} \not\sim \mu_{(k,y)}$.

2. It remains to show for any pair $i, j \in \{1, \dots, m\}$ with $i \neq j$ that every valuation in $\Omega_{\mathsf{tmp}(k,i)}$ is incompatible with every valuation in $\Omega_{\mathsf{tmp}(k,j)}$. W.l.o.g., we use an arbitrary pair $i, j \in \{1, \dots, m\}$ such that $i \neq j$, and an arbitrary valuation $\mu_{(k,x)} \in \Omega_{\mathsf{tmp}(k,i)}$. Then, valuation $\mu_{(k,x)}$ consists of all variable bindings specified by the corresponding input valuation $\mu_{(k-1,i)}$ Similarly, all valuations in $\Omega_{\mathsf{tmp}(k,j)}$ consist of all variable bindings from input valuation $\mu_{(k-1,j)}$. Since $i \neq j$, it holds by induction that $\mu_{(k-1,i)}$ is not compatible with $\mu_{(k-1,j)}$ and, thus, $\mu_{(k,x)}$ is not compatible with any valuation in $\Omega_{\mathsf{tmp}(k,j)}$.

■

Given Propositions 7.1 and 7.2, we are now ready to prove Theorem 7.1 (cf. page 137):

Proof of Theorem 7.1. Let $(B, \triangleleft)$ be an arbitrary logical selected at the begin of executing $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ by Algorithm 7.5; let the pipeline of link traversing iterators $I_0, \dots, I_n$ be the corresponding physical plan (where $n = |B|$); and let Ω be the set of all valuations that the last iterator, I_n, reports (via its `GetNext` function) during an arbitrary execution of the physical plan. W.l.o.g., we may prove Theorem 7.1 by showing (i) $\Omega \subseteq \mathcal{Q}^{B,S}(W)$ and (ii) Ω is finite.

(i) To show $\Omega \subseteq \mathcal{Q}^{B,S}(W)$ it suffices to show $\mu \in \mathcal{Q}^{B,S}(W)$ for an arbitrary valuation $\mu \in \Omega$. Let $\mu^* \in \Omega$ be such a valuation. By Proposition 7.1, $\sigma^* = \big(\mathrm{sub}_n^{\triangleleft}(B), \mu^*\big)$ is a partial solution for $\mathcal{Q}^{B,S}$ in W. This partial solution covers the whole BGP B of $\mathcal{Q}^{B,S}$ because $\mathrm{sub}_n^{\triangleleft}(B) = B$. Therefore, we may use our definition of partial solutions (cf. Definition 6.2, page 116) and our definition of $\mathrm{C}_{\mathsf{LD(M)}}$ queries (cf. Definition 6.10, page 123) to conclude $\mu^* \in \mathcal{Q}^{B,S}(W)$.

(ii) The finiteness of Ω is a direct consequence of Proposition 7.2. ■

While Propositions 7.1 and 7.2 suffice for proving Theorem 7.1, they cover only a single aspect of our execution model, namely, the computation of partial solutions. To verify that the iterator approach is an implementation of our query execution model it is also necessary to show that the approach performs open AE tasks. We recall that an *AE task* consists of a partial solution σ, a triple pattern tp, and an RDF triple t that matches tp (cf. Definition 6.8, page 122); performing such an AE task $\tau = (\sigma, t, tp)$ combines (i) generating a new partial solution by augmenting σ based on t and tp, and (ii) expanding the discovered subweb of the queried Web of Linked Data using t (cf. Section 6.3.6, page 121ff). In the remainder of this section we argue that the execution of lines 9 to 12 in the `GetNext` function of link traversing iterators (cf. Listing 7.4, page 135) presents an (implicit) performance of open AE tasks.

Example 7.6. During the query executions discussed in Examples 7.2 and 7.4 (cf. page 136 and 139, respectively), iterator I_1 consumes the empty valuation $\mu_\emptyset$ as an input valuation from its predecessor, root iterator I_0. According to Proposition 7.1 (cf. page 141), we may say that I_1 consumes partial solution $\sigma_{\mathsf{input}} = \big(\mathrm{sub}_0^{\triangleleft}(B), \mu_\emptyset\big)$, which, in this particular case, is the empty partial solution $\sigma_\emptyset$. By using this input, iterator I_1 (implicitly) computes partial solutions $\sigma_{(1,1)}$ and $\sigma_{(1,2)}$ (as discussed in Example 7.5, page 142). In terms of our execution model we interpret this computation of $\sigma_{(1,1)}$ and $\sigma_{(1,2)}$ as a computation of all augmentations of partial solution $\sigma_\emptyset$ that are possible based on all matching triples for tp_1 in the currently discovered subweb of the queried example Web W_{ex}. For instance, $\sigma_{(1,1)}$ is the $(t_{(1,1)}, tp_1)$-augmentation of $\sigma_\emptyset$ where $t_{(1,1)}$ denotes RDF triple $(\mathsf{product2}, \mathsf{producedBy}, \mathsf{producer1})$ (cf. Figure 7.2(b), page 137). Similarly, $\sigma_{(1,2)}$ is the $(t_{(1,2)}, tp_1)$-augmentation of $\sigma_\emptyset$ where $t_{(1,2)}$ denotes RDF triple $(\mathsf{product3}, \mathsf{producedBy}, \mathsf{producer1})$. Immediately after augmenting $\sigma_\emptyset$ based on these two RDF triples ($t_{(1,1)}$ and $t_{(1,2)}$), iterator I_1 uses these triples for expanding the discovered subweb $\mathfrak{D}$ to $\mathsf{EXP}\big(\mathsf{EXP}(\mathfrak{D}, t_{(1,1)}, W_{\mathsf{ex}}), t_{(1,2)}, W_{\mathsf{ex}}\big)$. Therefore, we may conclude that by executing lines 9 to 12 of its `GetNext` function, iterator I_1 (implicitly) performs two AE tasks, namely $\tau_{(1,1)} = (\sigma_\emptyset, t_{(1,1)}, tp_1)$ and $\tau_{(1,2)} = (\sigma_\emptyset, t_{(1,2)}, tp_1)$. □

The following result generalizes the conclusion from the example.

Proposition 7.3. *Let $\mathcal{Q}^{B,S}$ be a $C_{\mathsf{LD(M)}}$ query that uses a BGP B of size $n = |B|$; let $(B, \lhd)$ be a logical plan for executing $\mathcal{Q}^{B,S}$ over a Web of Linked Data W; let the pipeline of link traversing iterators $I_0, \dots, I_n$ be the corresponding physical plan; and let exec denote a particular execution of this physical plan. Furthermore, let $k \in \{1, \dots, n\}$; let μ_{input} be a valuation that iterator I_k consumes (from its predecessor I_{k-1}) during exec; let $\sigma_{\mathsf{input}} = (\mathrm{sub}^{\lhd}_{k-1}(B), \mu_{\mathsf{input}})$; and let $\mathfrak{D}_x$ denote the snapshot of the discovered subweb $\mathfrak{D}$ of W immediately after I_k consumed μ_{input}. During the particular execution of its* **`GetNext`** *function during which I_k consumes μ_{input}, I_k performs the following set $\mathcal{T}$ of open AE tasks:*

$$\mathcal{T} = \Big\{ \big(\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k)\big) \,\Big|\, t \in \mathsf{AllData}(\mathfrak{D}_x) \text{ is a matching triple for } tp'_k = \mu_{\mathsf{input}}[\mathrm{tp}(I_k)] \Big\} .$$

Proof. In addition to the symbols introduced in Proposition 7.3, let *gnexec* denote the execution of the `GetNext` function during which I_k consumes μ_{input} from I_{k-1}, and let $\mathfrak{P}_x$ be the set of partial solutions for $\mathcal{Q}^{B,S}$ in W such that the pair $st_x = (\mathfrak{P}_x, \mathfrak{D}_x)$ represents the QE state of the query execution immediately after I_k consumes μ_{input}, that is, before I_k performs line 8 during *gnexec* (cf. Listing 7.4, page 135). Similarly, let $st_y = (\mathfrak{P}_y, \mathfrak{D}_y)$ be the QE state of the query execution immediately after I_k completes line 12 (in Listing 7.4) during *gnexec*. Finally, let $tp'_k = \mu_{\mathsf{input}}[\mathrm{tp}(I_k)]$.

To prove Proposition 7.3 we have to prove the following claims:

1. Each $\tau \in \mathcal{T}$ is an AE task for $\mathcal{Q}^{B,S}$ over W.
2. Each AE task $\tau \in \mathcal{T}$ is not hidden in state st_x.
3. Each AE task $\tau \in \mathcal{T}$ is open in state st_x.
4. Each AE task $\tau \in \mathcal{T}$ is not open (anymore) in state st_y.
5. Any AE task τ' for $\mathcal{Q}^{B,S}$ over W is (still) open in state st_y if τ' is open in state st_x and $\tau' \notin \mathcal{T}$.

We notice that the fifth claim verifies that our specification of $\mathcal{T}$ is complete while the first four claims verify the soundness of this specification.

Before we prove these claims we need to specify states st_x and st_y more accurately. In particular, we need to specify $\mathfrak{P}_x$ and $\mathfrak{P}_y$. We recall that $\mathfrak{P}_x$ and $\mathfrak{P}_y$ are the sets of all those partial solutions for $\mathcal{Q}^{B,S}$ in W that have already been computed up to the point in the query execution represented by state st_x and st_y, respectively. To specify these sets we emphasize that any pipeline of link traversing iterators may (implicitly) generate partial solutions only by precomputing valuations (cf. line 9 in Listing 7.4). Hence,

$$\mathfrak{P}_x = \big\{ (\mathrm{sub}^{\lhd}_{k'}(B), \mu) \,\big|\, k' \in \{0, \dots, n\} \text{ and iterator } I_{k'} \text{ precomputes valuation } \mu \text{ before iterator } I_k \text{ consumes } \mu_{\mathsf{input}} \big\} .$$

By Proposition 7.1 (cf. page 141), $\mathfrak{P}_x \subseteq \mathcal{O}(\mathcal{Q}^{B,S}, W)$ holds with $\mathcal{O}(\mathcal{Q}^{B,S}, W)$ denoting the set of all partial solutions for $\mathcal{Q}^{B,S}$ over W (cf. Definition 6.2, page 116).

To specify $\mathfrak{P}_y$ we note that no other iterator, except for I_k, precomputes valuations while I_k executes lines 8 to 12 during *gnexec*. Thus,

$$\mathfrak{P}_y = \mathfrak{P}_x \cup \big\{ (\mathrm{sub}_k^{\lhd}(B), \mu) \,\big|\, \mu \in \Omega_{\mathsf{tmp}(k,x)} \big\},$$

where $\Omega_{\mathsf{tmp}(k,x)}$ denotes the particular version of set Ω_{tmp} that I_k precomputes during *gnexec*. Based on lines 8 and 9 in Listing 7.4 we have:

$$\Omega_{\mathsf{tmp}(k,x)} = \big\{\, \mu_{\mathsf{input}} \cup \mu' \,\big|\, \mu' \text{ is a valuation with } \mathrm{dom}(\mu') = \mathrm{vars}(tp'_k) \text{ and } \mu'[tp'_k] \in \mathsf{AllData}(\mathfrak{D}_y) \,\big\}\,.$$

We now prove the aforementioned five claims in the following order: 5, 1, 2, 4, 3.

(5.) W.l.o.g., let AE task $\tau = (\sigma, t, tp)$ with partial solution $\sigma = (E, \mu)$ be an arbitrary AE task for $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ over Web of Linked Data W such that $\tau \notin \mathcal{T}$ and τ is open in state st_x. We prove by contradiction that τ is also open in state st_y; that is, we assume τ is not open (anymore) in st_y. In this case, by Definition 6.12 (cf. page 125), it must hold that (i) τ is hidden in st_y or (ii) $st_y = \tau[st_x]$. However, since AE task τ is open in state st_x, it is not hidden in st_x (cf. Definition 6.12) and, thus, it cannot be hidden in state st_y (cf. Remark 6.3, page 124). Therefore, it must hold that $st_y = \tau[st_x]$ and, thus, the set of partial solutions $\mathfrak{P}_y$ contains the (t, tp)-augmentation of σ (cf. Definition 6.9, page 122). For this augmentation we know the following:

- By Definition 6.3 (cf. page 117), this augmentation is a pair (E^*, μ^*) with (i) E^* being a BGP that consists of triple pattern tp and all triple patterns from BGP E (i.e., $E^* = E \cup \{tp\}$) and (ii) μ^* being a valuation that extends valuation μ such that $\mathrm{dom}(\mu^*) = \mathrm{vars}(E^*)$ and $\mu^*[E^*] = \mu[E] \cup \{t\}$.
- Since AE task τ is open in QE state $st_x = (\mathfrak{P}_x, \mathfrak{D}_x)$, we also know $(E^*, \mu^*) \notin \mathfrak{P}_x$.
- Therefore, $(E^*, \mu^*) \in \mathfrak{P}_y \setminus \mathfrak{P}_x$ and, thus, $E^* = \mathrm{sub}_k^{\lhd}(B)$ and $\mu^* \in \Omega_{\mathsf{tmp}(k,x)}$.

Then, due to $E^* = E \cup \{tp\}$ we have $E = \mathrm{sub}_{k-1}^{\lhd}(B)$ (cf. Remark 7.2, page 142) and, thus, tp is the triple pattern processed by the k-th iterator; i.e., $tp = \mathrm{tp}(I_k)$. Furthermore, from $\mu^* \in \Omega_{\mathsf{tmp}(k,x)}$ we know there exist an RDF triple $t' \in \mathsf{AllData}(\mathfrak{D}_y)$ and a valuation μ' such that (i) $\mu^* = \mu_{\mathsf{input}} \cup \mu'$, (ii) $\mathrm{dom}(\mu') = \mathrm{vars}(tp'_k)$, and (iii) $t' = \mu'[tp'_k]$. Hence, RDF triple t' is a matching triple for triple pattern $tp'_k = \mu_{\mathsf{input}}[\mathrm{tp}(I_k)]$ and, thus, also for triple pattern $tp = \mathrm{tp}(I_k)$. Putting everything together, it must hold that RDF triple t' and input solution μ_{input} are the RDF triple t and valuation μ as given in AE task $\tau = (\sigma, t, tp)$ with $\sigma = (E, \mu)$; i.e., $t = t'$ and $\mu = \mu_{\mathsf{input}}$. Therefore, $\sigma = \sigma_{\mathsf{input}}$ and, thus, $\tau = (\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k))$ and $\tau \in \mathcal{T}$. This finding contradicts our premise $\tau \notin \mathcal{T}$. Hence, τ is open in state st_y.

(1.) Let $\tau = (\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k)) \in \mathcal{T}$ be an arbitrary tuple in $\mathcal{T}$. To prove that τ is an AE task for $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ over Web of Linked Data W, per Definition 6.8 (cf. page 122), we have to show the following four properties:

1. $\sigma_{\mathsf{input}} \in \mathcal{O}(\mathcal{Q}^{B,S}, W)$,

2. $t \in \mathsf{AllData}(R)$ with R being the $(S, c_{\mathsf{Match}}, B)$-reachable subweb of W,

3. $\mathrm{tp}(I_k) \in B \setminus \mathrm{sub}^{\triangleleft}_{k-1}(B)$, and

4. t is a matching triple for triple pattern $tp'_k = \mu_{\mathsf{input}}[\mathrm{tp}(I_k)]$.

Properties 1, 2, and 4 hold because $\tau \in \mathcal{T}$. Property 3 holds by definition (cf. Section 7.1.1, page 130ff).

(2.) Let $\tau = (\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k)) \in \mathcal{T}$ be an arbitrary tuple in $\mathcal{T}$. We have shown that τ is an AE task for $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ over Web of Linked Data W. To prove that τ is not hidden in state $st_x = (\mathfrak{P}_x, \mathfrak{D}_x)$, per Definition 6.11 (cf. page 124), we have to show:

1. $\sigma_{\mathsf{input}} \in \mathfrak{P}_x$ and

2. $t \in \mathsf{AllData}(\mathfrak{D}_x)$.

Again, the second property holds because $\tau \in \mathcal{T}$. Property 1 holds because iterator I_{k-1} (pre)computed valuation μ_{input} before I_k consumed μ_{input} from I_{k-1}.

(4.) Let $\tau = (\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k)) \in \mathcal{T}$ be an arbitrary tuple in $\mathcal{T}$. We have shown that τ is an AE task for $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ over Web of Linked Data W. To prove that τ is *not* open in state st_y we may either show that τ is hidden in state st_y or that $st_y = \tau[st_y]$ (cf. Definition 6.12, page 125). However, since τ is not hidden in state st_x (see above), it cannot be hidden in state st_y (cf. Remark 6.3, page 124). Hence, we show $st_y = \tau[st_y]$. By Definition 6.9 (cf. page 122), this equivalence holds if the following two statements are true:

1. $\mathsf{AUG}(\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k)) \in \mathfrak{P}_y$ (i.e., the set of partial solutions in state $st_y = (\mathfrak{P}_y, \mathfrak{D}_y)$ contains the $(t, \mathrm{tp}(I_k))$-augmentation of partial solution σ_{input}), and

2. $\mathfrak{D}_y = \mathsf{EXP}(\mathfrak{D}_y, t, W)$.

We first show the latter: By comparing the definition of $\mathcal{T}$ with line 8 in Listing 7.4 (cf. page 135), we note that RDF triple t is one of the matching triples that iterator I_k uses during the particular execution *gnexec* (as introduced in Proposition 7.3). Since $\mathfrak{D}_y$ denotes the snapshot of $\mathfrak{D}$ after iterator I_k completes line 12 during *gnexec*, it follows trivially that $\mathfrak{D}_y = \mathsf{EXP}(\mathfrak{D}_y, t, W)$.

We now show $\mathsf{AUG}(\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k)) \in \mathfrak{P}_y$: Based on Definition 6.3 (cf. page 117) we know that $\mathsf{AUG}(\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k))$ must be a pair (E^*, μ^*) with the following two elements:

- E^* is a BGP with $E^* = \mathrm{sub}^{\triangleleft}_{k-1}(B) \cup \{\mathrm{tp}(I_k)\}$ and, thus, $E^* = \mathrm{sub}^{\triangleleft}_{k}(B)$.

- μ^* is a valuation that extends input solution μ_{input} such that $\mathrm{dom}(\mu^*) = \mathrm{vars}(E^*)$ and $\mu^*[E^*] = \mu_{\mathsf{input}}[\mathrm{sub}^{\triangleleft}_{k-1}(B)] \cup \{t\}$.

Since RDF triple t is one of the matching triples that iterator I_k uses during *gnexec*, μ^* is one of the valuations that iterator I_k precomputes in line 9 of Listing 7.4 (during *gnexec*). Therefore, the pair (E^*, μ^*) is a partial solution for $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ in Web of Linked Data W (cf. Proposition 7.1, page 141) and, thus, $(E^*, \mu^*) \in \mathfrak{P}_y$.

(3.) Let $\tau = (\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k)) \in \mathcal{T}$ be an arbitrary tuple in $\mathcal{T}$. We have shown that τ is an AE task for $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ over Web of Linked Data W and τ is not hidden state st_x. To prove that τ is open in st_x it remains to show $st_x \neq \tau[st_x]$ (cf. Definition 6.12, page 125). By Definition 6.9 (cf. page 122), $st_x \neq \tau[st_x]$ holds if $\mathsf{AUG}(\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k)) \notin \mathfrak{P}_x$ (i.e., if the set of partial solutions in state $st_x = (\mathfrak{P}_x, \mathfrak{D}_x)$ does not contain the $(t, \mathrm{tp}(I_k))$-augmentation of partial solution σ_{input}). We recall that iterator I_k is the only iterator from the pipeline that may (implicitly) generate this augmentation (cf. Remark 7.1, page 142). As seen in the discussion of the previously shown Claim 4, iterator I_k precomputes the valuation for this augmentation during *gnexec*. Given that I_k precomputes a valuation only once (cf. Proposition 7.2, page 142), it is thus impossible that I_k already generated $\mathsf{AUG}(\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k))$ before *gnexec*. Therefore, $\mathsf{AUG}(\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k)) \notin \mathfrak{P}_x$ and, thus, $st_x \neq \tau[st_x]$. ■

Proposition 7.3 specifies the set of open AE tasks that a link traversing iterator performs during a single execution of its `GetNext` function. These may not be the only AE tasks that the iterator performs during the whole query execution process. Instead, a set $\mathcal{T}$ as specified in the proposition exists for each input valuation that the iterator consumes from its predecessor.

Example 7.7. We come back to Example 7.6 (cf. page 144), which focuses on open AE tasks that example iterator I_1 performs during the query executions discussed in Examples 7.2 and 7.4 (cf. page 136 and 139, respectively). Since, I_1 consumes only a single input valuation (namely, the empty valuation $\mu_\emptyset$ provided by the root iterator), the two open AE tasks introduced in Example 7.6, i.e., $\tau_{(1,1)}$ and $\tau_{(1,2)}$, are the only AE tasks that I_1 performs in any possible execution of the example plan.

We now focus on iterator I_2. During the first execution (discussed in Example 7.2), I_2 first obtains valuation $\mu_{(1,1)}$ from iterator I_1. The corresponding set of open AE tasks is $\mathcal{T}_{(2,1)} = \{\tau_{(2,1)}\}$ with $\tau_{(2,1)} = (\sigma_{(1,1)}, t_{(2,1)}, tp_2)$ because (i) iterator I_2 expands the discovered subweb $\mathfrak{D}$ using RDF triple $t_{(2,1)}$ (cf. Figure 7.2, page 137), and (ii) I_2 (implicitly) generates partial solution $\sigma_{(2,1)} = (\{tp_1, tp_2\}, \mu_{(2,1)})$ as the $(t_{(2,1)}, tp_2)$-augmentation of partial solution $\sigma_{(1,1)}$ (given in Example 7.5, page 142).

Later, during the same execution, I_2 obtains the other valuation from I_1, that is, $\mu_{(1,2)}$. The set of open AE tasks that I_2 performs based on this valuation is $\mathcal{T}_{(2,2)} = \{\tau_{(2,2)}\}$ with $\tau_{(2,2)} = (\sigma_{(1,2)}, t_{(2,2)}, tp_2)$ and $t_{(2,2)} = (\mathsf{product1}, \mathsf{oldVersionOf}, \mathsf{product3})$.

Now revisit the other possible execution of the same example plan (discussed in Example 7.4). In this case, I_2 first obtains valuation $\mu_{(1,2)}$ (instead of $\mu_{(1,1)}$) and, at this point, the discovered subweb $\mathfrak{D}$ of the queried Web consists of the LD documents d_{Pr1}, d_{p2}, and d_{p3}. Hence, RDF triple $t_{(2,2)} \in data_{\mathsf{ex}}(d_{\mathsf{p1}})$, as used by the aforementioned AE task $\tau_{(2,2)}$, has not yet been discovered and, thus, $\tau_{(2,2)}$ is still hidden. As a consequence, during this alternative execution of the example plan, the set of open AE tasks that iterator I_2 may perform based on valuation $\mu_{(1,2)}$ is empty. Therefore, to proceed with the execution, iterator I_2 discards $\mu_{(1,2)}$ and requests the next input valuation from I_1. Based on this next input, $\mu_{(1,1)}$, iterator I_2 is able to perform the same set of open AE tasks as in the execution process of Example 7.2, that is, the set $\mathcal{T}_{(2,1)} = \{\tau_{(2,1)}\}$.

After the performance of AE task $\tau_{(2,1)}$, the discovered subweb $\mathfrak{D}$ now includes the previously undiscovered RDF triple $t_{(2,2)} \in data_{\mathsf{ex}}(d_{\mathsf{p1}})$ and, thus, the previously hidden AE task $\tau_{(2,2)}$ is not hidden anymore. However, at this point during the execution process, iterator I_2 does not come back to the previous (already discarded) input valuation $\mu_{(1,2)}$. Therefore, I_2 never performs AE task $\tau_{(2,2)}$ during the execution process outlined in Example 7.4, and neither does any other iterator from the pipeline (as we see from Proposition 7.3, page 145, and Remark 7.1, page 142). As a result of not performing $\tau_{(2,2)}$, the execution process does not generate partial solution $\sigma_{(2,2)} = (B_{\mathsf{ex}}, \mu_{(2,2)})$ and, thus, the answer provided for the example query is not the complete query result. □

The example indicates that Proposition 7.3 presents a formal explanation of why some iterator-based execution processes cannot provide complete query results. To elaborate more on the reason for such an inability, suppose $(B, \lhd)$ is a logical plan for executing a $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ over a Web of Linked Data $W = (D, data, adoc)$, and I_k is an iterator in the corresponding physical plan. Then, for each solution $\mu^* \in \mathcal{Q}^{B,S}(W)$ there exists a partial solution $\sigma = (\mathrm{sub}_k^{\lhd}(B), \mu)$ such that valuations μ^* and μ are compatible (i.e., $\mu^* \sim \mu$). During any possible execution of the given plan, I_k is the only iterator that might generate σ (cf. Remark 7.1, page 142). For any other partial solution $\sigma' = (\mathrm{sub}_k^{\lhd}(B), \mu')$ that I_k might generate, it holds that $\mu^* \not\sim \mu'$ (because $\mu \not\sim \mu'$, as shown by Proposition 7.2). Therefore, for any possible execution of the given plan, generating partial solution σ (by I_k) is a necessary condition for producing solution μ^* (and, thus, for answering the query completely).

Some executions may not meet this condition: Our proof of Proposition 7.3 reveals that to generate partial solution σ, iterator I_k needs to perform an AE task $\tau = (\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k))$ with $\sigma_{\mathsf{input}} = (\mathrm{sub}_{k-1}^{\lhd}(B), \mu_{\mathsf{input}})$ such that $\sigma = \mathsf{AUG}(\sigma_{\mathsf{input}}, t, \mathrm{tp}(I_k))$ and, thus, $\mu_{\mathsf{input}} \sim \mu$. We note that τ is the only AE task that has this property because there exists only a single input valuation μ_{input} with $\mu_{\mathsf{input}} \sim \mu$ (cf. Proposition 7.2). By Proposition 7.3, there exists at most one point during any possible execution of the given plan, at which I_k may perform the AE task τ (that is, the point when I_k consumes input valuation μ_{input} from predecessor iterator I_{k-1}). However, if, at this point, the currently discovered subweb $\mathfrak{D}$ of the queried Web does *not* comprise an LD document $d \in D$ with $t \in data(d)$, then AE task τ is still hidden (cf. Definition 6.11, page 124) and I_k misses the opportunity to perform τ and, thus, to generate σ. As discussed in the Example 7.7, the query execution process outlined in Example 7.4 (cf. page 139) represents such a case: When example iterator I_2 would have to perform AE task $\tau_{(2,2)} = (\sigma_{(1,2)}, t_{(2,2)}, tp_2)$, this task is still hidden because the corresponding RDF triple $t_{(2,2)}$ has not yet been discovered. An even more extreme example presents the execution in Example 7.3 (cf. page 138) that uses the alternative logical plan.

Example 7.8. Example 7.3 considers the alternative logical plan for executing the example query. The link traversing iterator I'_1 in the corresponding (alternative) physical plan is responsible for triple pattern $\mathrm{tp}(I'_1) = (?previous, \mathsf{oldVersionOf}, ?product)$. During the first execution of its `GetNext` function, I'_1 consumes the empty valuation $\mu_\emptyset$ as an input valuation from root iterator I'_0. At this point, the currently discovered subweb $\mathfrak{D}$ of the queried example Web contains a single LD document, namely d_{Pr1} (cf. Example 7.3). Since $data_{\mathsf{ex}}(d_{\mathsf{Pr1}})$ does not contain a matching triple for triple pattern $\mu_\emptyset[\mathrm{tp}(I'_1)]$, the corresponding set $\mathcal{T}$ of open AE tasks that I'_1 "performs" is empty (cf. Proposition 7.3). Thus, during the execution process in Example 7.3, both link traversing iterators, I'_1 and I'_2, do not perform any open AE task. □

While the set of open AE task that a link traversing iterator performs may be empty (for some or even for all executions of its `GetNext` function), it is impossible, on the other hand, that such an iterator performs an infinite number of AE tasks.

Corollary 7.1. *The overall number of open AE tasks that link traversing iterators perform during any possible execution of Algorithm 7.5 is finite.*

Proof. This proposition is a direct consequence of Propositions 7.2 and 7.3 (cf. page 142 and 145, respectively): From Proposition 7.2 it follows that, for each link traversing iterator, there always only exists a finite number of sets $\mathcal{T}$ as specified in Proposition 7.3. Each of these sets is finite because any discovered subweb of a queried Web of Linked Data contains a finite number of RDF triples only (cf. Definition 6.4, page 118). ■

Corollary 7.1 verifies that Algorithm 7.5 always terminates. We emphasize that such a guarantee does not exist for all implementations of our traversal-based execution model. In particular, implementation approaches that guarantee completeness of query results, cannot guarantee termination of any possible execution process (given that some $\mathrm{C}_{\mathsf{LD(M)}}$ queries are not finitely computable by an LD machine). Furthermore, we note that Corollary 7.1 is in line with Theorem 7.1 (cf. page 137): Since any pipeline of link traversing iterators may only perform a finite number of AE tasks, it cannot report an infinite number of query solutions, even if the expected query result (as defined by the query semantics) is infinite.

Our results of this section show that a pipeline of link traversing iterators performs open AE tasks and, thus, implements our traversal-based query execution model. However, this implementation does not guarantee completeness; that is, by executing a $\mathrm{C}_{\mathsf{LD(M)}}$ query with this implementation approach we may not obtain the complete query result (but the result that we obtain is guaranteed to be sound). Furthermore, we identified the following properties of this implementation approach:

- Based on different logical plans the approach may report different subsets of the complete query result.
- Even for the same physical plan we may observe different execution processes and obtain different subsets of the complete query result.
- Any possible query execution process is guaranteed to terminate.

In the following, we investigate these properties experimentally.

7.3. Experimental Analysis

In this section we analyze the performance of the iterator-based implementation approach in practice. Primarily, we are interested in the issue of incompleteness. That is, we focus on the following questions:

Question 7.1. What is the percentage of reachable documents discovered by the iterator-based implementation approach?

Question 7.2. To what degree are answers provided by the approach complete (i.e., what is the ratio of reported solutions to all solutions in corresponding query results)?

Question 7.3. Does there exist a correlation between the degree of completeness and the percentage of reachable documents discovered by the approach?

Question 7.4. Does the distribution of data over a queried Web of Linked Data affect the degree of completeness? If it does, what (link graph) structure of a queried Web is beneficial for the approach?

Question 7.5. What is the practical impact of the property that different logical plans for a query may not be semantically equivalent?

Question 7.6. What is the practical impact of the property that different executions of the same plan may return different results?

Question 7.7. Which logical plans for a query enable us to achieve the highest degree of completeness (in comparison to all logical plans for the query)?

In addition to these questions we test the following hypothesis:

Hypothesis 7.1. Overall query execution time is dominated by the time needed for retrieving data from the queried Web of Linked Data.

For our experimental analysis we focus on $\mathrm{C}_{\mathsf{LD(M)}}$ queries whose set of seed URIs consists of all subject-position and object-position URIs mentioned in the BGP of the query. Formally, for any $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ that we consider it holds that

$$S = \{u \in \{s,o\} \cap \mathcal{U} \,|\, (s,p,o) \in B\}.$$

To ensure reproducibility we conducted most of the experiments for our analysis in a simulation environment. Although the execution of Linked Data queries over the WWW is the main use case for the concepts discussed in this dissertation, the WWW is unsuitable as an experimental environment. Reasons for this unsuitability are nondeterministic timeouts, temporarily unresponsive Web servers, unpredictable effects of Web caches (including intermediate proxy servers), and other, non-repeatable behavior. Furthermore, certain datasets published as Linked Data on the WWW change frequently; as a consequence, experiments based on queries that discover and use such data become non-reproducible quickly. Therefore, we set up a controlled simulation environment that implements multiple Webs of Linked Data for our experiments.

The only experiment during which we queried Linked Data on the WWW is related to Hypothesis 7.1 (for which results would otherwise be easily manipulable by parameters configured for the simulation environment). This WWW-based experiment also provides us with a general understanding of typical query execution times, response times, and size of retrieved data (number of documents and RDF triples) as can be expected for Linked Data queries over the WWW.

This section proceeds as follows: First, we briefly introduce the query execution system used for our experiments. Then, we describe and discuss the WWW-based experiment. Afterwards, we focus on the simulation-based experiments.

7.3.1. Our Query Execution System

As a basis for our experiments we developed a query execution system called *SQUIN* [73]. The program is written in Java and is available as Free Software at the project homepage [68]. SQUIN consists of the following main components:

dataset This component provides a physical main memory representation of a query-local dataset and, thus, enables the query system to (temporarily) store and to access all data retrieved from the Web during a query execution. The basis of this component is a main memory data structure that stores RDF triples from all discovered documents in a single index. This index consists of six hash tables to efficiently support all possible types of triple pattern based lookups. In earlier work we provide a comprehensive discussion of this data structure [77].

lookup This component processes URI lookups using the HTTP protocol. Due to a multi-threaded implementation, multiple lookup tasks may run in parallel. For the experiments we allow SQUIN to use a maximum of 20 lookup threads. Furthermore, we configured a timeout of 30 seconds for each lookup. In addition to the actual lookups, the lookup component manages (i) the status of currently running lookups and (ii) the results of completed lookups. By using such information, the component avoids looking up the same URI multiple times.

engine This component provides the actual query engine that implements a traversal-based query execution using the iterator-based implementation approach. The engine makes use of the aforementioned components to find matching triples and to initiate URI lookups during the execution of a query. In its primary mode of operation the engine clears all data structures (e.g., the query-local dataset, the index of completed URI lookups) before it starts executing a given query.

In the version used for our experiments, SQUIN contains no component for selecting a logical execution plan. Instead, the order in which triple patterns of a query appear in the given serialization of the query are used as the logical plan.

For the experiments we instrumented SQUIN to measure (i) the number of returned solutions for each query, (ii) query execution time, (iii) the number of retrieved documents, (iv) the number of URI lookups that timed out, and (v) the average URI lookup time. We then set up a benchmark system that integrates SQUIN. This system runs in a virtual machine on a server in the university network. Hence, it has comparably fast access to the WWW. The virtual machine has a single (virtual) processor and 4 GB of main memory. The operating system on the virtual machine is openSUSE Linux 11.2 with Sun Java 1.6.0.

7.3.2. WWW-Based Experiment

In our first experiment we executed particular $\mathrm{C}_{\mathsf{LD(M)}}$ queries over the WWW to test Hypothesis 7.1. In the following we describe this experiment and discuss the results.

Queries

For this experiment we used a mix of 18 queries. These queries, denoted by WQ1 to WQ18, can be found in the Appendix (cf. Section D.1, page 211ff). Eleven of these queries have been proposed as Linked Data queries for the FedBench benchmark suite [141]. These *"queries vary in a broad range of characteristics, such as number of sources involved, number of query results, and query structure"* [141]. However, we had to slightly adjust five of these queries (WQ6, WQ7, WQ8, WQ9, and WQ10) because their original versions use terms from outdated vocabularies. These adjustments do not change the intent of the queries or their structural properties.

In addition to these eleven FedBench queries, we used seven queries that are updated versions of queries used for experiments in our earlier work [72, 79]. These seven queries are designed such that each of the respective reachable subwebs covers data from a larger number of data providers than the reachable subwebs of the FedBench queries. Hence, these seven queries add more diversity to the query mix.

Overall the query mix covers a variety of structural properties. The BGPs in these queries consist of two to eight triple patterns and contain between two to seven different query variables. Some of these BGPs are "star-shaped" (i.e., one variable is contained in all triple patterns of the BGP), others are "path-shaped" (i.e., every variable is contained in at most two triple patterns), and a third group presents mixtures of star-shaped and path-shaped BGPs. Table 7.1 characterizes all 18 test queries w.r.t. these properties.

Procedure

For the experiment we executed the 18 queries sequentially (i.e., one after the other). Such a sequential execution avoids measuring artifacts of concurrent executions. To exclude possible interference between subsequent query executions, we use SQUIN in its primary mode of operation as described in Section 7.3.1 (cf. page 152f), that is, each query execution within the sequence starts with an initially empty query-local dataset. Hereafter, we refer to these executions of the test queries as *data-retrieving executions*.

Query	**Number of seed URIs**	**Number of triple patterns**	**Number of variables**	**Primary structure**
WQ1	1	3	3	path-shaped
WQ2	1	3	3	path-shaped
WQ3	1	4	4	mixed
WQ4	1	5	4	path-shaped
WQ5	2	3	2	star-shaped
WQ6	1	3	3	path-shaped
WQ7	2	3	3	path-shaped
WQ8	1	5	5	mixed
WQ9	1	3	2	path-shaped
WQ10	1	3	3	path-shaped
WQ11	1	5	5	star-shaped
WQ12	2	6	5	mixed
WQ13	1	5	5	path-shaped
WQ14	1	6	5	mixed
WQ15	1	2	2	path-shaped
WQ16	1	3	3	path-shaped
WQ17	1	8	7	mixed
WQ18	1	6	5	mixed

Table 7.1.: Structural properties of the queries used for the WWW-based experiment.

To minimize the potential of impacting the experiment by unexpected network traffic we performed five of the aforementioned sequential runs and combine the measurements by calculating the arithmetic mean for each query, respectively. By this procedure we obtain the following primary measurements for each test query:

- the average number of documents retrieved during data-retrieving executions of the query,
- the average number of solutions returned for the query during data-retrieving executions, and
- the average time for completing the data-retrieving executions of the query.

Furthermore, for each test query we also recorded the minimum and maximum of (i) the number of retrieved documents, (ii) the number of returned solutions, and (iii) the query execution time as measured during the five runs (of data-retrieving executions).

To test Hypothesis 7.1 we also need the data retrieval time, that is, the fraction of the query execution time that SQUIN spends on data retrieval. However, this fraction is difficult to determine during data-retrieving executions because data retrieval and query-local data processing are deeply interwoven in SQUIN (as suggested by our query execution model); furthermore, due to a multi-threaded implementation of data retrieval, SQUIN usually performs multiple URI lookups in parallel (cf. Section 7.3.1, page 152f). Consequently, simply summing up the runtime of all URI lookups would not be an accurate approach for measuring the fraction of query execution time spent on data retrieval (i.e., data retrieval time). Therefore, we applied the following cache-based method to measure the data retrieval time for each test query.

We executed each test query twice: First, as for the aforementioned data-retrieving executions, we executed the query using SQUIN in its primary mode of operation (i.e., starting the query execution with an empty query-local dataset). Hence, during this execution, SQUIN populates the (initially empty) query-local dataset as usual. After this first execution we kept the populated query-local dataset and reused it as the initial query-local dataset for a second execution of the same test query. However, for this second execution we deactivated SQUIN's data retrieval functionality and, thus, used SQUIN as if it was a standard SPARQL query engine. That is, we evaluated the test query over the fixed dataset that we obtained from the first execution. As a result, we may use the difference between the query execution time of this second execution—hereafter called *cache-based execution*—and the average query execution time of the data-retrieving executions as a measure of data retrieval time. Formally:

Definition 7.2 (Average Data Retrieval Time). Let q be a test query; let t_{overall} be the average query execution time measured for the data-retrieving executions of q; and let t_{local} be the query execution time measured for the cache-based execution of q. The *average data retrieval time* for test query q is $t_{\mathsf{retrieval}} := t_{\mathsf{overall}} - t_{\mathsf{local}}$. □

Arguably, this method of measuring data retrieval time is not completely accurate. By starting the cache-based executions with query-local datasets that are already populated completely, our query engine may compute additional intermediate solutions (that are not computed based on an initially empty, continuously augmented query-local dataset) [71]. Thus, for the cache-based executions, the amount of query-local data processing may be greater than what it actually is in the data-retrieving executions. Therefore, our method may underestimate the actual data retrieval times. However, underestimation does not invalidate a verification of Hypothesis 7.1.

Measurements

We conducted the experiment from September 17, 2012 to September 18, 2012. Thus, the measurements that we report in the following, reflect the situation of Linked Data on the WWW during that time (and may be different for other points in time).

The chart in Figure 7.4 depicts the average number of solutions returned for each of the 18 test queries during the five primary, data-retrieving runs. Figure 7.5 reports the corresponding number of retrieved documents. The range bar laid over each of the main bars in these two charts denotes the minimum and maximum values that contribute to the average represented by the main bar (for the exact minimum and maximum values we refer to Table D.1 in the appendix, cf. page 215).

These range bars indicate that the measurements for queries WQ5, WQ8, WQ17, and WQ18 varied *significantly* for the five runs, whereas the measurements for the other

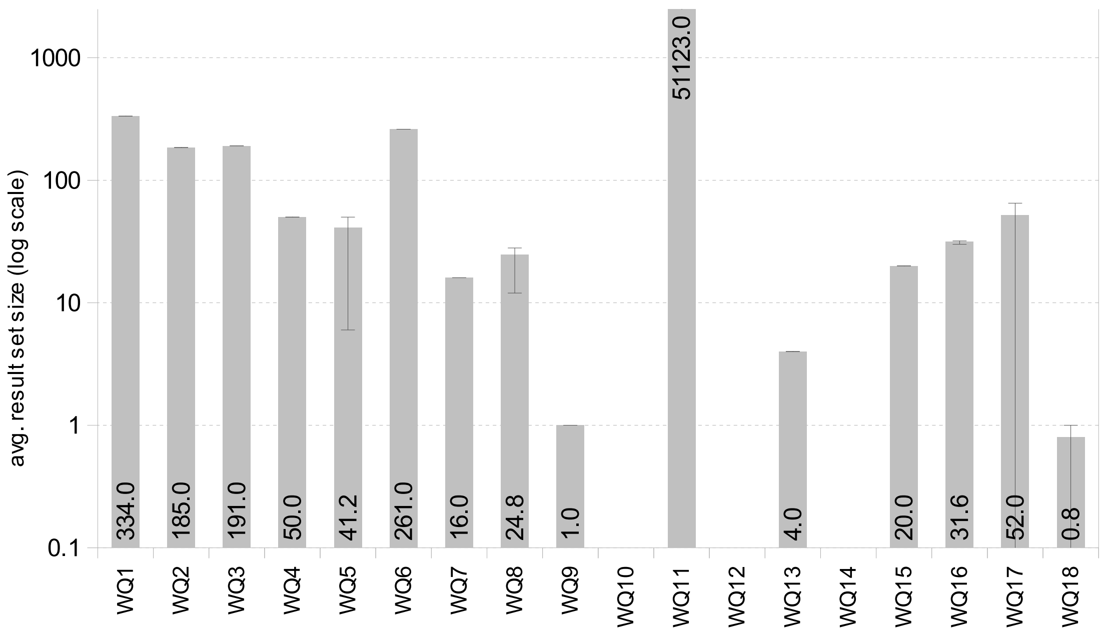


Figure 7.4.: Average number of solutions returned for each of the 18 test queries in the WWW-based experiment.

queries have been consistent. A closer inspection of the statistics recorded during the experiment reveals that for query WQ5, WQ8, WQ17, and WQ18, some atypical URI lookup timeouts occurred during *one* of the five executions, respectively.

For instance, there are four executions of query WQ8 during which SQUIN recorded 48 lookup timeouts and retrieved 331 LD documents, respectively; during the other execution of query WQ8, SQUIN observed an atypical number of 160 timeouts. As a result, during this atypical execution, SQUIN retrieved only 59 of the 331 LD documents.[1] The smaller number of retrieved documents also has an effect on the number of solutions that SQUIN returned during the atypical execution of query WQ8 (see the lower bound of the corresponding range bar in Figure 7.4).

An even more extreme example is query WQ17 for which the lookup of the seed URI of the query timed out during one of the five executions. In this case, the lack of seed data made the traversal-based discovery of further data impossible. As a result, during this atypical execution of query WQ17, SQUIN did not retrieve a single document and returned no solutions for the query.

While Figures 7.4 and 7.5 show measurements for the data-retrieving runs, corresponding numbers for the cache-based run follow from these measurements: We ensured that the particular (data-retrieving) executions based on which we populated the query-local

[1]We explain the disparity in the differences between the typical and the atypical number of timeouts (48 vs. 160) and between the typical and the atypical number of retrieved documents (331 vs. 59) as follows: Some of the documents missed due to the additional timeouts during the atypical execution, have enabled SQUIN to discover further documents during the four typical executions.

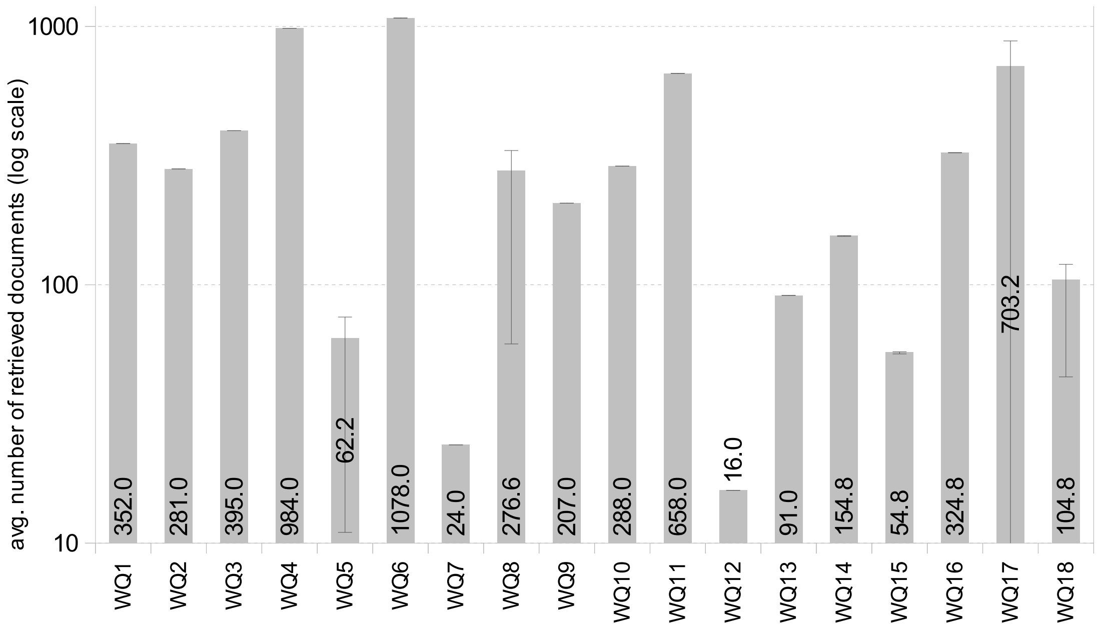


Figure 7.5.: Average number of documents retrieved during executions of the 18 test queries in the WWW-based experiment.

datasets for the cache-based executions are not atypical. Hence, for each test query, the cache-based run (re)used a pre-populated dataset that consisted of the data from all documents whose retrieval contributed to the corresponding measurement in the rightmost column of Table D.1(b) (cf. page 215). The number of solutions returned in the cache-based executions is consistent with the numbers reported for the typical data-retrieving executions (that is, the numbers in the rightmost column of Table D.1(a), page 215).

Figure 7.6 reports query execution times. In particular, the dark gray, hatched bars in the chart represent the query execution times measured during the cache-based run. The light gray bars represent the average query execution times of the primary, data-retrieving executions; range bars, again, denote the minimum and maximum values that contribute to the average (Table D.2 in the appendix lists the exact minimum and maximum values; cf. page 216).

Result

Based on Figure 7.6 we observe that for each test query the average execution time measured during the data-retrieving runs is significantly larger than the time required for the cache-based execution. More precisely, these times differ by two (for query WQ17) to five (for queries WQ6, WQ7, WQ8, WQ10, WQ15, and WQ16) orders of magnitude. As discussed in the context of Definition 7.2 (cf. page 155), these differences approximate the net times that SQUIN required for retrieving data during the traversal-based executions of the test queries. Thus, the measurements show that the overall (traversal-based)

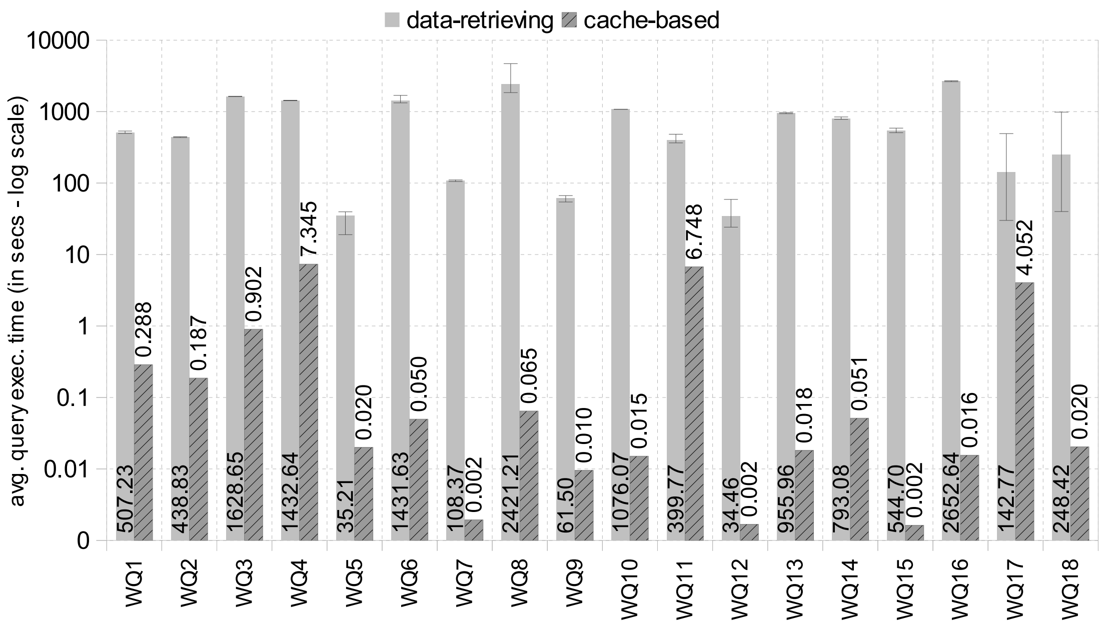


Figure 7.6.: Comparison of overall execution times for each of the 18 test queries in the WWW-based experiment.

query execution time for the test queries is dominated by the data retrieval time, which verifies Hypothesis 7.1.

7.3.3. Simulation Based Experiments

To answer Questions 7.1 to 7.7 we set up a controlled environment that allows us to simulate multiple Webs of Linked Data. Based on this environment, we conducted a series of experiments during which we executed all possible query plans for particular $C_{\mathsf{LD(M)}}$ queries over different simulated Webs. In the following we first describe the experimental setup, that is, the simulation environment, the simulated Webs of Linked Data, and the test queries; thereafter, we discuss the experiments and their results.

Simulation Environment

For the simulation environment, we developed a Java servlet called *WODSim* (which is available at the SQUIN Web site [68]). This servlet simulates a (finite) Web of Linked Data by answering all URI lookup requests that Web clients may send to potential Web servers in the simulated Web. The basis of such a simulated Web are (gzip-compressed) RDF documents stored in a particular structure of subdirectories in the file system of the servlet container that deploys WODSim. More precisely, each of these documents contains the data for an LD document of the simulated Web. When WODSim serves such document (on behalf of a simulated Web server), it transforms the data of the document by prefixing any URI mentioned in the document with the URL at which the WODSim instance is deployed on the WWW. This transformation ensures that all URIs that appear in the simulated Web refer back to the WODSim instance hosting the simulation. Hence, systems built for accessing Linked Data on the WWW (such as SQUIN) may traverse the simulated Web without crossing over to the wider WWW.

For the experiments, we deployed WODSim using an Apache Tomcat Web server (version 6.0.33) on a second virtual machine. This virtual machine has the same configuration and runs on the same server as the virtual machine that hosts our SQUIN-based benchmark system (cf. Section 7.3.1, page 152f).

Simulated Webs of Linked Data

The overall goal of our experiments is to investigate the effects of using link traversing iterators for queries over a Web of Linked Data. However, the observability of these effects may be highly dependent on how such a queried Web is structured and how data is distributed. Therefore, we used multiple Webs of Linked Data for all of our experiments. In fact, some of the experiments (in particular, those related to Questions 7.3 and 7.4) inherently require executing the same query plans in different Webs. To be able to meaningfully compare such executions over different Webs, we generated each of these Webs using the same base dataset.

This base dataset consists of synthetic RDF data created with the data generator that is part of the Berlin SPARQL Benchmark (BSBM) suite [19]. The data describes entities in a fictitious distributed e-commerce scenario, including different producers and vendors, products, product offers, and reviews from multiple reviewing sites. Figure 7.7 illustrates the RDF vocabulary used for these descriptions.

To obtain the particular BSBM dataset based on which we generated our test Webs, we executed the BSBM data generator using a scaling factor of 200. The resulting base dataset consists of 75,150 RDF triples and describes 7,329 entities (namely: 21 product types, 999 product features, 5 producers, 200 products, 2 vendors, 4,000 offers, and 1 rating site with 101 reviewers and 2,000 reviews). Each of these entities is identified by a single, unique URI. We denote the set of these URIs by U_{test} (i.e., $U_{\mathsf{test}} \subseteq \mathcal{U}$ and $|U_{\mathsf{test}}| = 7,329$), and the base dataset by G_{test}.

Each test Web that we generated from the base dataset consists of 7,329 LD documents, each of which is authoritative for a different URI $u \in U_{\mathsf{test}}$. To distribute the base dataset G_{test} over these documents, we partitioned G_{test} into 7,329 (potentially overlapping) subsets. While the particular partitioning process differed for the different test Webs (as described in the following), we ensured that every RDF triple in the subset for the LD document generated for URI $u \in U_{\mathsf{test}}$ has u as subject or as object (as encouraged by the Linked Data principles [14]). As a result, every test Web is a Web of Linked Data $W_{\mathsf{test}} = (D, data, adoc)$ that has the following six properties:

1. $|D| = 7,329$.
2. For each URI $u \in U_{\mathsf{test}}$, $adoc(u) \in D$.
3. For each pair of distinct URIs $u, u' \in U_{\mathsf{test}}$ (i.e., $u \neq u'$), $adoc(u) \neq adoc(u')$.

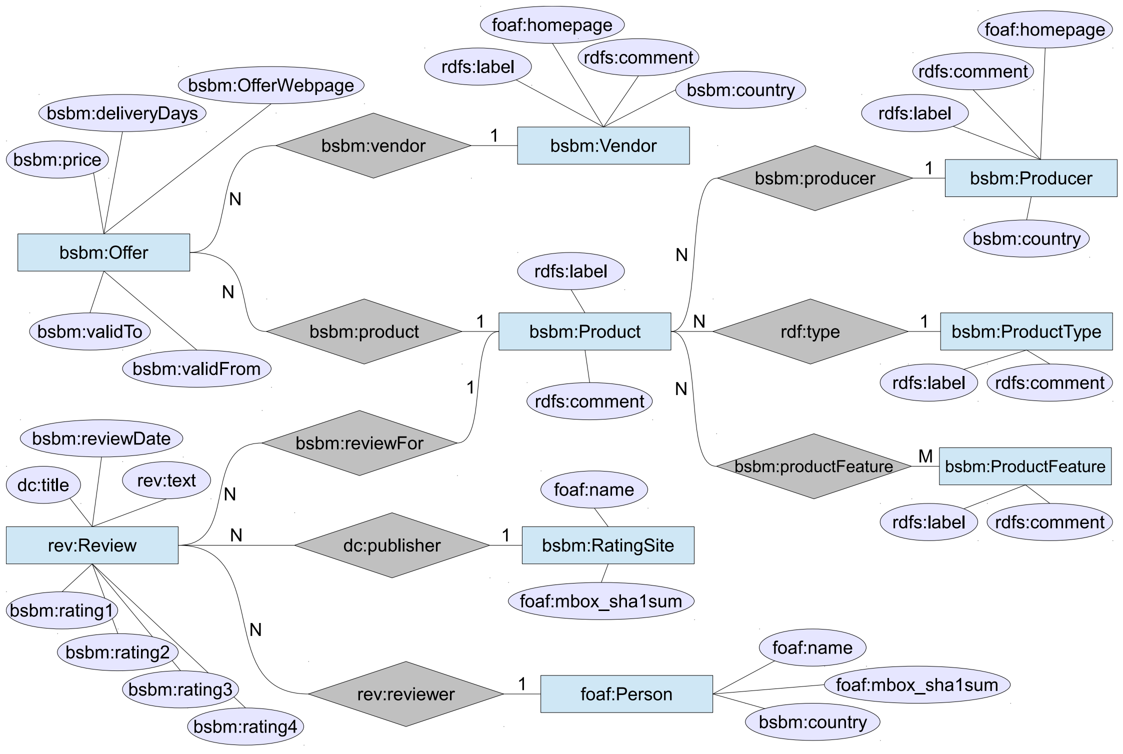


Figure 7.7.: An entity-relationship model that illustrates the RDF vocabulary used for BSBM datasets. Entity sets represent classes; relationship sets represent properties. Attributes represent additional properties whose ranges are literals. Key attributes (representing the URI of each entity) are omitted.

4. For each URI $u \notin U_{\mathsf{test}}$, $adoc(u) = \perp$.

5. $\mathsf{AllData}(W_{\mathsf{test}}) = G_{\mathsf{test}}$.

6. For every URI $u \in U_{\mathsf{test}}$ and RDF triple $(s, p, o) \in data(adoc(u))$, $u \in \{s, o\}$.

Property 5 shows that all RDF triples in our test Webs come from the base dataset and, conversely, each RDF triple from the base dataset is available in every test Web.

Since none of the RDF triples generated by the BSBM data generator contains a blank node, the base dataset consists of two types of RDF triples only: RDF triples from the set $\mathcal{U} \times \mathcal{U} \times \mathcal{U}$, henceforth referred to as *URI-only triples*, and RDF triples from the set $\mathcal{U} \times \mathcal{U} \times \mathcal{L}$, henceforth referred to as *literal triples*.

To ensure Properties 5 and 6 for every test Web, we always added each of the literal triples to the data of the LD document generated for the subject URI of that triple; i.e., for each literal triple $t = (s, p, o) \in G_{\mathsf{test}}$, we ensured that $t \in data(adoc(s))$ holds for every test Web $W_{\mathsf{test}} = (D, data, adoc)$.

For any URI-only triple $t \in G_{\mathsf{test}}$, it was possible to either add t only to the LD document for the subject of t, add t only to the LD document for the object of t, or add t to both of these documents. The first of these three options establishes a *subject-based data link* pointing to the LD document for the object of t, the second option establishes an *object-based data link* to the LD document for the subject of t, and the third option establishes both the subject-based and the object-based data link. It is easy to see that choosing among these three options (where the choice may differ for each URI-only triple) may have a significant impact on the structure of the resulting test Web.

We applied a random-based approach to choose among these three options for any URI-only triple. This approach is based on the following two metrics (which we shall use to characterize the structure of our test Webs): Informally, the *bidirectional interlinkage* (or simply *b-interlinkage*) of a test Web is the percentage of all URI-only triples that establish both a subject-based and an object-based data link; the *non-bidirectional, subject-based interlinkage* (or simply *nbsb-interlinkage*) of a test Web is the percentage of the remaining URI-only triples that establish a subject-based data link (and not an object-based data link). Formally, we define these metrics for an arbitrary Web of Linked Data (not just our test Webs) as follows:

Definition 7.3 (b-interlinkage and nbsb-interlinkage). Let $W = (D, data, adoc)$ be a Web of Linked Data and let T_{or}, T_{and}, and T_{subj} be the following sets of (URI-only) RDF triples in $\mathsf{AllData}(W)$:

$$T_{\mathsf{or}} = \Big\{ (s,p,o) \in \mathcal{U} \times \mathcal{U} \times \mathcal{U} \,\Big|\, (s,p,o) \in data\big(adoc(s)\big) \text{ or } (s,p,o) \in data\big(adoc(o)\big) \Big\},$$

$$T_{\mathsf{and}} = \Big\{ (s,p,o) \in \mathcal{U} \times \mathcal{U} \times \mathcal{U} \,\Big|\, (s,p,o) \in data\big(adoc(s)\big) \text{ and } (s,p,o) \in data\big(adoc(o)\big) \Big\},$$

$$T_{\mathsf{subj}} = \Big\{ (s,p,o) \in \mathcal{U} \times \mathcal{U} \times \mathcal{U} \,\Big|\, (s,p,o) \in data\big(adoc(s)\big) \text{ and } (s,p,o) \notin data\big(adoc(o)\big) \Big\}.$$

Then, the *b-interlinkage* of W, denoted by $\mathrm{bi}(W)$, and the *nbsb-interlinkage* of W, denoted by $\mathrm{nbsbi}(W)$, are defined as follows:

$$\mathrm{bi}(W) := 100\% \cdot \frac{|T_{\mathsf{and}}|}{|T_{\mathsf{or}}|} \qquad \text{and} \qquad \mathrm{nbsbi}(W) := 100\% \cdot \frac{|T_{\mathsf{subj}}|}{|T_{\mathsf{or}} \setminus T_{\mathsf{and}}|} . \qquad \square$$

We emphasize that there exists exactly one test Web with a b-interlinkage of 0% and an nbsb-interlinkage of 100%. In this test Web, hereafter denoted by $W^{0,100}_{\mathsf{test}}$, any base dataset triple is contained only in the LD document generated for the subject of that triple. Similarly, there exists exactly one test Web, $W^{0,0}_{\mathsf{test}}$, with both a b-interlinkage of 0% and an nbsb-interlinkage of 0%, and there exists exactly one test Web, W^{100}_{test}, with 100% b-interlinkage (nbsb-interlinkage is irrelevant for the latter).

Any other pair of b-interlinkage and nbsb-interlinkage does not uniquely determine a particular test Web. Instead, for any such pair (b, n), there exist multiple test Webs, each of which has a b-interlinkage of b and an nbsb-interlinkage of n. These test Webs have link graphs that are very similar to each other. When we partitioned the base dataset G_{test} to generate such a test Web $W^{b,n}_{\mathsf{test}} = (D, \ data, adoc)$, we used a random-based approach. That is, for each URI-only triple $(s, p, o) \in G_{\mathsf{test}}$, we *randomly* selected one of the aforementioned three options (i.e., adding the triple to LD document $adoc(s)$, to LD document $adoc(o)$, or to both) such that the resulting test Web has the given pair of b-interlinkage b and nbsb-interlinkage n. Hence, each of the three options had a certain probability of being (randomly) selected; essentially, these probabilities are a function of the given interlinkage values b and n (i.e., we used b as the probability for choosing to add a URI-only triple $(s, p, o) \in G_{\mathsf{test}}$ to both LD documents $d_s = adoc(s)$ and $d_o = adoc(o)$, and we used n as the conditional probability for choosing to add the triple to d_s if it has not been added to both d_s and d_o).

For most of our evaluation questions, we are interested in test Webs whose pair of interlinkage values is most representative of the interlinkage values of (real) Linked Data on the WWW. To determine such representative interlinkage values we analyzed a comprehensive corpus of real Linked Data that has been crawled for the Billion Triple Challenge 2011 [64]. This corpus consists of about 7.9 million LD documents; the overall number of RDF triples distributed over these documents is 2.15 billion (where the overall number of unique RDF triples is 1.97 billion) and the number of URIs mentioned in these triples is 103 million. For this corpus of real Linked Data, we determined a b-interlinkage of 62% and an nbsb-interlinkage of 47%.

Given this pair of values, we generated ten different test Webs using the random-based approach as outlined before. Thus, each of these ten Webs has a b-interlinkage of 62% and an nbsb-interlinkage of 47% (and the aforementioned six properties). Furthermore, although they differ slightly, the link graphs of these ten Webs are very similar to each other. For instance, for an arbitrarily specified path, the prior probability that this path exists in the respective link graph is the same for each of these test Webs (and different from the corresponding prior probability that we may find for any test Web that has another pair of interlinkage values). Hereafter, we denote these ten test Webs by $W_{\mathsf{test}}^{62,47,1}$, $W_{\mathsf{test}}^{62,47,2}$, $\dots$, $W_{\mathsf{test}}^{62,47,10}$.

In addition to these, structurally very similar test Webs, we are interested in a set of test Webs that covers a wide variety of link graphs (in order to study Question 7.4). For this purpose, we randomly generated another ten test Webs based on ten *different* pairs of b-interlinkage and nbsb-interlinkage. We denote these test Webs by $W_{\mathsf{test}}^{0,33}$, $W_{\mathsf{test}}^{0,66}$, $W_{\mathsf{test}}^{33,0}$, $W_{\mathsf{test}}^{33,33}$, $W_{\mathsf{test}}^{33,66}$, $W_{\mathsf{test}}^{33,100}$, $W_{\mathsf{test}}^{66,0}$, $W_{\mathsf{test}}^{66,33}$, $W_{\mathsf{test}}^{66,66}$, and $W_{\mathsf{test}}^{66,100}$, where the superscript identifies the corresponding pair of b-interlinkage and nbsb-interlinkage. Finally, we complemented this set of test Webs for Question 7.4 by adding the aforementioned test Webs $W_{\mathsf{test}}^{0,0}$, $W_{\mathsf{test}}^{0,100}$, and W_{test}^{100}.

We materialized each of the 23 generated test Webs into a separate structure of sub-directories in the file system of our WODSim-based simulation server. Hence, switching from one of the test Webs to another is a matter of restarting the WODSim servlet with a particular configuration parameter that points to the corresponding directory.

Queries and Query Plans

As a basis for the simulation-based experiments we used six $\mathrm{C}_{\mathsf{LD(M)}}$ queries, SQ1 to SQ6, which can be found in the Appendix (cf. Section D.3, page 216f). These queries differ w.r.t. their structural properties, as summarized in Table 7.2.

Query	Number of seed URIs	Number of triple patterns	Number of variables	Primary structure
SQ1	2	3	2	path-shaped
SQ2	1	3	3	path-shaped
SQ3	1	3	3	star-shaped
SQ4	1	4	3	path-shaped
SQ5	1	4	3	mixed
SQ6	2	4	3	path-shaped

Table 7.2.: Structural properties of the test queries used for the simulation-based experiments.

In addition to the structural diversity, these queries also differ w.r.t. the types of paths that establish the query-specific reachable subwebs of our test Webs. For instance, for query SQ2, any such path needs to start with a subject-based data link from the authoritative document for Review110 to a document for the product associated with the review; from there, paths may alternate between subject-based data links from documents for products to documents for product features and object-based data links from documents for product features to documents for products. In contrast, for query SQ3, such paths can only have a length of 1 and need to consist of an object-based data link from the authoritative document for Product128 to a document for a review about the product.

For each of the six test queries we generated all possible query plans, each of which presents a different permutation of the set of triple patterns as given for the respective query. As a result, we obtained an overall number of 90 query plans (six for the three queries with three triple patterns and 24 for the three queries with four triple patterns).

Procedure

For the simulation-based experiments we used SQUIN to perform different query execution runs. Each of these runs consists of executing the 90 query plans over a particular test Web. As for the WWW-based experiment, we avoided measuring artifacts of concurrent query executions by executing all query plans sequentially, one after another, and we excluded possible interference between subsequent query executions by using SQUIN in its primary mode of operation (i.e., for each query plan, SQUIN starts with an initially empty query-local dataset). Another aspect that might limit the comparability of our measurements is the nondeterministic behavior of link traversing iterators (as discussed in the context of Example 7.4, page 139). Since this type of nondeterminism is irrelevant for testing Hypothesis 7.1, we deliberately ignored it for the WWW-based experiments. However, for the simulation-based experiments, we need to take the nondeterministic behavior of link traversing iterators into consideration.

To investigate whether this nondeterministic behavior has a practical impact (cf. Question 7.6), we added two deterministic implementations of a link traversing iterator to SQUIN. These implementations represent any set of precomputed solutions (i.e., the set

Ω_{tmp} in Listing 7.4, page 135) as a list and return such solutions in the order in which they appear in the list. The particular (total) order used for the list is based on how SQUIN represents valuations internally and is irrelevant for our discussion. While both deterministic implementations use this (artificial) order, one of them always returns the precomputed solutions starting from the begin of the list, whereas the other implementation always starts from the end. Consequently, we refer to query execution runs for which we use these (deterministic) implementations as *ascending runs* and *descending runs*, respectively. *Nondeterministic runs*, in contrast, use the standard, nondeterministic implementation.

We performed these runs for all of the aforementioned 23 test Webs. During the execution of each of the 90 query plans in such a run, we measured the number of documents retrieved and the number of solutions returned. We do not report query execution times here because, as per Hypothesis 7.1 (which we have verified by the WWW-based experiment), measuring query execution times basically means measuring the response time of our WODSim-based simulation environment. Furthermore, we emphasize that due to the reliability of this environment, SQUIN did not observe any URI lookup failures or timeouts during the simulation-based experiments. Thus, all measurements presented in the following section are error-free.

For some of our evaluation questions (in particular, for Questions 7.1, 7.2, 7.3, and 7.4) we need to know what the (complete) query result of our 6 test queries over each of the test Webs is and how many of the 7,329 LD documents of each test Web are reachable in the context of each test query. However, these numbers cannot be measured during ascending, descending, or nondeterministic runs (because link traversing iterators cannot guarantee completeness). Hence, to obtain these numbers we performed an additional *completeness run* for each test Web. During such a run we executed each of the 6 test queries using a two-phase algorithm that is similar to the algorithm implemented by the 2P machine (cf. Algorithm 4.1, page 88). That is, given a test query, the algorithm first retrieves all reachable LD documents by traversing data links that qualify according to reachability criterion c_{Match} (recall that our test queries are $\mathrm{C}_{\mathsf{LD(M)}}$ queries; hence, c_{Match}-semantics applies). Second, the algorithm evaluates the BGP of the test query over all data from the retrieved documents and, thus, computes the (complete) query result. Since the algorithm is a straightforward adaptation of Algorithm 4.1, we omit proving its properties (soundness, completeness, and termination for any Web of Linked Data that is finite such as our test Webs); instead, we refer to the formal discussion of Algorithm 4.1 (in particular, Lemmas 4.2, 4.3, and 4.4; cf. pages 88 to 89).

Measurements

The measurements that we obtained using test Webs $W_{\mathsf{test}}^{62,47,1}$ to $W_{\mathsf{test}}^{62,47,10}$ are very similar to each other. Therefore, this section presents detailed measurements for test Web $W_{\mathsf{test}}^{62,47,1}$ only. Thereafter, we discuss our observations and refer to the measurements obtained using the other test Webs when necessary.

The charts in Figures 7.8 and 7.9 depict these measurements (cf. pages 166 and 167). More precisely, Figures 7.8(a), 7.8(c), 7.8(e),7.8(g), 7.9(a), and 7.9(c) report the number of solutions returned by executing the query plans for test query SQ1, SQ2, SQ3, SQ4, SQ5, and SQ6 over test Web $W_{\mathsf{test}}^{62,47,1}$, respectively. The corresponding numbers of retrieved documents can be seen in Figures 7.8(b), 7.8(d), 7.8(f),7.8(h), 7.9(b), and 7.9(d). The x-axises in these charts represent the query plans. Labels on these axises (given in parentheses) indicate a specific structural property of the plans that is relevant for our discussion below; we shall introduce this property in the context of this discussion.

The bars in these charts represent the respective measurements obtained during the ascending, descending, and nondeterministic run; the corresponding measurements of the completeness run are represented as a dotted line that stretches across the bars (because, for each test query, the cardinality of the complete query result is independent of the query plan used, and so is the number of reachable documents).

For instance, from Figures 7.8(a) and 7.8(b) we see that the complete query result for test query SQ1 (over test Web $W_{\mathsf{test}}^{62,47,1}$) consists of 563 solutions and the corresponding reachable subweb of the test Web comprises 3,834 documents. During the ascending run, the first plan for query SQ1 discovered and retrieved 2,201 of the 3,834 reachable documents and returned 482 of the 563 solutions (see the leftmost bars in the figures).

Discussion of Questions 7.1 and 7.2

The first and most striking observation from our measurements in Figures 7.8 and 7.9 is that the lack of semantic equivalence between different logical plans for a query (as shown theoretically in Section 7.2.1, page 137ff) has a *significant* practical impact. For any of the six test queries, the degree of completeness of the computed query results differs drastically between the different execution plans, and so does the number of discovered documents. For all test queries except SQ4, there exist plans that return empty query results. In contrast, other plans achieve degrees of result completeness of up to 86%, 86%, 100%, 94%, 50%, and 78% for query SQ1 to query SQ6, respectively.

Table 7.3 (on page 168) shows that we observed similar differences (as well as similar minimum and maximum degrees of result completeness) in the other test Webs generated with 62% b-interlinkage and 47% nbsb-interlinkage. While we shall discuss reasons for these differences below, these differences show that it is impossible to provide a general answer for Questions 7.1 and 7.2 (other than that the percentages may range from 0% to 100%). Therefore, in the remainder of this section we focus on the other evaluation questions and discuss them one after another.

Discussion of Question 7.3

To investigate whether link traversing iterator based query plans exhibit a correlation between (i) the degree of result completeness and (ii) the percentage of reachable documents discovered (Question 7.3), we computed these two numbers for each of the 90 query plans for every ascending run (i.e., over any of our test Webs); we then plotted the resulting pairs of numbers as points in scatter charts.

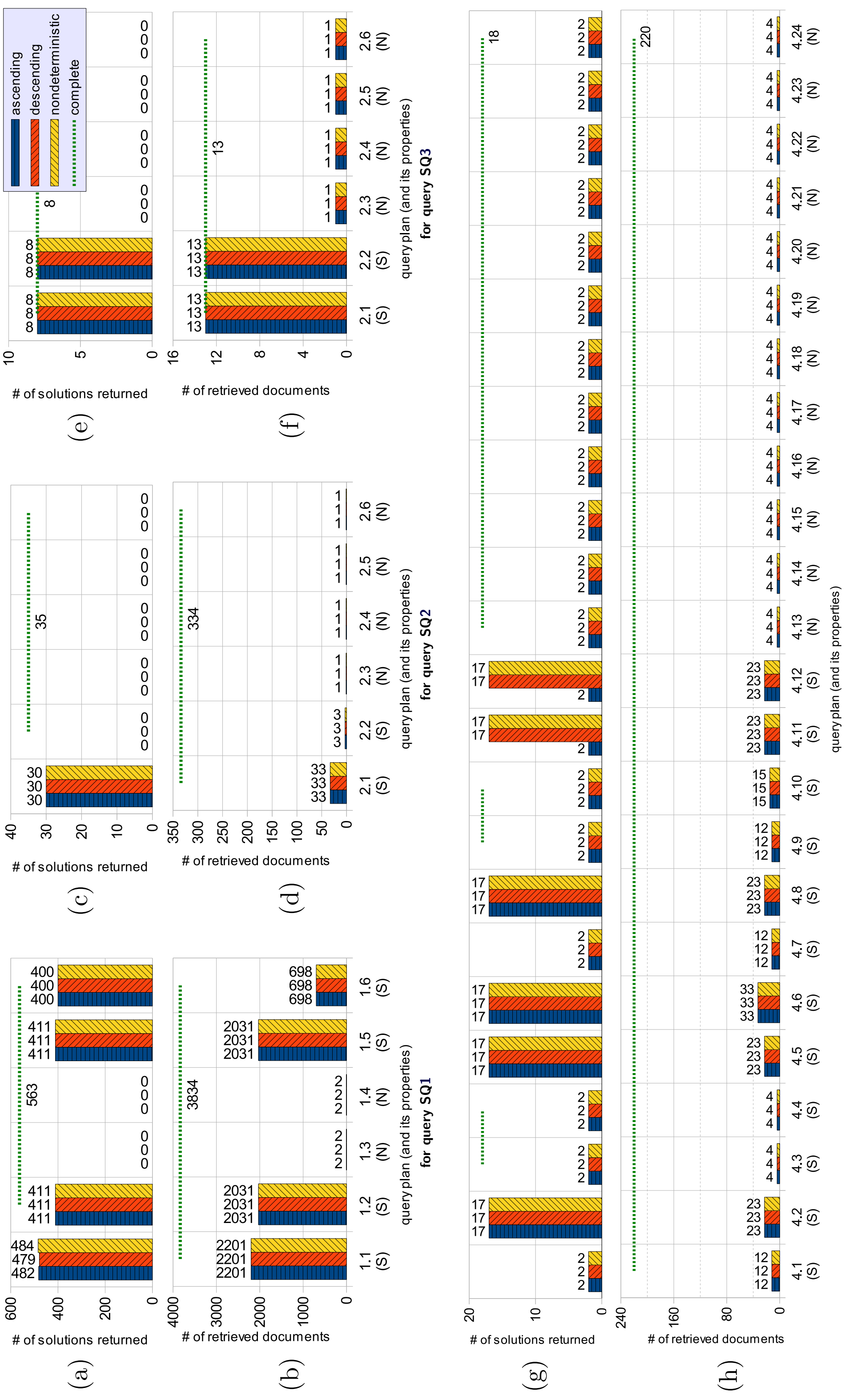


Figure 7.8.: Measurements for querying test Web $W_{\text{test}}^{62,47,1}$ using all possible query plans for test queries SQ1 to SQ4.

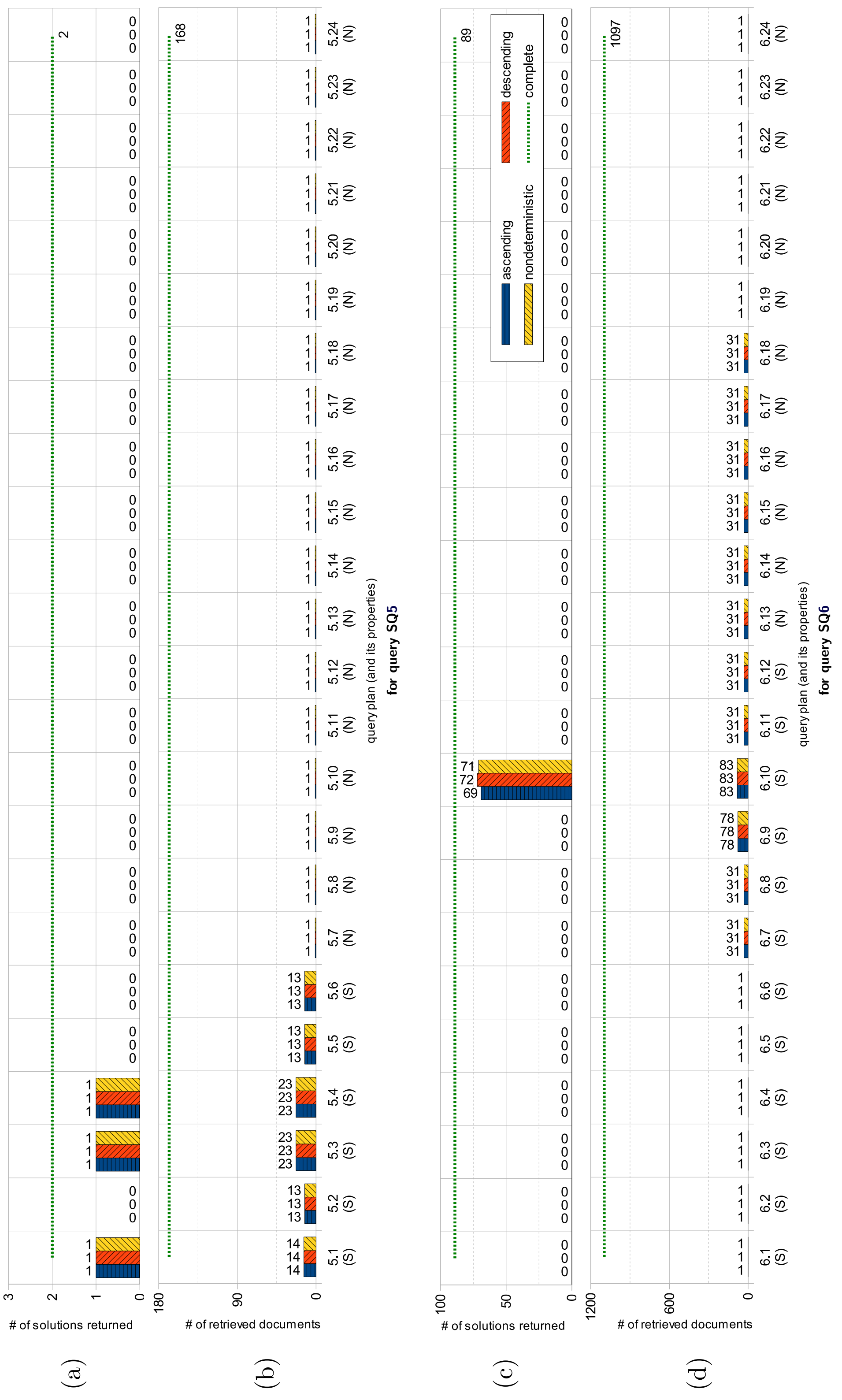


Figure 7.9.: Measurements for querying test Web $W_{\text{test}}^{62,47,1}$ using all possible query plans for test queries SQ5 and SQ6.

Test	Query SQ1			Query SQ2			Query SQ3			Query SQ4			Query SQ5			Query SQ6		
Web	min	max	cmpl	min	max	cmpl	min	max	cmpl	min	max	cmpl	min	max	cmpl	min	max	cmpl
$W_{\mathsf{test}}^{62,47,1}$	0	0.86	563	0	0.86	35	0	1	8	0.11	0.94	18	0	0.5	2	0	0.78	90
$W_{\mathsf{test}}^{62,47,2}$	0	0.82	554	0	0.89	36	0	1	8	1	1	1	0	1	2	0.36	0.89	89
$W_{\mathsf{test}}^{62,47,3}$	0	0.82	560	0	0.88	34	0	1	9	0.11	0.95	19	0	1	1	0	0.68	88
$W_{\mathsf{test}}^{62,47,4}$	0	0.86	559	0	0.82	34	0	1	10	0.05	0.81	21	0	1	1	0.32	0.81	88
$W_{\mathsf{test}}^{62,47,5}$	0	0.83	561	0	0.83	35	0	1	8	0.11	0.95	19	0	1	1	0.33	0.91	89
$W_{\mathsf{test}}^{62,47,6}$	0	0.87	566	0	0.79	33	0	1	9	1	1	1	0	0.5	2	0.30	0.78	87
$W_{\mathsf{test}}^{62,47,7}$	0	0.84	555	0	0.78	36	0	1	8	0.11	0.89	18	0	1	1	0.31	0.83	90
$W_{\mathsf{test}}^{62,47,8}$	0	0.82	561	0	0.77	35	0	1	10	0.05	0.74	19	0	1	2	0.30	0.88	89
$W_{\mathsf{test}}^{62,47,9}$	0	0.84	563	0	0.80	35	0	1	10	1	1	1	0	1	3	0.32	0.80	89
$W_{\mathsf{test}}^{62,47,10}$	0	0.81	562	n/a	n/a	0	0	1	10	0.11	0.90	19	n/a	n/a	0	0.33	0.75	89

Table 7.3.: Minimum and maximum degree of result completeness achieved by executing the different possible query plans for test queries SQ1 to SQ6 over different test Webs (during the ascending runs). To put these numbers into perspective, the *cmpl* columns list the cardinality of the complete query results for each test query, respectively.

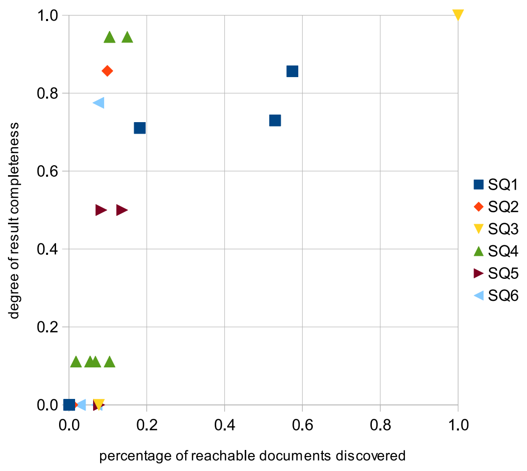


Figure 7.10.: Correlation between the percentage of reachable documents discovered by any query plan and the degree of result completeness achieved by that plan (for the ascending run over test Web $W_{\text{test}}^{62,47,1}$); any point represents a particular plan for one of the six test queries.

Figure 7.10 depicts such a chart for the ascending run over test Web $W_{\text{test}}^{62,47,1}$ (i.e., the points in this chart are based on the measurements presented in Figures 7.8 and 7.9). For instance, the first two plans for query SQ3 are represented by two (coincident) points in the top-right corner of the chart (both plans returned the complete query result and retrieved all reachable documents in test Web $W_{\text{test}}^{62,47,1}$). We emphasize that the analogous charts generated for the other test Webs are very similar to Figure 7.10.

The distribution of points in each of these charts does *not* suggest a general, mutual dependency between result completeness and the percentage of discovered documents. However, we notice that for each test query, the query plans that return the most solutions (i.e., achieve the highest degree of completeness) are among the plans that discover a comparably high number of reachable documents. On the other hand, our measurements refute the logical converse of this observation. That is, a statement such as “those query plans for a query that discover a comparably high number of reachable documents, achieve the highest (or, at least, a comparably high) degree of completeness” is not supported by our measurements. As counterexamples, consider the measurements for query plans 4.1, 4.7, 4.9, and 4.10 in Figures 7.8(g) and 7.8(h). Another, more extreme counterexample are the measurements for query plan 6.9 in Figures 7.9(c) and 7.9(d).

Discussion of Question 7.4

We now turn to Question 7.4, which focuses on the dependency between the distribution of data over a queried Web of Linked Data and the degree of result completeness achieved by link traversing iterators. To discuss this question, we focus on a single query execution plan per test query. In particular, for each test query, we focus on a plan that achieved the highest degree of completeness during the ascending run over test Web $W_{\text{test}}^{62,47,1}$ (as reported in Figures 7.8 and 7.9). For query SQ1 this is plan 1.1, for SQ2 it is plan 2.1, and for SQ3, SQ4, SQ5, and SQ6 it is 3.1, 4.2, 5.1, and 6.10, respectively. For ease of reference, we used these plans for the particular serialization of our six test queries as given in the Appendix (cf. Section D.3, page 216f); that is, the order in which triple patterns of the test queries appear in the given serializations is the order prescribed by the aforementioned six plans (i.e., 1.1, 2.1, 3.1, 4.2, 5.1, and 6.10).

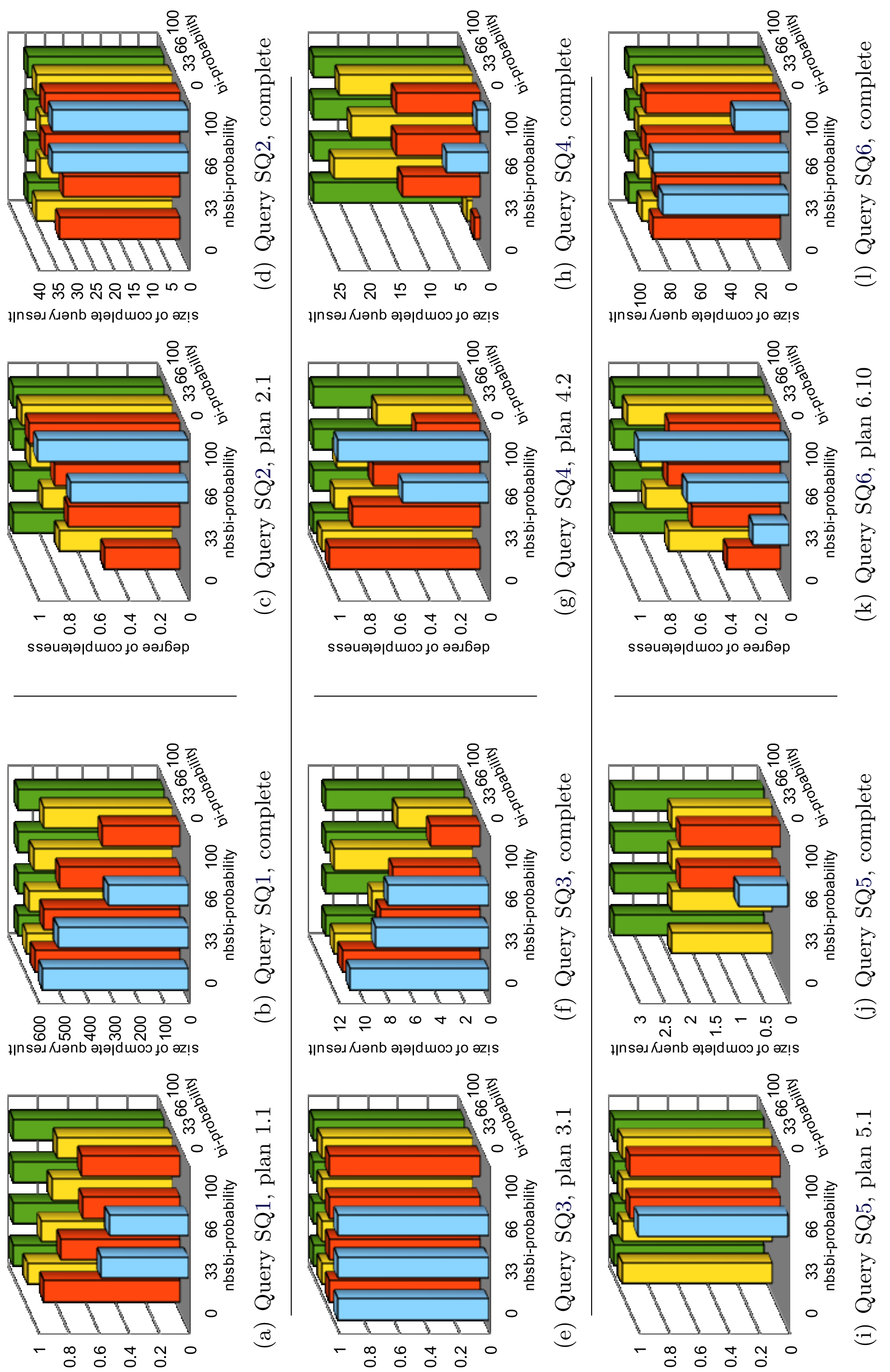
The charts in Figure 7.11 present the degree of completeness achieved by these six query plans during ascending runs over different test Webs. In particular, these are the 13 test Webs that we generated using b-interlinkage and nbsb-interlinkage of 0%, 33%, 66%, and 100%, respectively. Each bar in these charts represents the measurement for one of these test Webs (since nbsb-interlinkage is irrelevant for a b-interlinkage of 100%, the four bars in the back of each chart represent the same measurement, respectively). To put these measurements into perspective, Figure 7.11 also includes charts that report the cardinalities of the corresponding complete query results (as measured by the corresponding completeness runs). These charts reveal that the complete results of our test queries over some test Webs are empty (under c_{Match}-semantics). For instance, SQ1 over $W_{\text{test}}^{0,100}$ has an empty result (cf. Figure 7.11(b)). We explain the latter by the lack of object-based data links in $W_{\text{test}}^{0,100}$—without such links the (SQ1-specific) set of reachable documents in $W_{\text{test}}^{0,100}$ consists only of the seed documents for SQ1, because all triple patterns of SQ1 have a variable in the subject position. Apparently, in cases like this (i.e., emptiness of the complete query result), discussing the degree of result completeness achieved by iterator-based query execution plans is meaningless.

As a first, general observation from Figure 7.11, we notice that for plans 1.1, 2.1, 4.2, and 6.10, the degree of result completeness depends on the b-interlinkage of the queried test Web: With an increasing b-interlinkage (i.e., a larger number of bidirectional data links), the result completeness achieved by these plans also increases in almost all cases.

We also observe that in test Web W_{test}^{100} (which has a b-interlinkage of 100%), all six plans achieve the maximum degree of completeness. We emphasize that this is not the case for all 90 query plans. Instead, there exist plans that returned an empty set of query solutions for *every* test Web. Most of these plans have the following property: None of the respective seed documents contains a single matching triple for the first triple pattern in such a plan (such matching triples may be available in other reachable documents but those cannot be discovered by executing the plan in question). For instance, plan 3.3 prescribes the following order for the three triple patterns in query SQ3:

```
?review  bsbm:rating1  ?rating .
?review  bsbm:reviewFor  <http:// ... /Product128> .
?review  dc:title  ?reviewTitle .
```

The document that is the authoritative document for `Product128` is the seed document for this query. However, in none of our test Webs, this document contains triples that match the first triple pattern of plan 3.3 (i.e., the `bsbm:rating1` triple pattern). Therefore, plan 3.3 cannot return a single solution for query SQ3 in any test Web. This is in stark contrast to plan 3.1 (as we can see in Figure 7.11(e)).



(a) Query SQ1, plan 1.1
(b) Query SQ1, complete
(c) Query SQ2, plan 2.1
(d) Query SQ2, complete
(e) Query SQ3, plan 3.1
(f) Query SQ3, complete
(g) Query SQ4, plan 4.2
(h) Query SQ4, complete
(i) Query SQ5, plan 5.1
(j) Query SQ5, complete
(k) Query SQ6, plan 6.10
(l) Query SQ6, complete

Figure 7.11.: Degree of result completeness achieved by particular query plans during ascending runs over different test Webs (generated based on different pairs of b-interlinkage and nbsb-interlinkage). To put these numbers into perspective, every other diagram reports the cardinalities of the corresponding complete query results.

Query plan 3.1 (as well as plan 5.1) achieves the maximum degree of result completeness in any relevant test Web (that is, any test Web in which the complete result for query SQ3 is not empty). For plan 3.1 this observation is easily explained: The (SQ3-specific) reachable subweb of any test Web contains (i) the aforementioned seed document (i.e., the authoritative document for Product128) and (ii) the authoritative documents for reviews about Product128 linked to by an object-based data link in the seed document. These data links can only be established by (URI-only) RDF triples that match the bsbm:reviewFor triple pattern given in query SQ3. Since plan 3.1 assigns this pattern to the first link traversing iterator, this iterator discovers and retrieves the *complete* set of reachable documents even before the remaining iterators aim to find matching triples for the other triple patterns of query SQ3.

Plan 5.1 also achieves the maximum degree of result completeness in any relevant test Web (cf. Figure 7.11(i)). However, in contrast to plan 3.1, plan 5.1 does not guarantee an exhaustive discovery of the respective reachable subweb of a queried test Web (as the measurements in Figure 7.9(b) show). For instance, there may exist a review that is not about Product128 but that is associated with a reviewer who also reviewed Product128. While the authoritative document for such a review may be reachable in some test Web, an execution of plan 5.1 cannot discover this document because the plan ignores object-based data links from reviewers to reviews (the variable ?review is always already bound when the third iterator evaluates the corresponding rev:reviewer triple pattern). However, this limitation has no negative impact on result completeness because such review data (that is reachable but cannot be discovered by executing plan 5.1), is not relevant for query SQ5 (which is only concerned with reviews about Product128). What is important is that, in any test Web, data about reviews (and about their reviewers) that is both reachable and relevant for query SQ5, can always be discovered by plan 5.1; the paths to such data in the link graphs of our test Webs corresponds to the order of triple patterns as prescribed by the plan (i.e., object-based data links from Product128 to its reviews and subject-based data links from these reviews to their reviewers).

The latter finding (i.e., the finding that the correspondence between paths to relevant data and triple pattern order in a query plan may have an impact on whether the plan discovers that data) also explains the following observation: For the other four plans covered in Figure 7.11 (i.e., plans 1.1, 2.1, 4.2, and 6.10), the degree of completeness also depends on the nbsb-interlinkage of the queried test Web (not only on the b-interlinkage).

We explain this observation for query plan 2.1, for which the expected (complete) query results appeared to be almost the same for most of the test Webs (in fact, the same holds for the respective sets of reachable documents). We first note that Review110 (mentioned in query SQ2) is about Product128, which is associated with 36 different product features in our base dataset G_{test}. Consequently, in any test Web in which the LD documents for these 36 product features are reachable (based on query SQ2), the complete result for query SQ2 consists of 36 solutions. Such a document (for one of the 36 product features) might be reachable due to the existence of a path (in the link graph of the queried Web) that corresponds to the triple patterns in query plan 2.1, that is, (i) the path is of length two, (ii) the first edge on the path is a subject-based data link from the (seed) document for Review110 to the document for Product128, and (iii) the second edge is a subject-based data link from the document for Product128 to the document for the product feature. However, such document may also be reachable in a test Web whose link graph does not contain such a path: We note that the corresponding product feature is also associated with other products, which in turn have other features. Then, by traversing recursively between documents for products and documents for product features, we may eventually discover those features of Product128 that are not explicitly mentioned in the authoritative document for Product128 (which explains why even in test Webs with a lower nbsb-interlinkage the size of the complete result for query SQ2 is close to 36; cf. Figure 7.11(d)). However, an iterator-based execution of plan 2.1 cannot traverse recursively between product documents and feature documents. Instead, it relies on the availability of the aforementioned paths of length two to discover relevant documents of product features. Recall that these paths consist of two subject-based data links. Hence, their existence is more likely in test Webs that have a high nbsb-interlinkage (or a high b-interlinkage). Therefore, plan 2.1 misses an increasing number of query solutions when we go to test Webs with a lower nbsb-interlinkage (cf. Figure 7.11(c)).

In contrast, other query plans miss an increasing number of query solutions when the queried Webs have a *higher* nbsb-interlinkage. Examples for such plans are plan 1.1 and 4.2 (cf. Figures 7.11(a) and 7.11(g)). These plans rely on the availability of specific object-based data links (rather than subject-based). Clearly, query plans may also partly rely on both specific subject-based data links and specific object-based data links (which is more likely for queries with a larger number of triple patterns).

From these observations we draw the following conclusions for Question 7.4.

1. Unsurprisingly, the distribution of data over a queried Web of Linked Data clearly affects the degree of completeness.
2. Webs of Linked Data in which the seed document(s) for a given query contain RDF triples that match at least one of the triple patterns in the query are more beneficial for the iterator-based implementation approach.
3. Webs of Linked Data with a high b-interlinkage (i.e., many bidirectional data links) are more beneficial for the iterator-based implementation approach.
4. Webs of Linked Data with a low b-interlinkage may also be beneficial for the approach; however, the link graph of such a Web must include the traversal paths that query plans rely on.

Discussion of Questions 7.5 and 7.6

We have already answered Question 7.5 when we discussed Questions 7.1 and 7.2 (cf. page 165): The property that different logical plans for a query may not be semantically equivalent has a significant impact in practice: For *none* of our test queries, we have encountered a test Web for which all possible query plans perform equally well w.r.t. result completeness (with the exception of test Webs for which the complete query result is empty). In contrast, there exist cases where some plans achieve the maximum degree of

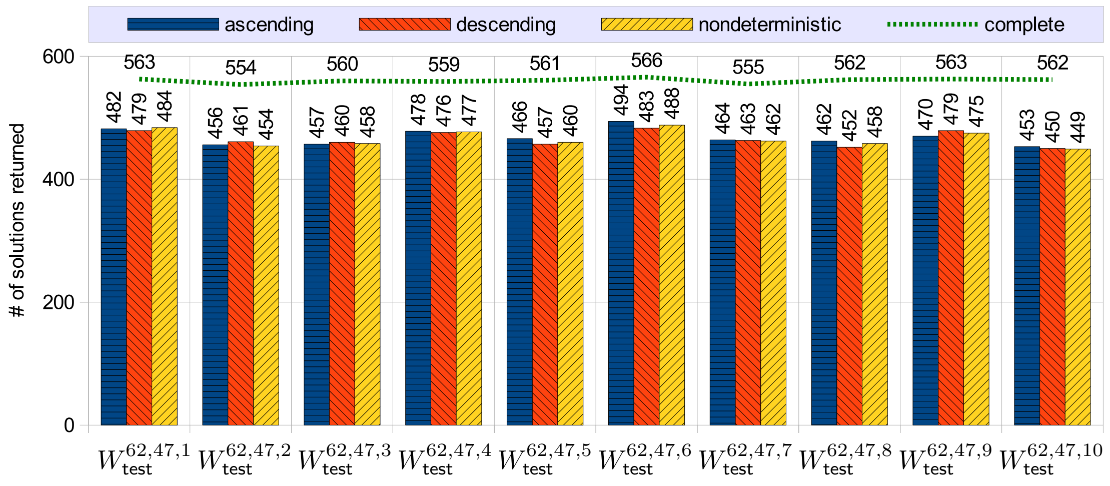


Figure 7.12.: Measurements for querying different test Webs using query SQ1, plan 1.1.

completeness (i.e., answer the query completely), while other plans cannot even provide a single query solution (Figure 7.8(e) illustrates such an example for query SQ3).

Related to the previous question, Question 7.6 focuses on the practical impact of the property that even the same query plan may result in different answers (as discussed in the context of Example 7.4, page 139). While our measurements show that such an impact exists, it is considerably less significant than the practical impact of the lack of semantic equivalence between different logical plans. As an example, consider query plan 1.1 for query SQ1: The chart in Figure 7.12 illustrates the number of query solutions that this plan reported during the ascending, descending, and nondeterministic runs over test Webs $W_{\text{test}}^{62,47,1}$ to $W_{\text{test}}^{62,47,10}$. We notice that, for each of these test Webs, the number of query solutions returned by plan 1.1 differed across the respective ascending, descending, and nondeterministic run. However, these differences are small compared to differences between the number of solutions that different plans for query SQ1 returned during, e.g., the ascending run over test Web $W_{\text{test}}^{62,47,1}$ (cf. Figure 7.8(a))

Discussion of Question 7.7

It remains to discuss which of the possible logical plans for a $\mathrm{C}_{\mathsf{LD(M)}}$ query guarantee the highest degree of completeness (Question 7.7). Presumably, properties of such plans that might influence the degree of completeness a plan may achieve, are endless. We focus on a single structural property. In particular, we note that some query plans are *seed-based*, that is, their first triple pattern contains one of the seed URIs. This observation leads to the following definition.

Definition 7.4 (Seed-Based Plan). Let $(B, \triangleleft)$ be a logical plan for the BGP B of a $\mathrm{C}_{\mathsf{LD(M)}}$ query $\mathcal{Q}^{B,S}$ (as per Definition 7.1, page 130); let tp_1 be a triple pattern such that $tp_1 \in B$ and $tp_1 \triangleleft tp$ for all $tp \in B \setminus \{tp_1\}$. The logical plan $(B, \triangleleft)$ is *seed-based* if there exists a URI $u \in S$ such that $u \in \text{uris}(tp_1)$. $\square$

Before we discuss whether our measurements confirm that such a property has an impact on the degree of completeness, we argue why such an impact might exist.

During any execution of a pipeline of link traversing iterators $I_0, I_1, \dots, I_n$, iterator I_1 (which is responsible for the first triple pattern) is the first iterator that generates valuations based on matching triples in the query-local dataset. The remaining link traversing iterators augment these valuations by adding bindings for their query variables. Hence, it is crucial to select a triple pattern for I_1 such that there exists a high likelihood that the early snapshot of the query-local dataset as used by I_1 already contains matching triples for the selected, first triple pattern. We recall that this particular snapshot of the query-local dataset consists of the data from the seed documents only. Since such a seed document is the authoritative document for a seed URI, data in such a document (primarily) includes RDF triples that contain the respective seed URI. Hence, those triples are more likely to match a triple pattern that also contains the seed URI rather than an arbitrary triple pattern (that does not contain the seed URI). Therefore, it seems reasonable to select a triple pattern that contains one of the seed URIs as the first triple pattern and, thus, to prefer seed-based query plans over plans that are not seed-based.

To evaluate the suitability of seed-based plans based on our measurements, we identified all seed-based plans among our 90 test plans. The annotations on the x-axises in Figures 7.8 and 7.9 indicate whether a plan is seed-based or not, denoted by S and N, respectively. Then, the measurements presented in these figures show that the query plans that achieved the highest degree of completeness for their respective query over test Web $W_{\mathsf{test}}^{62,47,1}$ are seed-based plans. Our measurements for the other test Webs confirm this observation. On the other hand, not all seed-based plans achieved a comparably high degree of completeness. Instead, some seed-based plans did not even return a single solution for their query over test Web $W_{\mathsf{test}}^{62,47,1}$. Apparently, none of the non-seed-based plans returned any solution either. We emphasize that the latter is not a general issue. In some test Webs, non-seed-based plans achieved a degree of completeness greater than zero. However, in these cases all corresponding seed-based plans achieved at least the same degree of completeness (or higher). Consequently, even if arbitrarily selecting a seed-based plan may not guarantee the highest possible degree of completeness, it is a better choice than selecting a non-seed-based plan.

We summarize the findings of our simulation-based experiments as follows:

- An estimate on the percentage of reachable documents discovered by the (iterator-based) approach or about the degree of result completeness achieved by that approach cannot be given. Even for the same query over the same Web of Linked Data, these numbers may range from 0% to 100% depending on the selected query plan. (For the discovered documents the minimum is always slightly above 0% because any plan "discovers" at least all seed documents.)

- Our experiments do not suggest a general correlation between the degree of completeness and the percentage of reachable documents discovered by the approach. However, the query plans that achieve the highest degree of completeness are among the plans that discover a comparably high number of reachable documents.

- The structure of the link graph of a queried Web of Linked Data affects the degree of completeness that query execution plans with link traversing iterators may achieve for $C_{\mathsf{LD(M)}}$ queries. In general, Webs of Linked Data that have a high b-interlinkage (that is, many bidirectional data links) are more beneficial for the iterator-based approach. Furthermore, it is necessary that the seed documents for a given query in a queried Web contain matching triples for a triple pattern in the query.

- The property that different logical plans for a query may not be semantically equivalent has a practical impact, and so does the property that the same plan may result in different answers. However, the latter is considerably less significant than the former.

- Seed-based query plans provide a better chance for achieving a comparably high degree of completeness than non-seed-based plans.

7.4. Summary

This chapter studies the suitability of the well-known iterator model for implementing the traversal-based query execution model presented in the previous chapter. To this end, we introduced the notion of link traversing iterators, which present a straightforward adaptation of iterators as used for the static case of querying a fixed set of RDF triples. The primary difference is that calling the `GetNext` function of a link traversing iterator may have the side effect of expanding the query-local dataset using data retrieved as per the traversal-based execution model. We then analyzed formally and experimentally the resulting, iterator-based implementation approach for our execution model.

The analysis revealed a major limitation of the approach: While the approach is sound (i.e., every valuation returned for any $C_{\mathsf{LD(M)}}$ query over any Web of Linked Data is a solution of the expected query result), the approach cannot guarantee completeness of the set of query solutions returned. We emphasize that this limitation is a specific property of the iterator-based implementation approach; it is *not* a property of the general, traversal-based execution strategy captured by our execution model (as we have shown in the previous chapter; cf. Theorem 6.1, page 126).

Further results of analyzing the implementation approach formally are the following. The approach guarantees termination for any possible query execution process. This guarantee presents an advantage of the approach over the general execution model, for which such a guarantee does not exist. However, as another issue we show that query execution plans consisting of link traversing iterators are not semantically equivalent. Hence, executing different possible query plans for the same query may result in obtaining different subsets of the complete query result. Similarly, even for the same query plan we may observe different execution processes and obtain different answers. Our experimental analysis verified that the latter two properties have an actual impact in practice.

Part III.

Conclusions

8. Conclusions

Due to the increasing adoption of publishing principles for Linked Data, the WWW evolves into a space in which more and more structured data is published and interlinked in a standardized manner. The feasibility to query this data space opens possibilities not conceivable before. As a result, we are witnessing the emergence of a new query paradigm that is tailored to the characteristics of Linked Data and the WWW. By relying only on the Linked Data publishing principles, this new paradigm, referred to as Linked Data query processing, bears the potential to enable users to fully benefit from the increasing amounts of Linked Data published on the WWW. This dissertation studies the foundations of this new query paradigm. In the following, we summarize the main results of this study and outline directions for future research.

8.1. Main Results

Our first main contribution is a formal framework for defining and analyzing queries over Linked Data on the WWW. This framework includes a data model that formalizes the notion of a *Web of Linked Data* and several related concepts. Additionally, the formal framework includes a computation model whose main concept is an *LD machine*, that is, a restricted Turing machine that formally captures the capabilities of systems that access Linked Data on the WWW as per the Linked Data principles. In this dissertation we used our formal framework to introduce and investigate adaptations of the query language SPARQL as a language for Linked Data queries.

The first adaptation, which we call $\text{SPARQL}_{\mathsf{LD}}$, is based on a full-Web query semantics according to which the scope of a query is all Linked Data on the queried Web. We showed formally that there does not exist a satisfiable $\text{SPARQL}_{\mathsf{LD}}$ query that is finitely computable by an LD machine (cf. Theorem 3.2, page 53). As a consequence, there cannot exist a query execution approach that guarantees an execution of a satisfiable $\text{SPARQL}_{\mathsf{LD}}$ query (over Linked Data on the WWW) that both terminates and returns the complete query result. Moreover, if such a query is non-monotonic, it is not even possible to guarantee a *nonterminating* execution that eventually returns all elements of the complete result (i.e., in terms of our formal framework, non-monotonic $\text{SPARQL}_{\mathsf{LD}}$ queries are not eventually computable by an LD machine; cf. Theorem 3.2). We emphasize that the limited computational feasibility of $\text{SPARQL}_{\mathsf{LD}}$ queries is not an artifact of allowing for infinitely large Webs of Linked Data in our data model. Instead, the reason for these limitations is the infiniteness of the set of all possible HTTP URIs.

Given the aforementioned limitations of $\text{SPARQL}_{\mathsf{LD}}$ queries, we introduced an alternative adaptation of SPARQL for Linked Data queries; that is, we defined a family of reachability-based query semantics. Queries under these semantics, called $\text{SPARQL}_{\mathsf{LD(R)}}$

queries, are restricted to range over well-defined, reachable subwebs of the queried Web. Each of the corresponding query semantics is based on a notion of reachability that is specified formally using the concept of reachability criteria. For instance, we discussed a reachability criterion called c_{Match}; according to c_{Match}, the scope of a given query is a reachable subweb covering the portion of a queried Web that can be reached by traversing along all those data links that may also be used for constructing the query result.

Our analysis of $\text{SPARQL}_{\mathsf{LD(R)}}$ queries identified a sub-family of reachability-based query semantics whose reachability criteria ensure finiteness of all query-specific reachable subwebs. We showed that $\text{SPARQL}_{\mathsf{LD(R)}}$ queries under a query semantics in this sub-family are finitely computable by an LD machine (cf. Proposition 4.12, page 91). Hence, it is possible to design a query execution approach that guarantees complete, terminating executions of such queries.

$\text{SPARQL}_{\mathsf{LD(R)}}$ queries under any of the other reachability-based query semantics (which includes the query semantics defined by the aforementioned reachability criterion c_{Match}) have a limited computational feasibility similar to $\text{SPARQL}_{\mathsf{LD}}$ queries (cf. Theorem 4.3, page 91). However, in contrast to $\text{SPARQL}_{\mathsf{LD}}$, the reason for these limitations in the case of $\text{SPARQL}_{\mathsf{LD(R)}}$ is the possibility of infinitely large Webs of Linked Data.

Further results regarding properties of $\text{SPARQL}_{\mathsf{LD}}$ queries and $\text{SPARQL}_{\mathsf{LD(R)}}$ queries focus on several decision problems: In particular, by taking into account the access limitations to a Web of Linked Data that is distributed over the WWW, it is impossible to decide for all $\text{SPARQL}_{\mathsf{LD}}$ queries whether there exists a complete, terminating execution of a given $\text{SPARQL}_{\mathsf{LD}}$ query over a given Web of Linked Data (cf. Theorem 3.1, page 52), and it is also undecidable whether the expected result of such a query over an arbitrary Web of Linked Data is finite (cf. Theorem 3.3, page 58). The same property holds for $\text{SPARQL}_{\mathsf{LD(R)}}$ queries (cf. Theorems 4.2 and 4.1 on pages 90 and 68, respectively). We showed these negative results formally by proving that the termination problem and the finiteness problem for $\text{SPARQL}_{\mathsf{LD}}$ (resp. for $\text{SPARQL}_{\mathsf{LD(R)}}$) are undecidable for an LD machine. However, there are (positive and negative) exceptions to these general results:

- For any $\text{SPARQL}_{\mathsf{LD(R)}}$ query under a (reachability-based) query semantics whose reachability criterion ensures finiteness, any expected query result (over any Web of Linked Data) is finite (cf. Proposition 4.2, page 66) and there exist terminating executions that return this result completely (cf. Proposition 4.11, page 87).
- For all queries ($\text{SPARQL}_{\mathsf{LD}}$ and $\text{SPARQL}_{\mathsf{LD(R)}}$) that are *boundedly satisfiable* (cf. Definition 2.8, page 24), any expected query result is finite by definition.
- There cannot exist an execution of an *unboundedly satisfiable* $\text{SPARQL}_{\mathsf{LD}}$ query that terminates after returning all elements of the expected query result (cf. Proposition 3.3, page 49).

In addition to defining and analyzing SPARQL-based Linked Data queries, we studied fundamental properties of a traversal-based strategy for executing such queries. While this strategy may be implemented in a multitude of ways, the strategy in general is based on two core principles: The first principle is to select recursively URIs for data retrieval by traversing particular data links at query execution time; the other one being the integration of data retrieval into the result construction process (instead of separating data retrieval and result construction into two consecutive phases). To put this strategy into the context of existing work, we provided a comprehensive review of possible techniques for executing Linked Data queries.

Additionally, we defined an abstract query execution model that formalizes our traversal-based query execution strategy in an implementation-independent manner. This model enabled us to conduct a formal analysis of the strategy in general (that is, without having to worry about the peculiarities of any approach for implementing the strategy). The main result of this analysis proves that the general strategy is sound and complete for one of the reachability-based query semantics that we introduced—namely, the query semantics that is defined by the aforementioned reachability criterion c_{Match} (cf. Theorem 6.1, page 126).

Finally, we investigated the suitability of using the well-known concept of pipelined iterators to implement our traversal-based query execution strategy. As a basis for this investigation, we introduced a straightforward approach for such an implementation and verified formally that this approach indeed implements the aforementioned abstract query execution model. Given this implementation approach we identified a serious shortcoming: The approach cannot guarantee completeness; that is, if we execute (conjunctive) $\text{SPARQL}_{\mathsf{LD(R)}}$ queries under c_{Match}-semantics using this approach, we may obtain incomplete query results. This limitation is due to the following behavior of pipelined iterators: Whenever such an iterator requests a next input element from the preceding iterator in the pipeline, it discards the previously retrieved input element. Usually, discarding such elements is not a problem because pipelined iterators typically operate over a dataset that is complete from the outset. However, in our scenario, the iterators operate over a dataset that they augment continuously; as a consequence, some data that would allow an iterator to generate particular output elements from a given input element, may become available only after the iterator has already discarded the input element and, thus, the iterator cannot generate these output elements.

Due to this issue, the iterator implementation of traversal-based query execution may generate incomplete query results, and we also showed that different, alternative query execution plans for a given query may not be semantically equivalent. Moreover, even for the same plan we may observe different execution processes and obtain different results. At least it is guaranteed that all query result elements returned by such a process are elements of the expected query result (cf. Theorem 7.1, page 137).

To achieve additional practical insight into the characteristics of using pipelined iterators for implementing traversal-based query execution, we analyzed this approach experimentally. The main results of our experiments can be summarized as follows: The times required to execute queries over “real” Linked Data on the WWW range from seconds up to almost two hours (depending on the query). The dominating factor of these query execution times is the time spend on data retrieval. The percentage of relevant reachable subwebs that the iterators discover during such executions ranges from slightly above 0% to 100%. Similarly, the percentage of expected query result elements returned by such executions ranges from 0% to 100%. In both cases these numbers depend on the link structure of queried Webs. In particular, we observed that the existence of many bidirectional data links increases the chances for achieving a comparably high percentage of query result completeness. Another important factor that influences the completeness of returned query results is the selected query execution plan. We identified a specific property (seed-basedness, cf. Definition 7.4, page 174) that characterizes plans whose executions a more likely to return a high percentage of expected query result elements.

8.2. Directions for Future Work

In this dissertation we focused on aspects of Linked Data query processing that we consider as most fundamental for studying the foundations of this new query processing paradigm. In the following, we point out further aspects that we have deliberately ignored and outline several ways to extend our work by taking these aspects into account.

8.2.1. Schema Heterogeneity

A fundamental characteristic of the WWW is that the publication of content is not coordinated centrally. While such an absence of coordination is critical to the openness and the growth of Linked Data on the WWW, it entails the typical data integration problem of schema heterogeneity. That is, Linked Data providers are free to choose the RDF vocabularies based on which they represent their data. Since different vocabularies may overlap w.r.t. the classes and properties that they define, a query expressed in terms of one vocabulary must be rewritten to benefit from data represented using a different vocabulary (alternatively, the data may be rewritten to match the vocabulary used by the query). Future research might extend our work to support query (or data) rewriting.

A prerequisite for such an extension is the availability of vocabulary mappings that specify the relationships between the RDF vocabularies used by query-relevant data sources. Such mappings may be provided by vocabulary maintainers or by third parties; ideally, these mappings are published as Linked Data along with the mapped vocabularies. Alternatively, it is possible to use schema matching [131] or ontology matching [148] to establish mappings between RDF vocabularies.

Based on such mappings, Linked Data queries might be rewritten by applying techniques that have been introduced for query answering using views (as surveyed by Halevy [62]). In fact, there already exist a few related approaches that focus on SPARQL query rewriting in the context of querying a federation of SPARQL endpoints [36, 90, 110, 117]. A noteworthy assumption of these approaches is that the queried set of data sources is given beforehand and, thus, a fixed set of all relevant vocabulary mappings can be specified prior to query execution. This assumption limits the suitability of these approaches for the traversal-based query execution model discussed in this dissertation (because this model integrates initially unknown data sources on the fly). Hence, to deal with schema heterogeneity in our execution model—and still leverage the full potential of this model—it requires novel query rewriting approaches in which the discovery (or creation) of mappings and the rewriting of queries is also performed on the fly.

8.2.2. Coreferences

Another typical data integration issue entailed by the decentralized publication of Linked Data on the WWW are coreferences: Although URIs are used as globally unique identifiers for denoting entities in Linked Data, the underlying RDF data model does not make the unique name assumption [92]; hence, different publishers may denote the same entity using different URIs. As a consequence, some of the data about such a coreferenced entity will be ignored by query processing approaches that do not detect and resolve the coreference. The models in this dissertation ignore the problem of coreferenced entities.

However, coreference resolution [52]—also referred to as duplicate detection [45] or reference reconciliation [43]—has been studied extensively in the database and information systems literature. Therefore, a possible extension of our work is to revisit existing coreference resolution approaches in the context of Linked Data query processing. A necessary basis for dealing with coreferences in this context are coreference-aware query semantics.

8.2.3. Trustworthiness and Data Quality

Another consequence of the openness of the WWW is that Linked Data from different data sources may be of different quality, that is, some data might be inaccurate, inconsistent, outdated, etc. As a result, users may consider some data more trustworthy and reliable than other data. Therefore, to provide a holistic solution for querying Linked Data on the WWW, the subjective trustworthiness of data (and of query results) cannot be ignored. Thus, our work might be extended accordingly. In particular, such an extension may include (i) augmenting our data model with a trust model and (ii) using a trust-aware query language instead of SPARQL.

The goal of adding a trust model is to enable an automated assessment of the trustworthiness of data in a Web of Linked Data. Hence, the main concept introduced by such a trust model would be a *trust function* that assigns every LD document—or even every RDF triple—a context-specific trustworthiness score. The particular meaning of such a score is to be specified by the trust model, and so is the definition of the trust function. In fact, defining a trust function is a challenging research problem in itself. As a possible starting point we refer to earlier work in which we review existing approaches for measuring trustworthiness of data and discuss different factors that may influence the decision to trust some data [70].

Given a suitable trust model, a possible query language for defining a trust-aware notion of Linked Data queries is tSPARQL [69]. This language extends SPARQL by redefining the SPARQL algebra such that the resulting algebra operates over sets of *trust-weighted valuations*, that is, conventional SPARQL valuations (as introduced in Section 3.2.1, cf. page 39) that are associated with a trustworthiness score. Furthermore, tSPARQL adds two new operators that enable users to describe trustworthiness requirements and to access the trustworthiness of (intermediate) solutions; the latter may be used to obtain a trustworthiness-related ordering of a query result or to output trustworthiness scores as part of a query result. Similar to SPARQL, tSPARQL is defined for expressing queries over fixed, a-priori defined collections of RDF data (for tSPARQL these collections need to be augmented with a trust function).

Apparently, it is possible to use tSPARQL as a query language for (trust-aware) Linked Data queries by defining a full-Web query semantics or reachability-based query semantics for tSPARQL in a manner similar to our definitions for SPARQL in Chapters 3 and 4. Then, an interesting problem would be, how to execute the resulting queries efficiently. For instance, a suitable execution approach should aim to avoid retrieving LD documents unless their data may meet the trustworthiness requirements given in queries. A related, more explicit extension of our work would be to add a trust-aware notion to our concept of a reachability criterion.

Instead of focusing only on trustworthiness (also often referred to as *believability* in the data quality literature [121]), other dimensions of data quality, such as accuracy, consistency, or timeliness [137], may be covered by extending tSPARQL (and our data model) accordingly. Of course, it is also possible to revisit other quality-related query languages in the context of Linked Data queries (or develop a new one).

8.2.4. Dynamic Environment

The models presented in this dissertation make the simplifying assumption that a queried Web of Linked Data is static (during the process of executing a query). However, given that Linked Data is published via ordinary, HTTP-based Web servers, it is impossible to isolate a particular state of a Web of Linked Data for a given query. Hence, an important extension of our work includes adding some notion of a dynamically changing Web of Linked Data to our models.

A possible basis for capturing such a dynamic environment is to formalize a *changing Web of Linked Data* as an infinite sequence of (static) Webs of Linked Data. Mendelzon and Milo use this approach for their dynamic model of the WWW [112]. It remains to be seen whether this approach is suitable for defining query semantics for dynamic Linked Data queries (i.e., queries over a changing Web of Linked Data).

An alternative direction of future work is to study what extra data access features need to be provided by Web servers in order to support queries over a particular state of a (dynamic) Web of Linked Data. If multiple such features are conceivable, it would be interesting to compare them w.r.t. the additional requirements that each of them entails for data publishers' infrastructures and the guarantees that they provide (such guarantees may be similar to the different notions of consistency supported by distributed systems [154] or the different levels of isolation of database transactions [57]).

8.2.5. Query Expressiveness

This dissertation focuses on Linked Data queries that can be expressed using a core fragment of the SPARQL query language. However, our definitions of these queries can be easily extended to cover additional features introduced in the W3C specification of SPARQL [63] (e.g., query result ordering, property paths, aggregates). The effect of such extensions on the computational feasibility of resulting queries might be analyzed. Along the same lines, a possible support of more powerful SPARQL entailment regimes [53] might be investigated. Similarly, our traversal-based query execution model might be extended to support more expressive queries. Another possibility for future work is to use our computation model to analyze the computational feasibility of other Linked Data query languages (such as the navigational languages discussed in Section 3.1.2, page 34f).

Part IV.

Appendix

A. Commonly Used Symbols

The following table lists symbols used in the formalisms of this dissertation, and the concepts denoted by these symbols.

Table A.1.: Symbols used in this dissertation.

Symbol	Concept	Reference
$adoc$	mapping from URIs to authoritative LD documents in a Web of Linked Data	Definition 2.1, page 16
$\mathsf{AllData}(W)$	the set of all RDF triples in a Web of Linked Data W	Section 2.1.1, page 17
$\mathsf{AUG}(\sigma, t, tp)$	the (t, tp)-augmentation of partial solution σ	Definition 6.3, page 117
$\mathcal{B}$	the (countably infinite) set of all blank nodes	Section 2.1.1, page 15
B	a basic graph pattern (BGP)	Section 6.1, page 112
c	a reachability criterion	Definition 4.1, page 62
$\mathcal{C}$	the infinite set of all possible reachability criteria	–
$\mathcal{C}_{\mathsf{const}}$	the infinite set of all constant reachability criteria	Definition 4.9, page 75
$\mathcal{C}_{\mathsf{ef}}$	the infinite set of all reachability criteria that ensure finiteness	Definition 4.5, page 67
$\mathcal{C}_{\mathsf{nef}}$	the infinite set of all reachability criteria that do not ensure finiteness	–
d	an LD document	Section 2.1.1, page 16
$\mathcal{D}$	the (countably infinite) set of all LD documents	Section 2.1.1, page 16
D	the set of LD documents in a Web of Linked Data	Definition 2.1, page 16

Continued on next page ...

Table A.1 – continued from previous page

Symbol	Concept	Reference
$\mathfrak{D}$	currently discovered subweb of a queried Web during a query execution	Section 6.3.7, page 125
$\mathfrak{D}_{\mathsf{init}(S,W)}$	the S-seed subweb of a Web of Linked Data W	Definition 6.5, page 119
$data$	maps each LD document in a Web of Linked Data to a set of RDF triples	Definition 2.1, page 16
$\mathrm{dom}(f)$	the domain of a function f	–
$\mathrm{enc}(x)$	encoding of some element x (where x may be a single RDF triple, a set of triples, a full Web of Linked Data, a valuation, etc.)	Appendix B, page 193f
$\mathsf{EXP}(W_{\mathfrak{D}}, t, W)$	the t-expansion of $W_{\mathfrak{D}}$ in a Web of Linked Data W	Definition 6.6, page 120
G	a set of RDF triples	Section 2.1.1, page 15
I	an iterator	Section 7.1.1, page 130
$\mathcal{L}$	the (countably infinite) set of all literals	Section 2.1.1, page 15
μ	a valuation	Section 3.2.1, page 39
$\mu_{\emptyset}$	the empty valuation (i.e., $\mathrm{dom}(\mu_{\emptyset}) = \emptyset$)	Section 3.2.1, page 39
$\mu[tp]$	the application of valuation μ to triple pattern tp	Section 3.2.1, page 40
$\mu[B]$	the application of valuation μ to BGP B	Section 6.1, page 112
P	a SPARQL expression	Section 3.2.1, page 39
$\mathcal{P}$	the infinite set of all possible SPARQL expressions	Definition 4.1, page 62
$\mathfrak{P}$	the set of partial solutions currently constructed during a query execution	Section 6.3.7, page 125
Q	an arbitrary Linked Data query	Definition 2.5, page 23
$\mathcal{Q}^{P}$	a SPARQL$_{\mathsf{LD}}$ query	Definition 3.1, page 42
$\mathcal{Q}^{P,S}_{c}$	a SPARQL$_{\mathsf{LD(R)}}$ query	Definition 4.4, page 63
$\mathcal{Q}^{B,S}_{c}$	a C$_{\mathsf{LD(R)}}$ query	Definition 6.1, page 113
$\mathcal{Q}^{B,S}$	a C$_{\mathsf{LD(M)}}$ query	Definition 6.10, page 123

Continued on next page ...

Table A.1 – continued from previous page

Symbol	Concept	Reference
R	a (particular) reachable subweb of a Web of Linked Data	–
ϱ	a grounding isomorphism	Definition 3.2, page 44
S	a set of seed URIs	–
st	an QE state	Definition 6.7, page 121
$\mathrm{sub}_k^{\prec}(B)$	a BGP that consists of the first k triple patterns from BGP B (assuming the order $\prec$ for the triple patterns in B)	Section 7.1.1, page 130
σ	a partial solution	Definition 6.2, page 116
$\sigma_\emptyset$	the empty partial solution	Section 6.3.2, page 117
$\mathcal{O}(\mathcal{Q}_c^{B,S}, W)$	the set of all partial solutions for $\mathrm{C}_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{B,S}$ in Web of Linked Data W	Definition 6.2, page 116
t	an RDF triple	Section 2.1.1, page 15
tp	a triple pattern	Section 3.2.1, page 39
$\mathrm{tp}(I_k)$	the triple pattern of iterator I_k	Section 7.1.1, page 130
$\mathcal{T}$	the infinite set of all possible RDF triples	Section 2.1.1, page 15
τ	an AE task	Definition 6.8, page 122
$\tau[st]$	performance of AE task τ in QE state st	Definition 6.9, page 122
$\mathrm{terms}(x)$	the set of all URIs, blank nodes, and literals mentioned in x (where x may be an RDF triple, or a set of RDF triples)	Section 2.1.1, page 15
$\mathcal{U}$	the (countably infinite) set of all URIs	Section 2.1.1, page 15
u	a URI	–
$\mathrm{uris}(x)$	the set of all URIs mentioned in x (where x may be an RDF triple, a set of RDF triples, a triple pattern, a SPARQL expression, a BGP, or a valuation)	Section 2.1.1, page 15; Section 3.2.1, page 39
$\mathcal{V}$	the (countably infinite) set of all query variables	Section 3.2.1, page 39

Continued on next page ...

Table A.1 – continued from previous page

Symbol	Concept	Reference
$\mathrm{vars}(x)$	the set of all variables mentioned in x (where x can be a triple pattern, a SPARQL expression, or a BGP)	Section 3.2.1, page 39
W	a Web of Linked Data	Definition 2.1, page 16
$W_{\mathfrak{D}}$	a discovered subweb of a Web of Linked Data W	Definition 6.4, page 118
$\mathcal{W}_{\mathsf{All}}$	the infinite set of all Webs of Linked Data	Definition 2.5, page 23
$\emptyset$	an empty set	–
$\perp$	the nonexistent LD document	Definition 2.1, page 16
$\sim$	compatibility of valuations	Section 3.2.1, page 40
$\lhd$	restrictiveness relation over reach. criteria	Definition 4.6, page 70
$\sqcap$	conjunctive combination of reach. criteria	Definition 4.7, page 71
$\sqcup$	disjunctive combination of reach. criteria	Definition 4.7, page 71

B. Encoding of Structures Related to Query Computation

This appendix defines an encoding for representing (fragments of) a Web of Linked Data and query results on the tapes of an LD machine (cf. Definition 2.9 on page 27)

For the encoding we use an alphabet that consists of the following 12 characters:

$$0\,1\,\mathrm{U}\,\mathrm{B}\,\mathrm{L}\,\langle\,\rangle\,,\,\langle\!\langle\,\rangle\!\rangle\,\sharp\,\rightarrow$$

Assume total orders $\prec_{\mathcal{U}}$, $\prec_{\mathcal{B}}$, $\prec_{\mathcal{L}}$, and $\prec_{\mathcal{V}}$ over the sets $\mathcal{U}$ (all URIs), $\mathcal{B}$ (all blank nodes), $\mathcal{L}$ (all literals), and $\mathcal{V}$ (all query variables), respectively. In all four cases such an order could simply be the lexicographic order of corresponding string representations.

Based on the orders $\prec_{\mathcal{U}}$, $\prec_{\mathcal{B}}$, and $\prec_{\mathcal{L}}$, we construct a total order $\prec_t$ over the set $\mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$. For two distinct elements $x, y \in \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$, $x \prec_t y$ holds if either

(i) $x \in \mathcal{U}$ and $y \notin \mathcal{U}$, or
(ii) $x \in \mathcal{B}$ and $y \in \mathcal{L}$, or
(iii) $x \in \mathcal{U}$ and $y \in \mathcal{U}$ and $x \prec_{\mathcal{U}} y$, or
(iv) $x \in \mathcal{B}$ and $y \in \mathcal{B}$ and $x \prec_{\mathcal{B}} y$, or
(v) $x \in \mathcal{L}$ and $y \in \mathcal{L}$ and $x \prec_{\mathcal{L}} y$.

Furthermore, we construct a total order $\prec_{\mathcal{T}}$ over the set of all RDF triples. For two distinct RDF triples $t_1 = (s_1, p_1, o_1)$ and $t_2 = (s_2, p_2, o_2)$, $t_1 \prec_{\mathcal{T}} t_2$ holds if either

(i) $s_1 \neq s_2$ and $s_1 \prec_t s_2$, or
(ii) $s_1 = s_2$ and $p_1 \neq p_2$ and $p_1 \prec_t p_2$, or
(iii) $s_1 = s_2$ and $p_1 = p_2$ and $o_1 \neq o_2$ and $o_1 \prec_t o_2$.

B.1. Encoding Basic Elements

For every URI $u \in \mathcal{U}$, the encoding of u, denoted by $\mathrm{enc}(u)$, is the word that begins with character U, followed by the binary representation of u.

For every blank node $bn \in \mathcal{B}$, the encoding of bn, denoted by $\mathrm{enc}(bn)$, is the word that begins with character B, followed by the binary representation of bn.

For every literal $c \in \mathcal{L}$, the encoding of c, denoted by $\mathrm{enc}(c)$, is the word that begins with character L, followed by the binary representation of c.

For every variable $?v \in \mathcal{V}$, the encoding of $?v$, denoted by $\mathrm{enc}(?v)$, is the binary representation of $?v$.

B.2. Encoding RDF Triples

The encoding of an RDF triple $t = (s, p, o)$, denoted by $\mathrm{enc}(t)$, is a word

$$\langle\, \mathrm{enc}(s)\, ,\, \mathrm{enc}(p)\, ,\, \mathrm{enc}(o)\, \rangle$$

The encoding of a set of RDF triples $T = \{t_1, \dots, t_n\}$, denoted by $\mathrm{enc}(T)$, is a word

$$\langle\!\langle\, \mathrm{enc}(t_1)\, ,\, \mathrm{enc}(t_2)\, ,\, \dots\, ,\, \mathrm{enc}(t_n)\, \rangle\!\rangle$$

where the $\mathrm{enc}(t_i)$ are ordered as follows: For each pair of RDF triples $t_x, t_y \in T$, subword $\mathrm{enc}(t_x)$ occurs before subword $\mathrm{enc}(t_y)$ in $\mathrm{enc}(T)$ if $t_x \prec_{\mathcal{T}} t_y$.

B.3. Encoding Webs of Linked Data

For a Web of Linked Data $W = (D, \mathit{data}, \mathit{adoc})$, the encoding of each LD document $d \in D$, denoted by $\mathrm{enc}(d)$, is the word $\mathrm{enc}(\mathit{data}(d))$. The encoding of W itself, denoted by $\mathrm{enc}(W)$, is a word

$$\sharp\, \mathrm{enc}(u_1)\, \mathrm{enc}(\mathit{adoc}(u_1))\, \sharp\, \mathrm{enc}(u_2)\, \mathrm{enc}(\mathit{adoc}(u_2))\, \sharp\, \dots\, \sharp\, \mathrm{enc}(u_i)\, \mathrm{enc}(\mathit{adoc}(u_i))\, \sharp\, \dots$$

where $u_1, u_2, \dots, u_i, \dots$ is the (potentially infinite but countable) list of all URIs for which $\mathit{adoc}(u_j) \neq \bot$, ordered according to $\prec_{\mathcal{U}}$.

We note that the word $\mathrm{enc}(W)$ is infinitely large if and only if Web of Linked Data W is infinite. Furthermore, we note that $\mathrm{enc}(W)$ may contain data of an LD document $d \in D$ multiple times; more precisely, the data of d is present as many times as there exist URIs $u \in \mathcal{U}$ with $\mathit{adoc}(u) = d$. We emphasize that such a duplication does not present a problem for analyzing set-based query semantics as we do in this dissertation.

B.4. Encoding Valuations

The encoding of a valuation μ with $\mathrm{dom}(\mu) = \{?v_1, \dots, ?v_n\}$, denoted by $\mathrm{enc}(\mu)$, is a word

$$\langle\!\langle\, \mathrm{enc}(?v_1) \rightarrow \mathrm{enc}\big(\mu(?v_1)\big)\, ,\, \dots\, ,\, \mathrm{enc}(?v_n) \rightarrow \mathrm{enc}\big(\mu(?v_n)\big)\, \rangle\!\rangle$$

where the subwords $\mathrm{enc}(?v_i) \rightarrow \mathrm{enc}\big(\mu(?v_i)\big)$ are ordered as follows: For each pair of variables $?v_x \in \mathrm{dom}(\mu)$ and $?v_y \in \mathrm{dom}(\mu)$, subword $\mathrm{enc}(?v_x) \rightarrow \mathrm{enc}\big(\mu(?v_x)\big)$ occurs before $\mathrm{enc}(?v_y) \rightarrow \mathrm{enc}\big(\mu(?v_y)\big)$ in $\mathrm{enc}(\mu)$ if $?v_x \prec_{\mathcal{V}} ?v_y$.

A *possible* encoding of a (potentially infinite) set of valuations $\Omega = \{\mu_1, \mu_2, \dots\}$, denoted by $\mathrm{enc}(\Omega)$, is a word

$$\mathrm{enc}(\mu_1)\, \mathrm{enc}(\mu_2) \dots$$

where the subwords $\mathrm{enc}(\mu_i)$ may occur in any order.

C. Basic Properties of SPARQL Queries

In the following, we discuss basic theoretical properties of SPARQL expressions that are relevant for the results in this dissertation. In particular, these properties are satisfiability and monotonicity, as well as the particular notions of bounded and unbounded satisfiability that we introduce in Section 2.1.2 (cf. page 22ff). We show that all four properties are undecidable for SPARQL in general. As a consequence, we identify specific fragments of SPARQL for which these properties are decidable. Table C.1 summarizes our findings (cf. page 196). A more extensive discussion may reveal further fragments for each property. However, given that SPARQL queries as functions over a fully accessible set of RDF triples are not the main focus of this dissertation, we consider such a discussion out of scope.

C.1. Satisfiability

As the first relevant property of SPARQL expressions we discuss satisfiability.

Definition C.1. A SPARQL expression P is *satisfiable* if there exists a (potentially infinite) set of RDF triples G such that the result of evaluating P over G is not empty, that is, $[\![P]\!]_G \neq \emptyset$. A SPARQL expression is *unsatisfiable* if it is not satisfiable. □

Example C.1. Let P_1 be SPARQL expression $\big((?v, \mathsf{name}, ?vn)$ FILTER $?vn = \text{"Vendor1"}\big)$ and let P_2 be SPARQL expression $\big((?v, \mathsf{name}, \text{"Vendor1"})$ FILTER $(\neg\text{bound}(?v))\big)$. It is easily verified that P_1 is satisfiable, whereas, P_2 is unsatisfiable (for a formal, more general proof of these claims we refer to Proposition C.3, page 199). □

The satisfiability problem (for SPARQL) is the following (ordinary) decision problem:

Problem:	SATISFIABILITY(SPARQL)
Input:	a SPARQL expression P
Question:	Is P satisfiable?

The following result shows that we cannot answer this question in general.

Proposition C.1. SATISFIABILITY(SPARQL) *is undecidable.*

Proof. The satisfiability problem for relational algebra is well known to be undecidable [4]. Furthermore, relational algebra and SPARQL have equivalent expressive power. More precisely, Angles and Gutierrez [7] show that there exist (Turing) computable, bijective mappings T_Q, T_D and T_S such that

Form of SPARQL expression P	**Condition(s)**	**Property**	**Corresponding result**
any without FILTER	-	satisfiable	Proposition C.2, p.197
$(P'$ FILTER $R)$	P' is satisfiable and R is $?v = c$ where $?v \in \text{cvars}(P')$ and $c \notin \mathcal{V}$	satisfiable	Proposition C.3, p.199
$(P'$ FILTER $R)$	R is $(\neg\text{bound}(?v))$ where $?v \in \text{cvars}(P')$	non-satisfiable	Proposition C.3, p.199
$(P'$ FILTER $R)$	R is $\text{bound}(?v)$ where $?v \notin \text{vars}(P')$	non-satisfiable	Proposition C.3, p.199
any	P is non-satisfiable	monotonic	Property C.1, p.201
any without OPT	-	monotonic	Proposition C.6, p.202
$(P_1$ OPT $P_2)$	$\text{vars}(P_1) = \emptyset$ and P_1 and P_2 are monotonic	monotonic	Proposition C.7, p.203
$(P_1$ OPT $P_2)$	$\text{vars}(P_2) = \emptyset$ and P_1 is monotonic	monotonic	Proposition C.7, p.203
$(P_1$ OPT $P_2)$	$\text{vars}(P_2) = \emptyset$ and P_1 is non-monotonic	non-monotonic	Proposition C.7, p.203
$(P_1$ OPT $P_2)$	$\text{vars}(P_1) \neq \emptyset$, $\text{vars}(P_2) \neq \emptyset$, and P_1 is not satisfiable	monotonic	Proposition C.7, p.203
$(P_1$ OPT $P_2)$	$\text{vars}(P_1) \neq \emptyset$, $\text{vars}(P_2) \neq \emptyset$, P_1 is monotonic, and P_2 is not satisfiable	monotonic	Proposition C.7, p.203
any	P is not satisfiable	not boundedly satisfiable	by definition
any	P is not satisfiable	not unboundedly satisfiable	by definition
any	P is satisfiable and $\text{vars}(P) = \emptyset$	boundedly satisfiable	(Lemma C.3, p.207)
any without FILTER, OPT, and UNION	$\text{vars}(P) \neq \emptyset$	unboundedly satisfiable	Proposition C.9, p.207
any without FILTER, OPT, and UNION	$\text{vars}(P) = \emptyset$	boundedly satisfiable	Corollary C.1, p.209

Table C.1.: Theoretical properties of different types of SPARQL expressions as proved in this dissertation.

1. T_Q transforms relational algebra queries into SPARQL expressions,
2. T_D transforms relations into sets of RDF triples,
3. T_S transforms relations (resulting from evaluating relational algebra queries) into SPARQL query results such that (i) $T_S(q(D)) = T_Q(q)(T_D(D))$ for each relational algebra query q and relational database instance D, and (ii) $T_S(\emptyset) = \emptyset$.

From the existence of such mappings, it follows that if SATISFIABILITY(SPARQL) were decidable, then the satisfiability problem for relational algebra would be decidable, a contradiction. ■

We shall see that the satisfiability of SPARQL-based Linked Data queries studied in this dissertation always corresponds to the satisfiability of the SPARQL expressions used for these queries. Therefore, satisfiability of SPARQL expressions is relevant for most of our results in this dissertation. For instance, any unsatisfiable Linked Data query is finitely computable as we know from Proposition 2.3 (cf. page 29). As a consequence, even if we cannot decide satisfiability of SPARQL expressions in general, we are interested in fragments of SPARQL for which satisfiability is decidable.

Such a fragment are SPARQL expressions without FILTER:

Proposition C.2. *SPARQL expressions without* FILTER *are satisfiable.*

As a basis for proving Proposition C.2 we write $\mathrm{lit}(tp)$ to denote the set of all literals in a triple pattern $tp = (s, p, o)$; i.e., $\mathrm{lit}(tp)$ is defined as follows:

$$\mathrm{lit}(tp) := \begin{cases} \{o\} & \text{if } o \in \mathcal{L}, \\ \emptyset & \text{else.} \end{cases}$$

Overloading notation, we write $\mathrm{lit}(P)$ to denote the set of all literals in a SPARQL expression P, which we define recursively as follows:

1. If P is a triple pattern tp, then $\mathrm{lit}(P) := \mathrm{lit}(tp)$.
2. If P is $(P_1 \text{ AND } P_2)$, $(P_1 \text{ UNION } P_2)$, or $(P_1 \text{ OPT } P_2)$, then $\mathrm{lit}(P) := \mathrm{lit}(P_1) \cup \mathrm{lit}(P_2)$.
3. If P is $(P' \text{ FILTER } R)$, then $\mathrm{lit}(P) := \mathrm{lit}(P')$.

Furthermore, we introduce the notion of an *all permutations triple set*, that is, the set of all RDF triples that can be constructed from a given set of URIs and literals. Formally, given a finite set $A \subseteq \mathcal{U} \cup \mathcal{L}$ of URIs and literals, the *all permutations triple set* for A, denoted by $\mathrm{AllPerm}(A)$, is a set of RDF triples defined by:

$$\mathrm{AllPerm}(A) := (A \cap \mathcal{U}) \times (A \cap \mathcal{U}) \times A\,.$$

To prove Proposition C.2 we use the following lemma.

Lemma C.1. *Let P be a SPARQL expression without* FILTER*; let $A \subseteq \mathcal{U} \cup \mathcal{L}$ be a finite set of URIs and literals such that* $\mathrm{uris}(P) \subseteq A$ *and* $\mathrm{lit}(P) \subseteq A$. *Then, for each valuation $\mu : \mathcal{V} \to \mathrm{uris}(P)$ there exists a valuation $\mu' \in [\![P]\!]_{\mathrm{AllPerm}(A)}$ such that $\mu'(?v) = \mu(?v)$ for all variables $?v \in \mathrm{dom}(\mu')$.*

Proof of Lemma C.1. We prove the lemma by induction on the possible structure of SPARQL expression P.

Base case: Suppose SPARQL expression P is a triple pattern tp. Due to the construction of $\mathrm{AllPerm}(A)$, we have $\mu[tp] \in \mathrm{AllPerm}(A)$. Consequently, there exists a valuation $\mu' \in [\![P]\!]_{\mathrm{AllPerm}(A)}$ such that $\mu'(?v) = \mu(?v)$ for all $?v \in \mathrm{dom}(\mu')$.

Induction step: We distinguish the following three cases:

- P is (P_1 AND P_2). By the induction hypothesis, there exist $\mu_1 \in [\![P_1]\!]_{\mathrm{AllPerm}(A)}$ and $\mu_2 \in [\![P_2]\!]_{\mathrm{AllPerm}(A)}$ such that (i) $\mu_1(?v) = \mu(?v)$ for all $?v \in \mathrm{dom}(\mu_1)$ and (ii) $\mu_2(?v) = \mu(?v)$ for all $?v \in \mathrm{dom}(\mu_2)$. In particular, $\mu_1 \sim \mu_2$, and hence, for $\mu' = \mu_1 \cup \mu_2$ we have $\mu' \in [\![P]\!]_{\mathrm{AllPerm}(A)}$ and $\mu'(?v) = \mu(?v)$ for all $?v \in \mathrm{dom}(\mu')$, as desired.

- P is (P_1 OPT P_2). By the induction hypothesis, there exist $\mu_1 \in [\![P_1]\!]_{\mathrm{AllPerm}(A)}$ and $\mu_2 \in [\![P_2]\!]_{\mathrm{AllPerm}(A)}$ such that (i) $\mu_1(?v) = \mu(?v)$ for all $?v \in \mathrm{dom}(\mu_1)$ and (ii) $\mu_2(?v) = \mu(?v)$ for all $?v \in \mathrm{dom}(\mu_2)$. In particular, $\mu_1 \sim \mu_2$, and therefore, $\mu' = \mu_1 \cup \mu_2 \in [\![P]\!]_{\mathrm{AllPerm}(A)}$. Altogether we have found a valuation $\mu' \in [\![P]\!]_{\mathrm{AllPerm}(A)}$ such that $\mu'(?v) = \mu(?v)$ for all $?v \in \mathrm{dom}(\mu')$, as desired.

- P is (P_1 UNION P_2). By the induction hypothesis, there exists a $\mu' \in [\![P_1]\!]_{\mathrm{AllPerm}(A)}$ such that $\mu'(?v) = \mu(?v)$ for all $?v \in \mathrm{dom}(\mu')$. Note that $\mu' \in [\![P]\!]_{\mathrm{AllPerm}(A)}$. ■

Lemma C.1 readily implies Proposition C.2:

Proof of Proposition C.2. To show that a SPARQL expression P without FILTER is satisfiable, let $A \subset \mathcal{U} \cup \mathcal{L}$ be a finite set of URIs and literals such that (i) A contains at least one URI (ii) $\mathrm{uris}(P) \subseteq A$, and (iii) $\mathrm{lit}(P) \subseteq A$. Then, by Lemma C.1, the evaluation of P over $\mathrm{AllPerm}(A)$ is nonempty, that is, $[\![P]\!]_{\mathrm{AllPerm}(A)} \neq \emptyset$ and, thus, P is satisfiable. ■

For SPARQL expressions with FILTER we know from Example C.1 that some expressions are satisfiable and some are not (cf. page 195). To generalize the findings from Example C.1 we adopt Schmidt et al.'s notion of *certain variables* [140, Definition 6]. Informally, certain variables of a SPARQL expression P, denoted by $\mathrm{cvars}(P)$, are the variables in P that are guaranteed to be bound in each solution for P. Formally, $\mathrm{cvars}(P)$ is defined recursively as follows:

1. If P is a triple pattern tp, then $\mathrm{cvars}(P) := \mathrm{vars}(P)$.

2. If P is (P_1 AND P_2), then $\mathrm{cvars}(P) := \mathrm{cvars}(P_1) \cup \mathrm{cvars}(P_2)$.

3. If P is $(P_1$ UNION $P_2)$, then $\mathrm{cvars}(P) := \mathrm{cvars}(P_1) \cap \mathrm{cvars}(P_2)$.

4. If P is $(P_1$ OPT $P_2)$, then $\mathrm{cvars}(P) := \mathrm{cvars}(P_1)$.

5. If P is $(P'$ FILTER $R)$, then $\mathrm{cvars}(P') := \mathrm{cvars}(P')$.

Using the concept of certain variables we may show the following result.

Proposition C.3. *Let P be a SPARQL expression $(P'$ FILTER $R)$.*

1. *If P' is satisfiable and R is filter condition $?v = c$ where $?v \in \mathrm{cvars}(P')$ and $c \in (\mathcal{U} \cup \mathcal{L})$, then P is satisfiable.*

2. *If R is filter condition $(\neg\mathrm{bound}(?v))$ where $?v \in \mathrm{cvars}(P')$, then P is not satisfiable.*

3. *If R is filter condition $\mathrm{bound}(?v)$ where $?v \notin \mathrm{vars}(P')$, then P is not satisfiable.*

Proof. As a preliminary for this proof we recall Schmidt et al.'s result which shows that each certain variable of a SPARQL expression P is guaranteed to be bound in each solution for P [140, Proposition 1]. Formally, let P be a SPARQL expression and let G be a set of RDF triples, then $\mathrm{cvars}(P) \subseteq \mathrm{dom}(\mu)$ holds for all $\mu \in [\![P]\!]_G$. We now prove the three claims of Proposition C.3.

Claim 1. Let P be a SPARQL expression of the form $(P'$ FILTER $R)$ such that (i) P' is satisfiable and (ii) R is $?v = c$ where $?v \in \mathrm{cvars}(P')$ and $c \in (\mathcal{U} \cup \mathcal{L})$. Since P' is satisfiable, there exists a set of RDF triples G such that $[\![P']\!]_{G'} \neq \emptyset$. Let G' be such a set and let μ be an arbitrary solution for P' in G', i.e., $\mu \in [\![P']\!]_{G'}$. We construct another set of RDF triples G'' from G' by replacing each occurrence of RDF term $c' = \mu(?v)$ in G' by c. Formally, $G'' = \{\sigma(t) \,|\, t \in P'\}$ where function σ maps each RDF triple (x_1, x_2, x_3) to another RDF triple (x'_1, x'_2, x'_3) such that $x'_i = c$ if $x_i = c'$; otherwise $x'_i = x_i$ (for all $i = \{1, 2, 3\}$). Due to this construction we have $\mu \in [\![P]\!]_{G''}$. Hence, P is satisfiable.

Claim 2. Let P be a SPARQL expression of the form $(P'$ FILTER $R)$ such that filter condition R is $(\neg\mathrm{bound}(?v))$ where $?v \in \mathrm{cvars}(P')$. We need to distinguish two cases: Either subexpression P' is satisfiable or it is unsatisfiable. In the latter case, P is also unsatisfiable because for any set of RDF triples G it holds that $[\![P']\!]_G = \emptyset$ and, thus, $[\![P]\!]_G = \{\mu \in \emptyset \,|\, \mu \text{ satisfies } R\} = \emptyset$. Hence, it remains to discuss the case where P' is satisfiable. If P' is satisfiable, there exists a set of RDF triples G such that $[\![P']\!]_G \neq \emptyset$. Let G' be such a set and let μ be an arbitrary solution for P' in G', i.e., $\mu \in [\![P']\!]_{G'}$. W.l.o.g., we prove that P is unsatisfiable by showing that $\mu \notin [\![P]\!]_{G'}$. Since $?v$ is a certain variable of P', it holds that $?v \in \mathrm{dom}(\mu)$ and, thus, μ does not satisfy filter condition R. Therefore, $\mu \notin [\![P]\!]_{G'}$.

Claim 3. Let P be a SPARQL expression of the form $(P'$ FILTER $R)$ such that R is filter condition $\mathrm{bound}(?v)$ where $?v \notin \mathrm{vars}(P')$. If P' is unsatisfiable we directly conclude that P is unsatisfiable (see our discussion of claim 2 above). If P' is satisfiable, there exists a set of RDF triples G such that $[\![P']\!]_G \neq \emptyset$. Let G' be such a set and let μ be an arbitrary solution for P' in G', i.e., $\mu \in [\![P']\!]_{G'}$. W.l.o.g., we prove that P is unsatisfiable by showing that $\mu \notin [\![P]\!]_{G'}$. Since $?v \notin \text{vars}(P')$ it holds that $?v \notin \text{dom}(\mu)$ and, thus, μ does not satisfy filter condition R. Therefore, $\mu \notin [\![P]\!]_{G'}$. ■

Proposition C.3 covers only a few patterns of SPARQL expressions with FILTER. Providing a more comprehensive list is out of scope of this dissertation. However, we conclude the discussion of satisfiability by providing equivalences based on which SPARQL expressions with an unsatisfiable subexpression may be rewritten into a semantically equivalent expression. By applying these equivalences recursively, it may be possible to show that the original expression is also unsatisfiable; or, for some cases an unsatisfiable subexpression may be eliminated in order to show satisfiability of the original expression.

Proposition C.4. *Let symbol $\equiv$ denote semantic equivalence of SPARQL expressions, that is, for two SPARQL expressions P_1 and P_2 it holds that $P_1 \equiv P_2$ if $[\![P_1]\!]_G = [\![P_2]\!]_G$ for any possible set of RDF triples G. For any SPARQL expression P, the following properties hold:*

1. *If P is $(P_1$ AND $P_2)$ and P_1 or P_2 is unsatisfiable, then P is unsatisfiable.*
2. *If P is $(P'$ FILTER $R)$ and P' is unsatisfiable, then P is unsatisfiable.*
3. *If P is $(P_1$ OPT $P_2)$ and P_1 is unsatisfiable, then P is unsatisfiable.*
4. *If P is $(P_1$ OPT $P_2)$ and P_2 is unsatisfiable, then $P \equiv P_1$.*
5. *If P is $(P_1$ UNION $P_2)$ and P_1 is unsatisfiable, then $P \equiv P_2$.*
6. *If P is $(P_1$ UNION $P_2)$ and P_2 is unsatisfiable, then $P \equiv P_1$.*

Proof. *Property 1*: Suppose P is $(P_1$ AND $P_2)$ and P_1 or P_2 is unsatisfiable. Since AND is commutative (cf. Pérez et al. [128, Lemma 2.5], Schmidt et al. [140, Figure 2]), it is sufficient to consider the case where P_1 is unsatisfiable. In this case, $[\![P_1]\!]_G = \emptyset$ holds for any set of RDF triples G and, thus, $[\![P]\!]_G = (\emptyset \bowtie [\![P_2]\!]_G) = \emptyset$. Hence, P is unsatisfiable.

Property 2: Suppose P is $(P'$ FILTER $R)$ and P' is unsatisfiable. Then, $[\![P']\!]_G = \emptyset$ holds for any set of RDF triples G and, thus, $[\![P]\!]_G = \{\mu \in \emptyset \mid \mu \text{ satisfies } R\} = \emptyset$. Hence, P is unsatisfiable.

Property 3: Suppose P is $(P_1$ OPT $P_2)$ and P_1 is unsatisfiable. Then, $[\![P_1]\!]_G = \emptyset$ holds for any set of RDF triples G and, thus, $[\![P]\!]_G = (\emptyset ⟕ [\![P_2]\!]_G) = \emptyset$. Hence, P is unsatisfiable.

Property 4: Suppose P is $(P_1$ OPT $P_2)$ and P_2 is unsatisfiable. Then, $[\![P_2]\!]_G = \emptyset$ holds for any set of RDF triples G and, thus, $[\![P]\!]_G = ([\![P_1]\!]_G ⟕ \emptyset) = [\![P_1]\!]_G$. Hence, $P \equiv P_1$.

Property 5: Suppose P is $(P_1$ UNION $P_2)$ and P_1 is unsatisfiable. Then, $[\![P_1]\!]_G = \emptyset$ holds for any set of RDF triples G and, thus, $[\![P]\!]_G = (\emptyset \cup [\![P_2]\!]_G) = [\![P_2]\!]_G$. Hence, $P \equiv P_2$.

Property 6: Suppose P is $(P_1$ UNION $P_2)$ and P_2 is unsatisfiable. Since UNION is commutative (cf. Pérez et al. [128, Lemma 2.5], Schmidt et al. [140, Figure 2]), $P \equiv P_1$ follows from Property 5. ■

C.2. Monotonicity

We now focus on monotonicity of SPARQL expressions.

Definition C.2. A SPARQL expression P is *monotonic* if the following statement holds for any pair G_1, G_2 of (potentially infinite) sets of RDF triples: If $G_1 \subseteq G_2$, then $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$. A SPARQL expression is *non-monotonic* if it is not monotonic. □

The following property is a trivial corollary of Definitions C.1 and C.2.

Property C.1. *Unsatisfiable SPARQL expressions are monotonic, and non-monotonic SPARQL expressions are satisfiable.*

As in the case of satisfiability, we have an (ordinary) decision problem for monotonicity:

Problem:	MONOTONICITY(SPARQL)
Input:	a SPARQL expression P
Question:	Is P monotonic?

The monotonicity problem for SPARQL is also undecidable:

Proposition C.5. MONOTONICITY(SPARQL) *is undecidable.*

We aim to prove the proposition by reducing SATISFIABILITY(SPARQL) to MONOTONICITY(SPARQL). For this proof we first show the following lemma.

Lemma C.2. *Let P_1 and P_2 be SPARQL expressions such that (i) P_2 is non-monotonic and (ii)* $\text{vars}(P_1) \cap \text{vars}(P_2) = \emptyset$*. Then, P_1 is satisfiable if and only if SPARQL expression* $(P_1 \text{ AND } P_2)$ *is non-monotonic.*

Proof. *If:* Suppose SPARQL expression $(P_1 \text{ AND } P_2)$ is non-monotonic. To show that P_1 is satisfiable we use proof by contradiction, that is, we assume P_1 is unsatisfiable. In this case, SPARQL expression $(P_1 \text{ AND } P_2)$ is also unsatisfiable (cf. Proposition C.4, page 200). This is a contradiction because, due to its non-monotonicity, SPARQL expression $(P_1 \text{ AND } P_2)$ is satisfiable (cf. Property C.1).

Only if: Suppose SPARQL expression P_1 is satisfiable. Hence, there exists a set of RDF triples G such that $[\![P_1]\!]_G \neq \emptyset$. Since P_2 is non-monotonic, there also exist two sets of RDF triples G_1 and G_2 such that (i) $G_1 \subseteq G_2$ and (ii) $[\![P_2]\!]_{G_1} \nsubseteq [\![P_2]\!]_{G_2}$. Let valuation μ_1 be an arbitrary solution for P_1 in G (i.e., $\mu_1 \in [\![P_1]\!]_G$). Furthermore, let valuation μ_2 be a solution for P_2 in G_1 (i.e., $\mu_2 \in [\![P_2]\!]_{G_1}$) such that $\mu_2 \notin [\![P_2]\!]_{G_2}$. Since $\text{vars}(P_1) \cap \text{vars}(P_2) = \emptyset$, it holds that $\mu_1 \sim \mu_2$ and, thus, $\mu_1 \cup \mu_2 \in [\![(P_1 \text{ AND } P_2)]\!]_{G \cup G_1}$. However, from $\mu_2 \notin [\![P_2]\!]_{G_2}$ it follows that $\mu_1 \cup \mu_2 \notin [\![(P_1 \text{ AND } P_2)]\!]_{G \cup G_2}$. Hence, SPARQL expression $(P_1 \text{ AND } P_2)$ is non-monotonic. ■

Given Lemma C.2, we now prove Proposition C.5.

Proof of Proposition C.5. As mentioned before we show the undecidability of problem MONOTONICITY(SPARQL) by reducing SATISFIABILITY(SPARQL) to MONOTONICITY(SPARQL). Assume MONOTONICITY(SPARQL) were decidable. In this case we could answer SATISFIABILITY(SPARQL) for any SPARQL expression P as follows.

Let P' be a non-monotonic SPARQL expression such that $\text{vars}(P) \cap \text{vars}(P') = \emptyset$. For instance, P' could be the non-monotonic SPARQL expression that we shall discuss in Example C.2 (cf. page 203). If P and the selected non-monotonic SPARQL expression have variables in common, it is trivial to construct P' by renaming the variables in the selected expression. By using our (hypothetical) decider for MONOTONICITY(SPARQL) we decide whether SPARQL expression (P AND P') is monotonic or not. Now, it is easy to use Lemma C.2 for deciding the satisfiability of P.

Since SATISFIABILITY(SPARQL) is undecidable (cf. Proposition C.1, page 195) we have a contradiction and, thus, MONOTONICITY(SPARQL) cannot be decidable. ■

Similar to satisfiability, monotonicity is also relevant for certain results in this dissertation. For instance, we shall see that in many cases LD machine computability of Linked Data queries depends on monotonicity. Hence, we now identify fragments of SPARQL for which we can show monotonicity (or non-monotonicity).

We already know that unsatisfiable SPARQL expressions are monotonic (see Property C.1). The following proposition shows monotonicity for all expressions *without* OPT:

Proposition C.6. *SPARQL expressions without* OPT *are monotonic.*

Proof. Let G_1, G_2 be an arbitrary pair of (potentially infinite) sets of RDF triples such that $G_1 \subseteq G_2$. W.l.o.g., we prove Proposition C.6 by showing that $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$ holds for all SPARQL expressions P without OPT. We use induction on the structure of possible SPARQL expressions for this proof.

Base case: Suppose SPARQL expression P is a triple pattern tp. We show $\mu \in [\![P]\!]_{G_2}$ for all valuations $\mu \in [\![P]\!]_{G_1}$. Let valuation μ be an arbitrary solution for P in G_1; i.e., $\mu \in [\![P]\!]_{G_1}$. Then, $\text{dom}(\mu) = \text{vars}(tp)$ and $\mu[tp] \in G_1$. Since $G_1 \subseteq G_2$, it holds that $\mu[tp] \in G_2$ and, thus, $\mu \in [\![P]\!]_{G_2}$.

Induction step: By induction, assume two SPARQL expressions P_1 and P_2 such that $[\![P_1]\!]_{G_1} \subseteq [\![P_1]\!]_{G_2}$ and $[\![P_2]\!]_{G_1} \subseteq [\![P_2]\!]_{G_2}$. Then, for any SPARQL expression P that can be constructed (using P_1, P_2, and the operators AND, FILTER, UNION), we show that $\mu \in [\![P]\!]_{G_2}$ holds for all valuations $\mu \in [\![P]\!]_{G_1}$. We distinguish the following cases:

- P is (P_1 AND P_2). For any $\mu \in [\![P]\!]_{G_1} = [\![P_1]\!]_{G_1} \bowtie [\![P_2]\!]_{G_1}$ there exist valuations $\mu_l \in [\![P_1]\!]_{G_1}$ and $\mu_r \in [\![P_2]\!]_{G_1}$ such that $\mu = \mu_l \cup \mu_r$ and $\mu_l \sim \mu_r$. By induction we have $\mu_l \in [\![P_1]\!]_{G_2}$ and $\mu_r \in [\![P_2]\!]_{G_2}$. Thus, $\mu \in [\![P_1]\!]_{G_2} \bowtie [\![P_2]\!]_{G_2} = [\![P]\!]_{G_2}$.
- P is (P' FILTER R). For any $\mu \in [\![P]\!]_{G_1}$ it holds that (i) $\mu \in [\![P']\!]_{G_1}$ and (ii) μ satisfies filter condition R. By induction we have $\mu \in [\![P']\!]_{G_2}$. Therefore, it holds that $\mu \in \{\mu' \in [\![P']\!]_{G_2} \mid \mu' \text{ satisfies } R\} = [\![P]\!]_{G_2}$.
- P is (P_1 UNION P_2). For any $\mu \in [\![P]\!]_{G_1} = [\![P_1]\!]_{G_1} \cup [\![P_2]\!]_{G_1}$ we have the following two (nonexclusive) cases:
  1. $\mu \in [\![P_1]\!]_{G_1}$. By induction we also have $\mu \in [\![P_1]\!]_{G_2}$.
  2. $\mu \in [\![P_2]\!]_{G_1}$. By induction we also have $\mu \in [\![P_2]\!]_{G_2}$.

  Hence, in both cases, $\mu \in [\![P_1]\!]_{G_2} \cup [\![P_2]\!]_{G_2} = [\![P]\!]_{G_2}$.

■

While all SPARQL expressions without OPT are monotonic, we now focus on expressions with OPT. These may not be monotonic as the following example demonstrates.

Example C.2. Let G_1 and G_2 be the following sets of RDF triples:

$$G_1 = \{(\mathsf{offer1.1}, \mathsf{offeredBy}, \mathsf{vendor1})\} \quad \text{and} \quad G_2 = G_1 \cup \{(\mathsf{vendor1}, \mathsf{name}, \text{"Vendor1"})\},$$

and let P be the SPARQL expression $((\mathsf{offer1.1}, \mathsf{offeredBy}, ?v)\ \mathsf{OPT}\ (?v, \mathsf{name}, ?vn))$. It holds that $[\![P]\!]_{G_1} = \{\mu_1\}$ and $[\![P]\!]_{G_2} = \{\mu_2\}$ where μ_1 and μ_2 are the valuations $\{?v \to \mathsf{vendor1}\}$ and $\{?v \to \mathsf{vendor1}, ?vn \to \text{"Vendor1"}\}$, respectively. Hence, $[\![P]\!]_{G_1} \nsubseteq [\![P]\!]_{G_2}$ and, thus, P is non-monotonic. □

However, some expressions with OPT are monotonic:

Example C.3. It can be easily seen that the following SPARQL expression is monotonic: $((?o, \mathsf{offeredBy}, \mathsf{vendor1})\ \mathsf{OPT}\ (\mathsf{vendor1}, \mathsf{name}, \text{"Vendor1"}))$. We shall provide a formal proof shortly (see Proposition C.7 below). □

While Examples C.2 and C.3 verify that (satisfiable) SPARQL expressions with OPT are either monotonic or non-monotonic, the following result provides us with an (incomplete) list of criteria for deciding about the monotonicity of such expressions.

Proposition C.7. *Let P be a SPARQL expression $(P_1\ \mathsf{OPT}\ P_2)$.*

1. *If $\mathrm{vars}(P_1) = \emptyset$, it holds: P is monotonic if P_1 and P_2 are monotonic.*
2. *If $\mathrm{vars}(P_2) = \emptyset$, it holds: P is monotonic if and only if P_1 is monotonic.*
3. *If $\mathrm{vars}(P_1) \neq \emptyset$ and $\mathrm{vars}(P_2) \neq \emptyset$, it holds: P is monotonic if (i) P_1 is unsatisfiable or (ii) P_1 is monotonic and P_2 is unsatisfiable.*

We prove the proposition by discussing its three claims one after another:

Proof of Proposition C.7, Claim 1. Let P be a SPARQL expression $(P_1\ \mathsf{OPT}\ P_2)$ where (i) $\mathrm{vars}(P_1) = \emptyset$ and (ii) P_1 and P_2 are monotonic.

We distinguish two cases w.r.t. the evaluation of subexpression P_1 over an arbitrary (potentially infinite) set of RDF triples G: (a) $[\![P_1]\!]_G = \emptyset$ and (b) $[\![P_1]\!]_G \neq \emptyset$. To show claim 1 we first discuss the effect of both cases on $[\![P]\!]_G$:

(a) If $[\![P_1]\!]_G = \emptyset$, then $[\![P]\!]_G = \emptyset$ (irrespective of the result of $[\![P_2]\!]_G$).

(b) If $[\![P_1]\!]_G \neq \emptyset$, then $[\![P_1]\!]_G = \{\mu_\emptyset\}$ where $\mu_\emptyset$ is the empty valuation ($\mathrm{dom}(\mu_\emptyset) = \emptyset$). Then, $[\![P]\!]_G = \{\mu_\emptyset\} ⟕ [\![P_2]\!]_G = \{\mu_\emptyset\} \bowtie [\![P_2]\!]_G = [\![P_2]\!]_G$, because $\mu_\emptyset$ is compatible with any valuation.

We now discuss what cases (a) and (b) mean for the monotonicity of P. Let G_1, G_2 be an arbitrary pair of (potentially infinite) sets of RDF triples such that $G_1 \subseteq G_2$. We distinguish the following cases for G_1 and G_2 w.r.t. (a) and (b):

- If we have case (a) for G_1 and case (a) for G_2, then it holds that $[\![P_1]\!]_{G_1} = [\![P_1]\!]_{G_2}$ and $[\![P]\!]_{G_1} = [\![P]\!]_{G_2}$. In this case P_1 and P satisfy the requirement for monotonicity (regardless of whether P_2 is monotonic or non-monotonic).
- If we have case (a) for G_1 and case (b) for G_2, then it holds that $[\![P_1]\!]_{G_1} \subseteq [\![P_1]\!]_{G_2}$ and $[\![P]\!]_{G_1} \subseteq [\![P]\!]_{G_2}$. In this case P_1 and P satisfy the requirement for monotonicity (regardless of whether P_2 is monotonic or non-monotonic).
- If we have case (b) for G_1 and case (b) for G_2, then $[\![P_1]\!]_{G_1} = [\![P_1]\!]_{G_2}$. In this case P_1 satisfies the requirement for monotonicity. However, P satisfies the requirement for monotonicity only if P_2 is monotonic. The latter holds according to our antecedent.
- If we have case (b) for G_1 and case (a) for G_2, then $[\![P_1]\!]_{G_1} \supset [\![P_1]\!]_{G_2}$. In this case, P_1 does not satisfy the requirement for monotonicity. Hence, this case is not relevant because we assume P_1 is monotonic.

In all three relevant cases SPARQL expression P is monotonic. ■

Proof of Proposition C.7, Claim 2. Let P be a SPARQL expression of the form $(P_1 \text{ OPT } P_2)$ where $\mathrm{vars}(P_2) = \emptyset$. We distinguish two cases w.r.t. the evaluation of subexpression P_2 over an arbitrary (potentially infinite) set of RDF triples G: (a) $[\![P_2]\!]_G = \emptyset$ and (b) $[\![P_2]\!]_G \neq \emptyset$. To show claim 2 we first discuss the effect of both cases on $[\![P]\!]_G$:

(a) If $[\![P_2]\!]_G = \emptyset$, then $[\![P]\!]_G = [\![P_1]\!]_G ⟕ \emptyset = ([\![P_1]\!]_G \bowtie \emptyset) \cup ([\![P_1]\!]_G \setminus \emptyset) = [\![P_1]\!]_G$.

(b) If $[\![P_2]\!]_G \neq \emptyset$, it holds that $[\![P_2]\!]_G = \{\mu_\emptyset\}$ where $\mu_\emptyset$ is the empty valuation with $\mathrm{dom}(\mu_\emptyset) = \emptyset$. Then, $[\![P]\!]_G = [\![P_1]\!]_G ⟕ \{\mu_\emptyset\} = [\![P_1]\!]_G \bowtie \{\mu_\emptyset\} = [\![P_1]\!]_G$ because $\mu_\emptyset$ is compatible with any valuation.

In both cases we have $[\![P]\!]_G = [\![P_1]\!]_G$ (for an arbitrary set of RDF triples G). Hence, P is semantically equivalent to P_1 and, thus, P has the same monotonicity as P_1. ■

Proof of Proposition C.7, Claim 3. Let P be a SPARQL expression of the form $(P_1 \text{ OPT } P_2)$ where $\mathrm{vars}(P_1) \neq \emptyset$ and $\mathrm{vars}(P_2) \neq \emptyset$.

First, we assume subexpression P_1 is unsatisfiable. In this case, P is also unsatisfiable (cf. Proposition C.4 on page 200). Then, since P is unsatisfiable, P is trivially monotonic (cf. Property C.1).

We now assume P_1 is monotonic and P_2 is unsatisfiable. Due to the latter we have $[\![P_2]\!]_G = \emptyset$ and, thus, $[\![P]\!]_G = [\![P_1]\!]_G ⟕ \emptyset = ([\![P_1]\!]_G \bowtie \emptyset) \cup ([\![P_1]\!]_G \setminus \emptyset) = [\![P_1]\!]_G$, for any set of RDF triples G. Hence, P is semantically equivalent to P_1. Then, the monotonicity of P follows from the monotonicity of P_1. ■

C.3. Bounded Satisfiability and Unbounded Satisfiability

Section 2.1.2 introduces bounded satisfiability and unbounded satisfiability as particular types of satisfiability that are relevant for some of the results in this dissertation (cf. page

22ff). While Section 2.1.2 introduces these notions of satisfiability for Linked Data queries (see, in particular, Definition 2.8, page 24), we now define and discuss bounded satisfiability and unbounded satisfiability for SPARQL expressions.

Definition C.3. A SPARQL expression P is *unboundedly satisfiable* if, for any natural number $k \in \{0, 1, 2, ...\}$, there exists a set of RDF triples G such that $|[\![P]\!]_G| > k$. A SPARQL expression P is *boundedly satisfiable* if it is satisfiable but *not* unboundedly satisfiable. □

Example C.4. Consider triple patterns $tp_1 = (u_1^*, u_2^*, ?v)$ and $tp_2 = (u_1^*, u_2^*, u_3^*)$ where $u_1^*, u_2^*, u_3^* \in \mathcal{U}$ and $?v \in \mathcal{V}$.

We show that tp_1 is unboundedly satisfiable as follows: The set of all potentially matching triples for tp_1 is $\mathcal{T}_{tp_1} = \{u_1^*\} \times \{u_2^*\} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$. This set is infinite because $\mathcal{U}$, $\mathcal{B}$, and $\mathcal{L}$ are infinite, respectively. Then, for any $k \in \{0, 1, 2, ...\}$ we select a subset $G_k \subset \mathcal{T}_{tp_1}$ of size k+1; i.e., $|G_k| = k + 1$. It is easy to see that $|[\![tp_1]\!]_{G_k}| = k + 1 > k$. Due to the infiniteness of $\mathcal{T}_{tp_1}$, such a subset exists for all $k \in \{0, 1, 2, ...\}$.

Triple pattern tp_2, in contrast, is a trivial example of a boundedly satisfiable SPARQL expression: Since $\mathrm{vars}(tp_2) = \emptyset$, tp_2 is an RDF triple. Hence, any possible set of RDF triples may contain at most a single matching triple for tp_2, namely (u_1^*, u_2^*, u_3^*). For any such set G (for which $(u_1^*, u_2^*, u_3^*) \in G$), $[\![tp_2]\!]_G = \{\mu_\emptyset\}$ holds with $\mu_\emptyset$ being the empty valuation. For any other set of RDF triples the query result contains no solution at all, that is $[\![tp_2]\!]_{G'} = \emptyset$ for any set of RDF triples G' for which $(u_1^*, u_2^*, u_3^*) \notin G'$. Therefore, $k = 1$ is an upper bound for the number of possible solutions that may be computed for tp_2; more precisely, there does not exist a set of RDF triples G such that $|[\![tp_2]\!]_G| > 1$.

Another, less trivial example of a boundedly satisfiable SPARQL expression is the expression $\big((u_1^*, u_2^*, ?v)\ \textsf{FILTER}\ (?v = u_1^* \vee ?v = u_3^*)\big)$. This expression contains the aforementioned triple pattern tp_1 as a subexpression. Although the relevant set $\mathcal{T}_{tp_1}$ of potentially matching triples for tp_1 is infinite (see above), there exist only two valuations in $[\![tp_1]\!]_{\mathcal{T}_{tp_1}}$ that satisfy the filter condition, namely $\mu = \{?v \to u_1^*\} \in [\![tp_1]\!]_{\mathcal{T}_{tp_1}}$ and $\mu' = \{?v \to u_3^*\} \in [\![tp_1]\!]_{\mathcal{T}_{tp_1}}$. Hence, μ and μ' are the only solutions for the given SPARQL expression in $\mathcal{T}_{tp_1}$. Similarly, in any other set of RDF triples there exist at most these two solutions. □

From Definition C.3 it follows trivially that any unboundedly satisfiable SPARQL expression is also satisfiable. Hence, our notions of bounded satisfiability and unbounded satisfiability partition the set of satisfiable SPARQL expressions into two, non-overlapping subsets. We emphasize that an expression that is *not* boundedly satisfiable is not necessarily unboundedly satisfiable; it may also be unsatisfiable (vice versa for expressions that are *not* unboundedly satisfiable).

Similar to the satisfiability problem and to the monotonicity problem discussed in the previous sections, we have (ordinary) decision problems for unbounded satisfiability and for bounded satisfiability.

Problem:	BoundedSatisfiability(SPARQL)
Input:	a SPARQL expression P
Question:	Is P boundedly satisfiable?

Problem:	UnboundedSatisfiability(SPARQL)
Input:	a SPARQL expression P
Question:	Is P unboundedly satisfiable?

Proposition C.8. BoundedSatisfiability(SPARQL) *and* UnboundedSatisfiability(SPARQL) *are undecidable.*

Proof. We first show that BoundedSatisfiability(SPARQL) is undecidable by reducing Satisfiability(SPARQL) to BoundedSatisfiability(SPARQL). For such a reduction we need a (Turing) computable function that maps any possible input for Satisfiability(SPARQL) to an input for BoundedSatisfiability(SPARQL). We use the identity function.

Assume BoundedSatisfiability(SPARQL) is decidable. Then, we may answer Satisfiability(SPARQL) for any SPARQL expression P as follows.

Let $u^* \in \mathcal{U}$ be a URI that is not mentioned in P (neither in any triple pattern nor in any filter condition); and let SPARQL expression P' be triple pattern (u^*, u^*, u^*). P' is boundedly satisfiable because, for any possible set of RDF triples G, either $[\![P']\!]_G = \emptyset$ holds or $[\![P']\!]_G = \{\mu_\emptyset\}$ holds (where $\mu_\emptyset$ is the empty valuation with $\mathrm{dom}(\mu_\emptyset) = \emptyset$). We use our (hypothetical) decider for BoundedSatisfiability(SPARQL) to decide whether SPARQL expression (P union P') is boundedly satisfiable.

- *Case 1:* If (P union P') is boundedly satisfiable, then P cannot be unboundedly satisfiable. Thus, P is either boundedly satisfiable (and, thus, satisfiable) or unsatisfiable. To determine which of both cases holds, we again use our (hypothetical) decider for BoundedSatisfiability(SPARQL) and simply ask whether P is boundedly satisfiable. Depending on the result, we conclude that P is satisfiable (if it is boundedly satisfiable) or unsatisfiable (if it is not boundedly satisfiable).

- *Case 2:* If (P union P') is *not* boundedly satisfiable, then (P union P') could be either unboundedly satisfiable or unsatisfiable. However, since P' is (boundedly) satisfiable, (P union P') cannot be unsatisfiable. Hence, (P union P') must be unboundedly satisfiable. Then, we may again use the fact that P' is boundedly satisfiable to conclude P must be unboundedly satisfiable (in order to make (P union P') unboundedly satisfiable). As a result we have that P is satisfiable.

Based on the assumption that BoundedSatisfiability(SPARQL) is decidable, the procedure outlined above decides Satisfiability(SPARQL). Therefore, given that Satisfiability(SPARQL) is undecidable (cf. Proposition C.1, page 195) we have a contradiction and, thus, BoundedSatisfiability(SPARQL) cannot be decidable.

We now focus on UnboundedSatisfiability(SPARQL). To show that UnboundedSatisfiability(SPARQL) is undecidable, we reduce Satisfiability(SPARQL) to UnboundedSatisfiability(SPARQL) (using the identity function again). That is, we assume the existence of a decider for UnboundedSatisfiability(SPARQL) and use this decider to answer Satisfiability(SPARQL) for any SPARQL expression P as follows: Let SPARQL expression P'' be a triple pattern $(u^*, u^*, ?v)$ such that (i) $u^* \in \mathcal{U}$

is a URI that is not mentioned in P (as above) and (ii) $?v$ is a fresh variable not used in P; i.e., $?v \in \mathcal{V} \setminus \mathrm{vars}(P)$. P'' is unboundedly satisfiable (cf. Proposition C.9 below). We note that any possible solution for P'' is compatible with any potential solution for P (independent of the queried set of RDF triples) because both expressions, P and P'', do not have any variable in common. Then, given the unbounded satisfiability of P'', the SPARQL expression (P AND P'') cannot be boundedly unsatisfiable; instead, it is either unboundedly satisfiable or it is unsatisfiable. To determine which of these cases holds, we use our (hypothetical) decider for UNBOUNDEDSATISFIABILITY(SPARQL).

- *Case 1:* If (P AND P'') is unboundedly satisfiable (and, thus, satisfiable), then, by Proposition C.4 (cf. page 200), we conclude that P is satisfiable.
- *Case 2:* If (P AND P'') is unsatisfiable (because it is not unboundedly satisfiable and it cannot be boundedly satisfiable), then, due to the (unbounded) satisfiability of P'' and by using Proposition C.4 (cf. page 200), we conclude that P is unsatisfiable.

By using the outlined procedure, we may decide SATISFIABILITY(SPARQL) if problem UNBOUNDEDSATISFIABILITY(SPARQL) is decidable. Thus, given that SATISFIABILITY(SPARQL) is undecidable, UNBOUNDEDSATISFIABILITY(SPARQL) cannot be decidable either. ■

Given that both BOUNDEDSATISFIABILITY(SPARQL) and UNBOUNDEDSATISFIABILITY(SPARQL) are undecidable, we want to identify a fragment of SPARQL for which we can show unbounded satisfiability and bounded satisfiability. We focus on SPARQL expressions that consist only of triple patterns and AND:

Proposition C.9. *Any SPARQL expression P without* FILTER, OPT, *and* UNION *is unboundedly satisfiable if and only if* $\mathrm{vars}(P) \neq \emptyset$.

To prove the proposition we first show the following necessary condition for unbounded satisfiability:

Lemma C.3. *If a SPARQL expression P is unboundedly satisfiable, then* $\mathrm{vars}(P) \neq \emptyset$.

Proof of Lemma C.3. Let P be an arbitrary unboundedly satisfiable SPARQL expression. Furthermore, let G be a set of RDF triples such that $|[\![P]\!]_G| > 1$. Such a set exists because P is unboundedly satisfiable. Then, there exist (at least) two distinct solutions $\mu_1 \in [\![P]\!]_G$ and $\mu_2 \in [\![P]\!]_G$. Because of $\mu_1 \neq \mu_2$ it holds that $\mathrm{dom}(\mu_1) \neq \mathrm{dom}(\mu_2)$ or $\mu_1 \not\sim \mu_2$ (or both). We discuss both cases in the following.

The first case, $\mathrm{dom}(\mu_1) \neq \mathrm{dom}(\mu_2)$, is only possible if $\mathrm{dom}(\mu_1) \neq \emptyset$ or $\mathrm{dom}(\mu_2) \neq \emptyset$ (or both). W.l.o.g., let $\mathrm{dom}(\mu_1) \neq \emptyset$. Since $\mu_1 \in [\![P]\!]_G$ and $\mathrm{dom}(\mu_1) \neq \emptyset$, it is trivial to show (by induction on the possible structure of SPARQL expression P) that $\mathrm{vars}(P) \neq \emptyset$.

In the second case, $\mu_1 \not\sim \mu_2$, there exists a common variable $?v \in \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2)$ such that $\mu_1(?v) \neq \mu_2(?v)$. This shows that $\mathrm{dom}(\mu_1) \neq \emptyset$ and $\mathrm{dom}(\mu_2) \neq \emptyset$. As in the first case, we may use $\mu_1 \in [\![P]\!]_G$ and $\mathrm{dom}(\mu_1) \neq \emptyset$ to show $\mathrm{vars}(P) \neq \emptyset$. ■

In addition to Lemma C.3, we need the following lemma to prove Proposition C.9.

Lemma C.4. *Let $\mathcal{T}^{\mathcal{UL}} = \mathcal{U} \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L})$ be the infinite set of all RDF triples that contain no blank node; and let P be a SPARQL expression without* FILTER*,* OPT*, and* UNION*. For any valuation μ with $\mathrm{dom}(\mu) \subseteq \mathrm{vars}(P)$ there exists a pair (G', μ') with $G' \subseteq \mathcal{T}^{\mathcal{UL}}$ and $\mu' \in [\![P]\!]_{G'}$ such that $\mu' \sim \mu$.*

Proof of Lemma C.4. We prove Lemma C.4 by induction on the possible structure of SPARQL expression P.

Base case: Suppose P is a triple pattern tp. Let μ be an arbitrary valuation such that $\mathrm{dom}(\mu) \subseteq \mathrm{vars}(tp)$. It suffices to show that there exists a pair (G', μ') with $G' \subseteq \mathcal{T}^{\mathcal{UL}}$ and $\mu' \in [\![tp]\!]_{G'}$ such that $\mu' \sim \mu$. It can be easily seen that there exists an RDF triple in $\mathcal{T}^{\mathcal{UL}}$ such that this triple is a matching triple for triple pattern $tp' = \mu[tp]$. Let $t \in \mathcal{T}^{\mathcal{UL}}$ be such a triple. Since $tp' = \mu[tp]$, t is also a matching triple for triple pattern tp. Then, let μ' be a valuation such that $\mathrm{dom}(\mu') = \mathrm{vars}(tp)$ and $\mu'[tp] = t$. Due to this construction of μ' we have $\mu' \in [\![tp]\!]_{G'}$ where $G' = \{t\} \subseteq \mathcal{T}^{\mathcal{UL}}$. Hence, for valuation μ the pair (G', μ') satisfies the condition in Lemma C.4.

Induction step: Let P_1 and P_2 be SPARQL expressions without FILTER, OPT, and UNION such that Lemma C.4 holds for both, P_1 and P_2. Since P_1 and P_2 contain no OPT, they are monotonic (cf. Proposition C.6, page 202). We now show that Lemma C.4 also holds for SPARQL expression $(P_1 \text{ AND } P_2)$ (which is also monotonic).

Let μ be an arbitrary valuation such that $\mathrm{dom}(\mu) \subseteq \mathrm{vars}((P_1 \text{ AND } P_2))$. It suffices to show that there exists a pair (G', μ') with $G' \subseteq \mathcal{T}^{\mathcal{UL}}$ and $\mu' \in [\![(P_1 \text{ AND } P_2)]\!]_{G'}$ such that $\mu' \sim \mu$.

Let μ_1 and μ_2 be valuations such that (i) $\mathrm{dom}(\mu_i) = \mathrm{dom}(\mu) \cap \mathrm{vars}(P_i)$ for all $i \in \{1, 2\}$, and (ii) $\mu = \mu_1 \cup \mu_2$. By induction, there exist pairs (G'_1, μ'_1) and (G'_2, μ'_2) such that for each $i \in \{1, 2\}$ it holds that (i) $G'_i \subseteq \mathcal{T}^{\mathcal{UL}}$, (ii) $\mu'_i \in [\![P_i]\!]_{G'_i}$, and (iii) $\mu'_i \sim \mu$. We use these pairs to construct $G' = G'_1 \cup G'_2$ and $\mu' = \mu'_1 \cup \mu'_2$. Since P_1 and P_2 are monotonic, $\mu'_i \in [\![P_i]\!]_{G'}$ for all $i \in \{1, 2\}$. However, it also holds that $\mu' \in [\![(P_1 \text{ AND } P_2)]\!]_{G'}$ and, since $\mu'_i \sim \mu$ for all $i \in \{1, 2\}$, we also have $\mu' \sim \mu$. Therefore, (G', μ') is the pair that satisfies the condition from Lemma C.4 for valuation μ. ■

Now we are ready to prove Proposition C.9.

Proof of Proposition C.9. Let P be a SPARQL expression without FILTER, OPT, and UNION. We have to show that P is unboundedly satisfiable if and only if $\mathrm{vars}(P) \neq \emptyset$. Since the *only if* part follows directly from Lemma C.3, we only have to show the *if* part. We use an induction on the possible structure of SPARQL expression P.

Base case: Suppose P is a triple pattern. W.l.o.g., let $tp = (u_1, u_2, ?v) \in \mathcal{U} \times \mathcal{U} \times \mathcal{V}$ be this triple pattern. It can be easily seen that the following argument applies to any other type of triple patterns that contain at least one variable, that is, any triple pattern in $((\mathcal{V} \cup \mathcal{U}) \times (\mathcal{V} \cup \mathcal{U}) \times (\mathcal{V} \cup \mathcal{U} \cup \mathcal{L})) \cap (\mathcal{U} \times \mathcal{U} \times \mathcal{U}) \cap (\mathcal{U} \times \mathcal{U} \times \mathcal{L})$.

Let $\mathcal{T}_{tp} = \{u_1\} \times \{u_2\} \times \mathcal{U}$. Notice, we deliberately exclude the sets of all blank nodes $\mathcal{B}$ and of all literals $\mathcal{L}$ from this definition (if we would use a triple pattern $tp' = (?v, u, c) \in \mathcal{V} \times \mathcal{U} \times \mathcal{L}$ instead of $tp = (u_1, u_2, ?v)$, then the constructed set would be $\mathcal{T}_{tp'} = \mathcal{U} \times \{u\} \times \{c\}$). It is easy to see that any RDF triple in $\mathcal{T}_{tp}$ is a matching triple for

tp and $\mathcal{T}_{tp}$ is infinite because $\mathcal{U}$ is infinite. Thus, for any natural number $k \in \{0, 1, 2, ...\}$ there exists a subset $G \subseteq \mathcal{T}_{tp}$ such that $\left|[\![tp]\!]_G\right| > k$. Hence, tp is unboundedly satisfiable.

Induction step: Since Proposition C.9 excludes FILTER, OPT, and UNION, we only have to discuss the case where P is a SPARQL expression $(P_1 \text{ AND } P_2)$ (with $\text{vars}(P) \neq \emptyset$). In this case, P, P_1, and P_2 are monotonic (cf. Proposition C.6, page 202). Since $\text{vars}(P) \neq \emptyset$, there exists an $i \in \{1, 2\}$ such that $\text{vars}(P_i) \neq \emptyset$. W.l.o.g., let $i = 1$. Then, by induction, P_1 is unboundedly satisfiable. We now show that P is also unboundedly satisfiable.

Let $k \in \{0, 1, 2, ...\}$ be an arbitrary natural number. Since P_1 is unboundedly satisfiable, there exists a set of RDF triples G_1 such that $\left|[\![P_1]\!]_{G_1}\right| > k$. Let G'_1 be such a set and let $k' = \left|[\![P_1]\!]_{G'_1}\right|$; hence, $k' > k$. For each (of the k' different) $\mu \in [\![P_1]\!]_{G'_1}$ we write $j(\mu)$ to denote the valuation for which $\text{dom}(\mu^*) = \text{vars}(P_1) \cap \text{vars}(P_2)$ and $\mu^* \sim \mu$ hold. Furthermore, for each $\mu \in [\![P_1]\!]_{G'_1}$ we write $J(\mu)$ to denote an arbitrary pair (G', μ') for which it holds that (i) $G' \subseteq \mathcal{T}^{\mathcal{UL}}$, (ii) $\mu' \in [\![P_2]\!]_{G'}$, and (iii) $\mu' \sim j(\mu)$. Such a pair exists (cf. Lemma C.4).

Then, for each $\mu \in [\![P_1]\!]_{G'_1}$ with $J(\mu) = (G', \mu')$, we have $\mu \cup \mu' \in [\![(P_1 \text{ AND } P_2)]\!]_{G'_1 \cup G'}$ because (i) $\text{dom}(\mu) \cup \text{dom}(\mu') = \text{vars}(P_1) \cup \text{vars}(P_2) = \text{vars}(P)$, (ii) $\mu' \sim j(\mu) \sim \mu$, and (iii) P_1, P_2, and $(P_1 \text{ AND } P_2)$ are monotonic.

Let $G = G'_1 \cup \bigcup_{G' \in \mathcal{G}}(G')$ where $\mathcal{G} = \{G' \mid \mu \in [\![P_1]\!]_{G'_1} \text{ and } J(\mu) = (G', \mu')\}$. Then, for each $\mu \in [\![P_1]\!]_{G'_1}$ with $J(\mu) = (G', \mu')$, we also have $\mu \cup \mu' \in [\![(P_1 \text{ AND } P_2)]\!]_G$. Because of $\left|[\![P_1]\!]_{G'_1}\right| = k'$, we have $\left|[\![(P_1 \text{ AND } P_2)]\!]_G\right| \geq k'$ and, since $k' > k$, it holds that $\left|[\![(P_1 \text{ AND } P_2)]\!]_G\right| > k$. This shows that P is unboundedly satisfiable. ■

The following result is a trivial consequence of Proposition C.9 (and the fact that any SPARQL expression without FILTER is satisfiable).

Corollary C.1. *Any SPARQL expression P without* FILTER*,* OPT*, and* UNION *is boundedly satisfiable if and only if* $\text{vars}(P) = \emptyset$.

Proof. The corollary follows immediately from the satisfiability of SPARQL expressions without FILTER (cf. Proposition C.2, page 197) and Proposition C.9. ■

Proposition C.9 and Corollary C.1 cover the fragment of SPARQL that uses only triple patterns and AND. In addition to this fragment, Example C.4 (cf. page 205) introduces a form of SPARQL expressions with FILTER for which bounded satisfiability is easy to prove. We leave a more extensive discussion of SPARQL expressions with FILTER, OPT, or UNION as future work.

D. Supplementary Information about the Experiments

D.1. Queries for the WWW-Based Experiment

For the queries enumerated in this section we assume the following prefix definitions:

```
PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:    <http://www.w3.org/2002/07/owl#>
PREFIX dct:    <http://purl.org/dc/terms/>
PREFIX dbowl:  <http://dbpedia.org/ontology/>
PREFIX dbprop: <http://dbpedia.org/property/>
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
PREFIX swc:    <http://data.semanticweb.org/ns/swc/ontology#>
PREFIX swrc:   <http://swrc.ontoware.org/ontology#>
PREFIX drugbank:  <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX diseasome: <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/>
PREFIX dailymed:  <http://www4.wiwiss.fu-berlin.de/dailymed/resource/dailymed/>
PREFIX tcm:    <http://purl.org/net/tcm/tcm.lifescience.ntu.edu.tw/>
PREFIX eurostat:  <http://www4.wiwiss.fu-berlin.de/eurostat/resource/eurostat/>
PREFIX gn:     <http://www.geonames.org/ontology#>
```

Query WQ1.

Query semantics: c_{Match}-semantics

BGP:
```
?paper swc:isPartOf
     <http://data.semanticweb.org/conference/iswc/2008/poster_demo_proceedings> .
?paper swrc:author ?p .
?p rdfs:label ?n .
```

Seed URI: `http://data.semanticweb.org/conference/iswc/2008/poster_demo_proceedings`

Query WQ2.

Query semantics: c_{Match}-semantics

BGP:
```
?proceedings swc:relatedToEvent
                        <http://data.semanticweb.org/conference/eswc/2010> .
?paper swc:isPartOf ?proceedings .
?paper swrc:author ?p .
```

Seed URI: `http://data.semanticweb.org/conference/eswc/2010`

Query WQ3.

Query semantics: c_{Match}-semantics

BGP:
```
?paper swc:isPartOf
     <http://data.semanticweb.org/conference/iswc/2008/poster_demo_proceedings> .
?paper swrc:author ?p .
?p owl:sameAs ?x .
?p rdfs:label ?n .
```

Seed URI: `http://data.semanticweb.org/conference/iswc/2008/poster_demo_proceedings`

Query WQ4.

Query semantics: c_{Match}-semantics

BGP:
```
?role swc:isRoleAt <http://data.semanticweb.org/conference/eswc/2010> .
?role swc:heldBy ?p .
?paper swrc:author ?p .
?paper swc:isPartOf ?proceedings .
?proceedings swc:relatedToEvent
                  <http://data.semanticweb.org/conference/eswc/2010> .
```

Seed URI: `http://data.semanticweb.org/conference/eswc/2010`

Query WQ5.

Query semantics: c_{Match}-semantics

BGP:
```
?a dbowl:artist <http://dbpedia.org/resource/Michael_Jackson> .
?a rdf:type dbowl:Album .
?a foaf:name ?n .
```

Seed URIs: `http://dbpedia.org/resource/Michael_Jackson`, `dbowl:Album`

Query WQ6.

Query semantics: c_{Match}-semantics

BGP:
```
?director dbowl:nationality <http://dbpedia.org/resource/Italy> .
?film dbprop:director ?director.
?film owl:sameAs ?x .
```

Seed URI: `http://dbpedia.org/resource/Italy`

Query WQ7.

Query semantics: c_{Match}-semantics

BGP:
```
<http://sws.geonames.org/2921044/> gn:childrenFeatures ?c .
?x gn:parentFeature <http://sws.geonames.org/2921044/> .
?x gn:name ?n .
```

Seed URI: `http://sws.geonames.org/2921044/`

Query WQ8.

Query semantics: c_{Match}-semantics

BGP:
```
?drug drugbank:drugCategory
 <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugcategory/micronutrient> .
?drug drugbank:casRegistryNumber ?id .
?drug owl:sameAs ?s .
?s foaf:name ?o .
?s dct:subject ?sub .
```

Seed URI: `http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugcategory/micronutrient`

Query WQ9.

Query semantics: c_{Match}-semantics

BGP:
```
?x dct:subject
        <http://dbpedia.org/resource/Category:FIFA_World_Cup-winning_countries> .
?p dbprop:managerclubs ?x .
?p foaf:name "Luiz Felipe Scolari"@en .
```

Seed URI: `http://dbpedia.org/resource/Category:FIFA_World_Cup-winning_countries`

Query WQ10.

Query semantics: c_{Match}-semantics

BGP:
```
?n dct:subject <http://dbpedia.org/resource/Category:Chancellors_of_Germany> .
?n owl:sameAs ?p2 .
?p2 <http://data.nytimes.com/elements/latest_use> ?u .
```

Seed URI: `http://dbpedia.org/resource/Category:Chancellors_of_Germany`

Query WQ11.

Query semantics: c_{Match}-semantics

BGP:
```
?x dbowl:team <http://dbpedia.org/resource/Eintracht_Frankfurt> .
?x rdfs:label ?y .
?x dbowl:birthDate ?d .
?x dbowl:birthPlace ?p .
?p rdfs:label ?l .
```

Seed URI: `http://dbpedia.org/resource/Eintracht_Frankfurt`

Query WQ12.

Query semantics: c_{Match}-semantics

BGP:
```
<http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/DB01273>
                                    drugbank:possibleDiseaseTarget ?disease .
?disease owl:sameAs ?sameDisease.
?altMedicine tcm:treatment ?sameDisease.
?altMedicine rdf:type tcm:Medicine.
?sameDisease rdfs:label ?diseaseLabel.
?altMedicine rdfs:label ?altMedicineLabel.
```

Seed URI: `http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugs/DB01273`

Query WQ13.

Query semantics: c_{Match}-semantics

BGP:
```
?u dct:subject
         <http://dbpedia.org/resource/Category:Universities_and_colleges_in_Lower_Saxony> .
?u dbowl:city ?c .
?c owl:sameAs ?cAlt .
?cAlt rdfs:label ?cityName .
?cAlt eurostat:unemployment_rate_total ?ur .
```

Seed URI: `http://dbpedia.org/resource/Category:Universities_and_colleges_in_Lower_Saxony`

Query WQ14.

Query semantics: c_{Match}-semantics

BGP:
```
?pub swc:isPartOf
              <http://data.semanticweb.org/conference/eswc/2009/proceedings> .
?pub swc:hasTopic ?topic .
?topic rdfs:label "Ontology engineering"@en .
?pub swrc:author ?author .
?author owl:sameAs ?authorAlt .
?authorAlt foaf:phone ?phone .
```

Seed URI: `http://data.semanticweb.org/conference/eswc/2009/proceedings`

Query WQ15.

Query semantics: c_{Match}-semantics

BGP:
```
<http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?p .
?p foaf:interest ?i .
```

Seed URI: `http://www.w3.org/People/Berners-Lee/card#i`

Query WQ16.

Query semantics: c_{Match}-semantics

BGP:
```
<http://www.w3.org/People/Berners-Lee/card#i> foaf:knows ?p .
?p foaf:knows ?p2 .
?p2 foaf:phone ?i .
```

Seed URI: `http://www.w3.org/People/Berners-Lee/card#i`

Query WQ17.

Query semantics: c_{Match}-semantics

BGP:
```
<http://www4.wiwiss.fu-berlin.de/dailymed/resource/organization/Mylan_Pharmaceuticals_Inc.>
                                                                 dailymed:producesDrug ?bd .
?bd dailymed:genericDrug ?gd .
?gd drugbank:possibleDiseaseTarget ?dt .
?dt diseasome:name "Epilepsy" .
?bd dailymed:activeIngredient ?ai .
?bd2 dailymed:activeIngredient ?ai .
?c dailymed:producesDrug ?bd2 .
?c rdfs:label ?cn .
```

Seed URI:

`http://www4.wiwiss.fu-berlin.de/dailymed/resource/organization/Mylan_Pharmaceuticals_Inc.`

Query WQ18.

Query semantics: c_{Match}-semantics

BGP:
```
?gd drugbank:drugCategory
 <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugcategory/antimalarials> .
?gd drugbank:brandedDrug ?bd .
?c dailymed:producesDrug ?bd .
?c rdfs:label "Pfizer Labs" .
?gd owl:sameAs ?gd2 .
?gd2 foaf:depiction ?p .
```

Seed URI: `http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugcategory/antimalarials`

D.2. Measurements of the WWW-Based Experiment

Query	min.	avg.	max.
WQ1	334	334.0	334
WQ2	185	185.0	185
WQ3	191	191.0	191
WQ4	50	50.0	50
WQ5	6	41.2	50
WQ6	261	261.0	261
WQ7	16	16.0	16
WQ8	12	24.8	28
WQ9	1	1.0	1
WQ10	0	0.0	0
WQ11	51123	51123.0	51123
WQ12	0	0.0	0
WQ13	4	4.0	4
WQ14	0	0.0	0
WQ15	20	20.0	20
WQ16	30	31.6	32
WQ17	0	52.0	65
WQ18	0	0.8	1

(a) result set size

Query	min.	avg.	max.
WQ1	352	352.0	352
WQ2	281	281.0	281
WQ3	395	395.0	395
WQ4	984	984.0	984
WQ5	11	62.2	75
WQ6	1078	1078.0	1078
WQ7	24	24.0	24
WQ8	59	276.6	331
WQ9	207	207.0	207
WQ10	288	288.0	288
WQ11	658	658.0	658
WQ12	16	16.0	16
WQ13	91	91.0	91
WQ14	154	154.8	155
WQ15	54	54.8	55
WQ16	324	324.8	325
WQ17	0	703.2	879
WQ18	44	104.8	120

(b) retrieved documents

Table D.1.: Result set size and number of retrieved documents as measured during the WWW-based experiment (cf. Section 7.3.2).

Query	without cache			with cache
	min.	avg.	max.	
1	494.592	507.226	534.864	0.288
2	433.915	438.832	445.012	0.187
3	1619.716	1628.652	1640.759	0.902
4	1421.391	1432.641	1443.578	7.345
5	18.996	35.215	39.581	0.020
6	1324.191	1431.628	1686.730	0.050
7	105.196	108.366	110.742	0.002
8	1842.871	2421.207	4690.692	0.065
9	54.425	61.498	66.783	0.010
10	1074.739	1076.068	1080.118	0.015
11	365.166	399.769	482.930	6.748
12	24.162	34.456	59.119	0.002
13	935.298	955.963	970.606	0.018
14	778.563	793.079	842.658	0.051
15	506.900	544.696	587.600	0.002
16	2620.847	2652.639	2706.518	0.016
17	30.004	142.770	492.626	4.052
18	39.917	248.415	984.125	0.020

Table D.2.: Query execution times measured during the WWW-based experiment.

D.3. Queries for the Simulation-Based Experiments

For the queries enumerated in this section we assume the following prefix definitions:

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>
PREFIX rev:  <http://purl.org/stuff/rev#>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
```

Query SQ1.

Query semantics: c_{Match}-semantics

BGP:
```
?o bsbm:vendor
       <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Vendor1> .
?o bsbm:product ?p .
?p bsbm:producer
     <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Producer2> .
```

Seed URIs: `http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Vendor1`, `http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Producer2`

Query SQ2.

Query semantics: c_{Match}-semantics

BGP:
```
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromRatingSite1/Review110>
                                                       bsbm:reviewFor ?product .
?product bsbm:productFeature ?feature .
?feature rdfs:label ?featureLabel .
```

Seed URI: `http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromRatingSite1/Review110`

Query SQ3.

Query semantics: c_{Match}-semantics

BGP:
```
?review bsbm:reviewFor
    <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer3/Product128>.
?review bsbm:rating1 ?rating .
?review dc:title ?reviewTitle .
```

Seed URI: `http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer3/Product128`

Query SQ4.

Query semantics: c_{Match}-semantics

BGP:
```
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer5/Product184>
                                                       bsbm:producer ?producer .
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer5/Product184>
                                                       rdf:type ?type .
?product2 bsbm:producer ?producer .
?product2 rdf:type ?type .
```

Seed URI: `http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer5/Product184`

Query SQ5.

Query semantics: c_{Match}-semantics

BGP:
```
?review bsbm:reviewFor
    <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer3/Product128>.
?review bsbm:rating1 10 .
?review rev:reviewer ?reviewer .
?reviewer bsbm:country ?country .
```

Seed URI: `http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer3/Product128`

Query SQ6.

Query semantics: c_{Match}-semantics

BGP:
```
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer3/Product128>
                                               bsbm:productFeature ?feature .
?product2 bsbm:productFeature ?feature .
?product2 bsbm:producer ?producer .
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer3/Product128>
                                               bsbm:producer ?producer .
```

Seed URI: `http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer3/Product128`

E. Proofs of Auxiliary Results

E.1. Proof of Lemma 3.1 (page 54)

Lemma 3.1. *Let $\mathcal{Q}^P$ be a satisfiable SPARQL$_{\mathsf{LD}}$ query that is monotonic; let M^P denote the full-Web machine for the SPARQL expression P used by $\mathcal{Q}^P$; and let W be an arbitrary Web of Linked Data encoded on the Web input tape of M^P. During the execution of Algorithm 3.1 by M^P, $[\![P]\!]_{T_j} \subseteq \mathcal{Q}^P(W)$ holds for all $j \in \{1, 2, \dots\}$.*

Since $\mathcal{Q}^P(W) = [\![P]\!]_{\mathsf{AllData}(W)}$ (cf. Definition 3.1, page 42), we may prove $[\![P]\!]_{T_j} \subseteq \mathcal{Q}^P(W)$ for all $j \in \{1, 2, \dots\}$ by showing $[\![P]\!]_{T_j} \subseteq [\![P]\!]_{\mathsf{AllData}(W)}$, respectively. Since SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$ is monotonic, its SPARQL expression P is monotonic as well (cf. Proposition 3.1, page 43). Therefore, for any $j \in \{1, 2, \dots\}$, $[\![P]\!]_{T_j} \subseteq [\![P]\!]_{\mathsf{AllData}(W)}$ holds if $T_j \subseteq \mathsf{AllData}(W)$. Thus, it remains to show that during the execution of Algorithm 3.1 by machine M^P on Web input $\mathrm{enc}(W)$, $T_j \subseteq \mathsf{AllData}(W)$ holds for all $j \in \{1, 2, \dots\}$.

Let $W = (D, data, adoc)$. As for any LD machine, the computation of M^P starts with an empty lookup tape (cf. Definition 2.9, page 27). Let w_j be the word on the lookup tape of M^P before M^P executes line 4 during the j-th iteration of the main processing loop in Algorithm 3.1. It can be easily seen that for any $j \in \{1, 2, \dots\}$ there exists a finite sequence $u_1, \dots, u_j$ of j different URIs such that w_j is the following word:

$$\mathrm{enc}(u_1)\,\mathrm{enc}(adoc(u_1))\,\sharp\,\dots\,\sharp\,\mathrm{enc}(u_j)\,\mathrm{enc}(adoc(u_j))\,\sharp$$

where, for any URI u_i for which $adoc(u_i) = \bot$, we assume sub-word $\mathrm{enc}(adoc(u_i))$ is the empty word. If U_j is the set that contains all URIs in this sequence $u_1, \dots, u_j$, it holds that $T_j = \{t \in data(adoc(u_i)) \mid u_i \in U_j \text{ and } adoc(u_i) \neq \bot\}$. Clearly, $T_j \subseteq \mathsf{AllData}(W)$. ■

E.2. Proof of Lemma 3.2 (page 54)

Lemma 3.2. *Let $\mathcal{Q}^P$ be a satisfiable SPARQL$_{\mathsf{LD}}$ query that is monotonic; let M^P denote the full-Web machine for the SPARQL expression P used by $\mathcal{Q}^P$; and let W be an arbitrary Web of Linked Data encoded on the Web input tape of M^P. For each solution $\mu \in \mathcal{Q}^P(W)$ there exists a $j_\mu \in \{1, 2, \dots\}$ such that during the execution of Algorithm 3.1 by M^P, $\mu \in [\![P]\!]_{T_j}$ holds for all $j \in \{j_\mu, j_\mu{+}1, \dots\}$.*

To prove Lemma 3.2 we first show that a full-Web machine eventually discovers any RDF triple in the queried Web of Linked Data:

Lemma E.1. *Let $\mathcal{Q}^P$ be a satisfiable SPARQL$_{\mathsf{LD}}$ query that is monotonic; let M^P denote the full-Web machine for the SPARQL expression P used by $\mathcal{Q}^P$; and let W be* *an arbitrary Web of Linked Data encoded on the Web input tape of M^P. For each RDF triple $t \in \mathsf{AllData}(W)$ there exists a $j_t \in \{1, 2, \dots\}$ such that during the execution of Algorithm 3.1 by M^P, $t \in T_j$ holds for all $j \in \{j_t, j_t{+}1, \dots\}$.*

Proof of Lemma E.1. Let $W = (D, data, adoc)$. Furthermore, let $t \in \mathsf{AllData}(W)$ be an arbitrary RDF triple in W. Hence, there exists an LD document $d \in D$ such that $t \in data(d)$. Let $d \in D$ be such an LD document. By our definition of a Web of Linked Data, there exists a URI $u \in \mathcal{U}$ such that $adoc(u) = d$ (see the requirement for mapping *adoc* in Definition 2.1, page 16). Let u be such a URI. Since $u \in \mathcal{U}$, there exists a $j_t \in \{1, 2, \dots\}$ such that machine M^P selects URI u for processing in the j_t-th iteration of the main loop in Algorithm 3.1. After completing the lookup of u during this iteration (cf. line 3 in Algorithm 3.1), the word on the lookup tape contains sub-word enc(d) (cf. Definition 3.4, page 53, and Definition 2.9, page 27). Since $t \in data(d)$, sub-word enc(d) contains sub-word enc(t) (cf. Appendix B, page 193f). Hence, $t \in T_{j_t}$. Since a full-Web machine only appends to (the right end of) the word on its lookup tape, M^P never removes enc(t) from that tape and, thus, $t \in T_j$ holds for all $j \in \{j_t, j_t{+}1, \dots\}$. ■

We now focus on Lemma 3.2. Since $\mathcal{Q}^P(W) = [\![P]\!]_{\mathsf{AllData}(W)}$ (cf. Definition 3.1, page 42), we may prove Lemma 3.2 by showing the following: For each solution $\mu \in [\![P]\!]_{\mathsf{AllData}(W)}$ there exists a $j_\mu \in \{1, 2, \dots\}$ such that during the execution of Algorithm 3.1 by full-Web machine M^P, $\mu \in [\![P]\!]_{T_j}$ holds for all $j \in \{j_\mu, j_\mu{+}1, \dots\}$. Because of the monotonicity of SPARQL$_{\mathsf{LD}}$ query $\mathcal{Q}^P$, SPARQL expression P is monotonic (cf. Proposition 3.1, page 43). For the proof we use an induction on the structure of SPARQL expression P.

Base case: Suppose SPARQL expression P is a triple pattern tp. W.l.o.g., let valuation μ be an arbitrary solution for P in $\mathsf{AllData}(W)$; i.e., $\mu \in [\![P]\!]_{\mathsf{AllData}(W)}$. According to the definition of SPARQL semantics it holds that (i) dom(μ) = vars(tp) and (ii) there exists an RDF triple $t \in \mathsf{AllData}(W)$ such that $t = \mu[tp]$ (cf. Section 3.2.1, page 38ff). Due to Lemma E.1, there exists a $j_t \in \{1, 2, \dots\}$ such that $t \in T_j$ for all $j \in \{j_t, j_t{+}1, \dots\}$. Since P is monotonic we conclude $\mu \in [\![P]\!]_{T_j}$ for all $j \in \{j_t, j_t{+}1, \dots\}$.

Induction step: Let P_1 and P_2 be SPARQL expressions such that the following induction hypothesis holds for all $i \in \{1, 2\}$:

- For each valuation $\mu \in [\![P_i]\!]_{\mathsf{AllData}(W)}$ there exists a $j_\mu \in \{1, 2, \dots\}$ such that, during the execution of Algorithm 3.1 by machine M^P, $\mu \in [\![P_i]\!]_{T_j}$ holds for all $j \in \{j_\mu, j_\mu{+}1, \dots\}$.

Let P be a SPARQL expression that can be constructed using P_1 and P_2 (i.e., P is of the form (P_1 AND P_2), (P_1 UNION P_2), (P_1 OPT P_2), or (P_1 FILTER R), where R is a filter condition), and let valuation $\mu \in [\![P]\!]_{\mathsf{AllData}(W)}$ be an arbitrary solution for P in $\mathsf{AllData}(W)$. Then, w.l.o.g., we show that there exists a $j_\mu \in \{1, 2, \dots\}$ such that, during the execution of Algorithm 3.1 by machine M^P, $\mu \in [\![P]\!]_{T_j}$ holds for all $j \in \{j_\mu, j_\mu{+}1, \dots\}$. We have to consider the following cases:

- P is (P_1 AND P_2). In this case there exist valuations $\mu_1 \in [\![P_1]\!]_{\mathsf{AllData}(W)}$ and $\mu_2 \in [\![P_2]\!]_{\mathsf{AllData}(W)}$ such that $\mu_1 \sim \mu_2$ and $\mu = \mu_1 \cup \mu_2$. By induction there exist

$j_{\mu_1}, j_{\mu_2} \in \{1, 2, \dots\}$ such that (i) $\mu_1 \in [\![P_1]\!]_{T_j}$ for all $j \in \{j_{\mu_1}, j_{\mu_1}{+}1, \dots\}$ and (ii) $\mu_2 \in [\![P_2]\!]_{T_j}$ for all $j \in \{j_{\mu_2}, j_{\mu_2}{+}1, \dots\}$. Let $j_\mu = \max(\{j_{\mu_1}, j_{\mu_2}\})$. Due to the monotonicity of P we have $\mu \in [\![P]\!]_{T_j}$ for all $j \in \{j_\mu, j_\mu{+}1, \dots\}$.

- P is $(P_1 \text{ UNION } P_2)$. In this case there exists a sub-expression $P' \in \{P_1, P_2\}$ and a valuation $\mu' \in [\![P']\!]_{\mathsf{AllData}(W)}$ such that $\mu' = \mu$. W.l.o.g., let P_1 be this sub-expression P'. Then, by induction, there exists a $j_{\mu'} \in \{1, 2, \dots\}$ such that $\mu' \in [\![P_1]\!]_{T_j}$ for all $j \in \{j_{\mu'}, j_{\mu'}{+}1, \dots\}$. Due to the monotonicity of P we have $\mu \in [\![P]\!]_{T_j}$ for all $j \in \{j_\mu, j_\mu{+}1, \dots\}$.

- P is $(P_1 \text{ OPT } P_2)$. We distinguish two cases:
  1. There exist valuations $\mu_1 \in [\![P_1]\!]_{\mathsf{AllData}(W)}$ and $\mu_2 \in [\![P_2]\!]_{\mathsf{AllData}(W)}$ such that $\mu_1 \sim \mu_2$ and $\mu = \mu_1 \cup \mu_2$. This case corresponds to the case in which P is $(P_1 \text{ AND } P_2)$ (see above).
  2. There exists a valuation $\mu_1 \in [\![P_1]\!]_{\mathsf{AllData}(W)}$ such that (i) $\mu_1 \not\sim \mu_2$ for all $\mu_2 \in [\![P_2]\!]_{\mathsf{AllData}(W)}$ and (ii) $\mu = \mu_1$. By induction there exists a $j_{\mu_1} \in \{1, 2, \dots\}$ such that $\mu_1 \in [\![P_1]\!]_{T_j}$ for all $j \in \{j_{\mu_1}, j_{\mu_1}{+}1, \dots\}$. Since P is monotonic we have $\mu \in [\![P]\!]_{T_j}$ for all $j \in \{j_{\mu_1}, j_{\mu_1}{+}1, \dots\}$.

- P is $(P_1 \text{ FILTER } R)$. In this case there exists a valuation $\mu' \in [\![P_1]\!]_{\mathsf{AllData}(W)}$ such that $\mu = \mu'$. By induction there exists a $j_{\mu'} \in \{1, 2, \dots\}$ such that $\mu' \in [\![P_1]\!]_{T_j}$ for all $j \in \{j_{\mu'}, j_{\mu'}{+}1, \dots\}$. Due to the monotonicity of P we have $\mu \in [\![P]\!]_{T_j}$ for all $j \in \{j_{\mu'}, j_{\mu'}{+}1, \dots\}$. ■

E.3. Proof of Lemma 4.1 (page 73)

Lemma 4.1. *Let $\mathcal{T}$, $\mathcal{U}$, $\mathcal{P}$, and $\mathcal{C}$ denote the infinite sets of all possible RDF triples, all URIs, all possible SPARQL expressions, and all possible reachability criteria, respectively. Furthermore, we define a function $X : \mathcal{C} \times \mathcal{P} \to \mathcal{U}$ that maps any pair of a reachability criterion $c \in \mathcal{C}$ and a SPARQL expression $P \in \mathcal{P}$ to a set of URIs:*

$$X(c, P) := \{ u \in \mathcal{U} \mid \exists t \in \mathcal{T} : u \in \mathrm{uris}(t) \text{ and } c(t, u, P) = \mathrm{true} \} .$$

Then, for each reachability criterion $c \in \mathcal{C}$ it holds that c ensures finiteness if and only if $X(c, P)$ is finite for all SPARQL expressions $P \in \mathcal{P}$.

W.l.o.g., let $c \in \mathcal{C}$ be an arbitrary reachability criterion.

If: Suppose $X(c, P)$ is finite for all SPARQL expressions $P \in \mathcal{P}$. To show that c ensures finiteness we use proof by contradiction. That is, we assume c does not ensures finiteness. Then, there exist a Web of Linked Data W, a finite set of URIs $S \subseteq \mathcal{U}$, and a SPARQL expression P, such that the (S, c, P)-reachable subweb of W is infinite. Let $R = (D_{\mathfrak{R}}, data_{\mathfrak{R}}, adoc_{\mathfrak{R}})$ be this subweb. Furthermore, let $D_{\mathsf{S}} \subseteq D_{\mathfrak{R}}$ be the corresponding set of seed documents (i.e., $D_{\mathsf{S}} = \{ d \in D_{\mathfrak{R}} \mid \exists u \in S : adoc_{\mathfrak{R}}(u) = d \}$). D_{S} is finite because the set of seed URIs S is finite. Therefore, the set of non-seed documents $D_{\mathsf{NS}} = D_{\mathfrak{R}} \setminus D_{\mathsf{S}}$

must be infinite. Since each of these non-seed documents are (c, P)-reachable from S in W, each of them satisfies the second condition in Definition 4.2 (cf. page 62). Hence, for each document $d \in D_{\mathsf{NS}}$ there exists a distinct URI $u \in \mathcal{U}$ with $adoc_{\mathfrak{R}}(u) = d$ and an RDF triple $t \in \mathcal{T}$ such that $u \in \text{uris}(t)$ and $c(t, u, P) = \text{true}$, and, thus, $u \in X(c, P)$. As a consequence, the set of URIs $X(c, P)$ is infinite (recall the infiniteness of D_{NS}), which is a contradiction. Therefore, reachability criterion c ensures finiteness.

Only if: To prove that $X(c, P)$ is finite for all SPARQL expressions $P \in \mathcal{P}$ if c ensures finiteness, we use proof by contraposition. That is, we show that c does not ensure finiteness if $X(c, P)$ is infinite for some SPARQL expression $P \in \mathcal{P}$.

Suppose $P \in \mathcal{P}$ is a SPARQL expression such that $X(c, P)$ is infinite. To show that c does not ensure finiteness we use the URIs in $X(c, P)$ to construct an infinite Web of Linked Data W^* and a set of seed URIs S^* such that the (S^*, c, P)-reachable subweb of W^* is infinite. To this end, we assume an arbitrary strict, total order $<$ over $X(c, P)$. Based on this order we obtain an infinite sequence of distinct URIs $u_1, u_2, ...$ such that, for each $i \in \{1, 2, ...\}$, $u_i < u_{i+1}$ and $u_i \in X(c, P)$.

We now use this sequence of URIs to construct the (infinite) Web of Linked Data $W^* = (D^*, data^*, adoc^*)$ as follows: For each URI u_i in our sequence (i.e., $i \in \{1, 2, ...\}$) we construct a distinct LD document $d_i \in D^*$ such that $adoc^*(u_i) = d_i$ and $data^*(d_i) = \{t\}$ where $t \in \mathcal{T}$ is an RDF triple such that $u_{i+1} \in \text{uris}(t)$ and $c(t, u_{i+1}, P) = \text{true}$. Such a triple exists because for the next URI in our sequence, u_{i+1}, it holds that $u_{i+1} \in X(c, P)$.

Then, each of these documents $d_i \in D^*$ (for all $i \in \{1, 2, ...\}$) is (c, P)-reachable from $S^* = \{u_1\}$ in W^* (where the seed URI u_1 is the first URI in our sequence). Due to the infinite number of these documents, the (S^*, c, P)-reachable subweb of W^* is infinite. Thus, reachability criterion c does not ensure finiteness (cf. Definition 4.5, page 67). ∎

E.4. Proof of Lemma 4.2 (page 88)

Lemma 4.2. *Let M be the 2P machine for a SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$; let $W = (D, data, adoc)$ be a Web of Linked Data encoded on the Web input tape of M; and let $d \in D$ be an LD document that is (c, P)-reachable from S in W. During the execution of Algorithm 4.1 by M there exists an iteration of the loop (lines 3 to 5) after which the word on the lookup tape of M contains* enc(d) *permanently.*

To prove Lemma 4.2 we first emphasize that 2P machine M only appends to the word on its lookup tape. Hence, M never removes enc(d) from that word once it has been added. The same holds for the encoding of any other LD document $d' \in D$.

Since d is (c, P)-reachable from S in W, the link graph of W contains at least one finite path $(d_0, ..., d_n)$ of LD documents $d_i \in D$ (for $i \in \{0, ..., n\}$) such that: (i) $n \in \{0, 1, ...\}$, (ii) $d_n = d$, (iii) $\exists\, u \in S : adoc(u) = d_0$, and (iv) for each $i' \in \{1, ..., n\}$ it holds that:

$$\exists\, t \in data(d_{i'-1}) : \Big(\exists\, u \in \text{uris}(t) : \big(adoc(u) = d_{i'} \text{ and } c(t, u, P) = \text{true}\big)\Big) \qquad \text{(E.1)}$$

Let $(d_0, ..., d_n)$ be such a path. To prove Lemma 4.2 we show by induction on n that there exists an iteration after which the word on the lookup tape of 2P machine M contains enc(d_n) (which is the same as enc(d) because $d_n = d$).

Base case ($n = 0$): Since there exists a seed URI $u \in S$ such that $adoc(u) = d_0$, it is easy to verify that after the 0-th iteration (i.e., before the first iteration) the word on the lookup tape of M contains $\mathrm{enc}(d_0)$ (cf. line 1 in Algorithm 4.1).

Induction step ($n > 0$): There exists an iteration after which the word on the lookup tape of M contains $\mathrm{enc}(d_{n-1})$. Let this be the j-th iteration. We show that there exists an iteration after which the word on the lookup tape of M contains $\mathrm{enc}(d_n)$. We distinguish two cases: After the j-th iteration the word on the lookup tape already contains $\mathrm{enc}(d_n)$ or it does not contain $\mathrm{enc}(d_n)$. We have to discuss the latter case only. Due to (E.1) there exist an RDF triple $t \in data(d_{n-1})$ and a URI $u \in \mathrm{uris}(t)$ such that $adoc(u) = d_n$ and $c(t, u, P) = \mathrm{true}$. Then, by the definition of Algorithm 4.1, there exists a $\delta \in \{1, 2, ...\}$ such that machine M finds t and u in the $(j+\delta)$-th iteration. During this iteration, M calls subroutine *lookup* for u (cf. line 4 in Algorithm 4.1). Hence, after the $(j+\delta)$-th iteration the lookup tape of M contains $\mathrm{enc}(d_n)$. ■

E.5. Proof of Lemma 4.3 (page 89)

Lemma 4.3. *Let M be the 2P machine for a SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$; let W be a Web of Linked Data encoded on the Web input tape of M; and let R denote the (S, c, P)-reachable subweb of W. For any RDF triple t for which* $\mathrm{enc}(t)$ *eventually appears on the lookup tape of M during the execution of Algorithm 4.1 by M it holds that $t \in \mathsf{AllData}(R)$.*

Let $W = (D, data, adoc)$. Furthermore, let w_j denote the word on the lookup tape of M when M finishes the j-th iteration of the loop in Algorithm 4.1; w_0 denotes the corresponding word before the first iteration (that is, after the initialization at line 1). To prove the lemma it is sufficient to show that for each $j \in \{0, 1, ...\}$ there exists a finite sequence $u_1, ..., u_{n_j}$ of n_j different URIs such that (i) for each $i \in \{1, ..., n_j\}$ either $adoc(u_i) = \perp$ or $adoc(u_i) \in D$ is an LD document which is (c, P)-reachable from S in W, and (ii) w_j is the following word:[1]

$$\mathrm{enc}(u_1)\,\mathrm{enc}(adoc(u_1))\,\sharp\; ...\; \sharp\,\mathrm{enc}(u_{n_j})\,\mathrm{enc}(adoc(u_{n_j}))\,\sharp$$

We use an induction on j for this proof.

Base case ($j = 0$): The computation of M starts with an empty lookup tape (as is the case for any LD machine based computation; cf. Definition 2.9, page 27). After the initialization at line 1 in Algorithm 4.1, word w_0 is a concatenation of sub-words $\mathrm{enc}(u)\,\mathrm{enc}(adoc(u))\,\sharp$ for all seed URIs $u \in S$. Hence, we have a corresponding sequence $u_1, ..., u_{n_0}$ such that $n_0 = |S|$ and $u_i \in S$ for all $i \in \{1, ..., n_0\}$. The order of the URIs in this sequence depends on the order in which the seed URIs have been looked up and is irrelevant for our proof. For each seed URI $u \in S$ it holds that either $adoc(u_i) = \perp$ or $adoc(u)$ is (c, P)-reachable from S in W (cf. case 1 in Definition 4.2, page 62).

Induction step ($j > 0$): Let there exist a finite sequence $u_1, ..., u_{n_{j-1}}$ of n_{j-1} different URIs such that (i) for each $i \in \{1, ..., n_{j-1}\}$ either $adoc(u_i) = \perp$ or $adoc(u_i) \in D$ is (c, P)-reachable from S in W, and (ii) word w_{j-1} is the following word:

[1]We assume $\mathrm{enc}(adoc(u_i))$ is the empty word if $adoc(u_i) = \perp$.

$$\mathrm{enc}(u_1)\,\mathrm{enc}(adoc(u_1))\,\sharp\,...\,\sharp\,\mathrm{enc}(u_{n_{j-1}})\,\mathrm{enc}(adoc(u_{n_{j-1}}))\,\sharp$$

Let t be the RDF triple and $u \in \mathrm{uris}(t)$ be the URI that machine M finds (encoded as part of word w_{j-1}) and uses in the j-th iteration; i.e., $c(t, u, P) = \mathrm{true}$ and subroutine *lookup* has not been called for URI u. The machine calls *lookup* for u, which changes the word on the lookup tape to w_j. Hence, w_j is equal to $w_{j-1}\,\mathrm{enc}(u)\,\mathrm{enc}(adoc(u))\,\sharp$ and, thus, the sequence of URIs for w_j is $u_1, ..., u_{n_{j-1}}, u$. It remains to show that LD document $adoc(u)$ is (c, P)-reachable from S in W if $adoc(u) \neq \bot$.

Suppose $adoc(u) \neq \bot$. Since RDF triple t is encoded as part of w_{j-1}, by the inductive hypothesis, t must be contained in the data of an LD document $d^* \in D$ that is (c, P)-reachable from S in W (and for which there exists an $i \in \{1, ..., n_{j-1}\}$ such that $adoc(u_i) = d^*$). Therefore, d^*, t, and u satisfy Condition 2 in Definition 4.2 (cf. page 62) and, thus, LD document $adoc(u)$ is (c, P)-reachable from S in W. ■

E.6. Proof of Lemma 4.4 (page 89)

Lemma 4.4. *Let M be the 2P machine for a SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ and let W be a Web of Linked Data encoded on the Web input tape of M. The computation of M halts after a finite number of steps if and only if the (S, c, P)-reachable subweb of W is finite.*

Let $W = (D, data, adoc)$ and let R denote the (S, c, P)-reachable subweb of W.

If: Suppose R is finite. To show that the computation of M halts after a finite number of steps we emphasize the following observations:

1. Each call of subroutine *lookup* by M terminates because the encoding of W on the Web input tape of M is ordered (cf. Appendix B on page 193f).

2. Based on Observation 1, machine M completes the initialization at line 1 in Algorithm 4.1 after a finite number of steps because the set of seed URIs S is finite.

3. At any point in the computation the word on the lookup tape of M is finite because (i) during each iteration of the loop in Algorithm 4.1, M appends one (encoded) LD document to that tape and (ii) the encoding of any LD document is finite (because the set of RDF triples $data(d)$ for each LD document $d \in D$ is finite).

4. During each iteration of the loop in Algorithm 4.1, machine M completes the scan of its lookup tape (cf. line 4) after a finite number of steps because the word on this tape is always finite (see Observation 3). Thus, together with the first observation, machine M finishes each iteration of the loop after a finite number of steps.

5. Machine M considers only those URIs for a call of subroutine *lookup* that are mentioned in some RDF triple t for which $t \in \mathsf{AllData}(R)$ holds (cf. Lemma 4.3). Given that R is finite, there exists only a finite number of such URIs. Since M considers each of these URIs only once (cf. line 4), the loop in Algorithm 4.1 as performed by M has a finite number of iterations.

6. Since the word on the lookup tape is finite (see Observation 3), the set G used at line 6 in Algorithm 4.1 is finite and, thus, $[\![P]\!]_G$ is finite. As a consequence, M requires only a finite number of computation steps for executing line 6.

Altogether, these observations show that the computation of M halts after a finite number of steps (given the finiteness of R).

Only if: We use proof by contraposition. That is we show that if R is *in*finite, then the computation of M does *not* halt after a finite number of steps: Suppose R is infinite. Then, it is easy to see that the data retrieval loop in Algorithm 4.1 (i.e., lines 3 to 5) never terminates. ■

E.7. Proof of Lemma 4.5 (page 92)

Lemma 4.5. *Let M be the ER machine for a* monotonic *SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ and let W be a Web of Linked Data encoded on the Web input tape of M. During the execution of Algorithm 4.2 by M it holds that $[\![P]\!]_{T_j} \subseteq \mathcal{Q}_c^{P,S}(W)$ for all $j \in \{1, 2, \ldots\}$.*

Let $W = (D, data, adoc)$. Furthermore, let R denote the (S, c, P)-reachable part of W. As a basis for proving Lemma 4.5 we use the following lemma:

Lemma E.2. *During the execution of Algorithm 4.2 by M on (Web) input* $\mathrm{enc}(W)$ *it holds that $T_j \subseteq \mathsf{AllData}(R)$ for all $j \in \{1, 2, \ldots\}$.*

Proof of Lemma E.2. This proof resembles the proof of the corresponding lemma for 2P machines (cf. Lemma 4.3, page 89). Let w_j denote the word on the lookup tape of ER machine M when M starts the j-th iteration of the main processing loop in Algorithm 4.2 (i.e., before line 3).

To prove $T_j \subseteq \mathsf{AllData}(R)$ for all $j \in \{1, 2, \ldots\}$ it is sufficient to show for each word w_j (where $j \in \{1, 2, \ldots\}$) there exists a finite sequence $u_1, \ldots, u_{n_j}$ of n_j different URIs such that (i) for each $i \in \{1, \ldots, n_j\}$ either $adoc(u_i) = \perp$ or $adoc(u_i) \in D$ is an LD document that is (c, P)-reachable from S in W, and (ii) w_j is the following word:[2]

$$\mathrm{enc}(u_1)\,\mathrm{enc}(adoc(u_1))\,\sharp\,\ldots\,\sharp\,\mathrm{enc}(u_{n_j})\,\mathrm{enc}(adoc(u_{n_j}))\,\sharp$$

We use an induction on j for this proof.

Base case ($j = 1$): The computation of M starts with an empty lookup tape (as any LD machine based computation; cf. Definition 2.9, page 27). Due to the initialization in line 1 of Algorithm 4.2, word w_1 is a concatenation of sub-words $\mathrm{enc}(u)\,\mathrm{enc}(adoc(u))\,\sharp$ for all (seed) URIs $u \in S$. Hence, there exists a corresponding sequence of URIs $u_1, \ldots, u_{n_1}$ where $n_1 = |S|$ and $u_i \in S$ for all $i \in \{1, \ldots, n_1\}$. The order of the URIs in that sequence depends on the order in which these URIs have been looked up and is irrelevant for our proof. For every seed URI $u \in S$ it holds that either $adoc(u_i) = \perp$ or $adoc(u)$ is (c, P)-reachable from S in W (cf. Condition 1 in Definition 4.2, page 62).

[2]We, again, assume $\mathrm{enc}(adoc(u_i))$ is the empty word if $adoc(u_i)) = \perp$.

Induction step ($j > 0$): Let there exist a finite sequence $u_1, \dots, u_{n_{j-1}}$ of n_{j-1} different URIs such that (i) for each $i \in \{1, \dots, n_{j-1}\}$ either $adoc(u_i) = \perp$ or $adoc(u_i) \in D$ is (c, P)-reachable from S in W, and (ii) word w_{j-1} is:

$$\mathrm{enc}(u_1)\,\mathrm{enc}(adoc(u_1))\,\sharp\ \dots\ \sharp\,\mathrm{enc}(u_{n_{j-1}})\,\mathrm{enc}(adoc(u_{n_{j-1}}))\,\sharp$$

In the j-th iteration machine M finds an RDF triple t encoded as part of word w_{j-1} such that $\exists\, u \in \mathrm{uris}(t) : c(t, u, P) = \mathrm{true}$ and subroutine *lookup* has not been called for URI u. The machine calls *lookup* for u, which changes the word on the lookup tape to w_j. Hence, w_j is equal to $w_{j-1}\,\mathrm{enc}(u)\,\mathrm{enc}(adoc(u))\,\sharp$ and, thus, the sequence of URIs for w_j is $u_1, \dots, u_{n_{j-1}}, u$. It remains to show that LD document $adoc(u)$ is (c, P)-reachable from S in W if $adoc(u) \neq \perp$.

Suppose $adoc(u) \neq \perp$. Since RDF triple t is encoded as part of w_{j-1}, by the inductive hypothesis, t must be contained in the data of an LD document $d^* \in D$ that is (c, P)-reachable from S in W (and for which there exists an $i \in \{1, \dots, n_{j-1}\}$ such that $adoc(u_i) = d^*$). Therefore, t and u satisfy the requirements as given in Condition 2 of Definition 4.2 and, thus, $adoc(u)$ is (c, P)-reachable from S in W. ■

Due to the monotonicity of $\mathcal{Q}_c^{P,S}$ it is trivial to show Lemma 4.5 using Lemma E.2 (recall, $\mathcal{Q}_c^{P,S}(W) = [\![P]\!]_{\mathsf{AllData}(R)}$). ■

E.8. Proof of Lemma 4.6 (page 93)

Lemma 4.6. *Let M be the ER machine for a* monotonic *SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ whose reachability criterion c does* not *ensure finiteness; and let W be a Web of Linked Data encoded on the Web input tape of M. For each solution $\mu \in \mathcal{Q}_c^{P,S}(W)$ there exists a $j_\mu \in \{1, 2, \dots\}$ such that during the execution of Algorithm 4.2 by M it holds that $\mu \in [\![P]\!]_{T_j}$ for all $j \in \{j_\mu, j_\mu{+}1, \dots\}$.*

Let $W = (D, data, adoc)$. Furthermore, let R denote the (S, c, P)-reachable part of W. As a basis for proving Lemma 4.6 we use the following lemma:

Lemma E.3. *For each RDF triple $t \in \mathsf{AllData}(R)$ there exists a $j_t \in \{1, 2, \dots\}$ such that during the execution of Algorithm 4.2 by M on (Web) input* $\mathrm{enc}(W)$ *it holds that $t \in T_j$ for all $j \in \{j_t, j_t{+}1, \dots\}$.*

Proof of Lemma E.3. Let w_j denote the word on the lookup tape of ER machine M when M starts the j-th iteration of the main processing loop in Algorithm 4.2 (i.e., before line 3). Let t be an arbitrary RDF triple such that $t \in \mathsf{AllData}(R)$. Hence, there exists an LD document $d \in D$ such that (i) $t \in data(d)$ and (ii) d is (c, P)-reachable from S in W. Let d be such a document. Since M only appends to its lookup tape, we prove that there exists a $j_t \in \{1, 2, \dots\}$ such that $t \in T_j$ for all $j \in \{j_t, j_t{+}1, \dots\}$, by showing that there exists a $j_t \in \{1, 2, \dots\}$ such that w_{j_t} contains the sub-word $\mathrm{enc}(d)$. This proof resembles the proof of the corresponding lemma for 2P machines (cf. Lemma 4.2, page 88).

Since document d is (c,P)-reachable from S in W, the link graph of W contains at least one finite path $(d_0,\dots,d_n)$ of LD documents $d_i \in D$ (for $i \in \{0,\dots,n\}$) such that: (i) $n \in \{0,1,\dots\}$, (ii) $d_n = d$, (iii) $\exists\, u \in S : adoc(u) = d_0$, and (iv) for each $i' \in \{1,\dots,n\}$,

$$\exists\, t \in data(d_{i'-1}) : \Big(\exists\, u \in \mathrm{uris}(t) : \big(adoc(u) = d_{i'} \text{ and } c(t,u,P) = \mathrm{true}\big)\Big). \tag{E.2}$$

Let $(d_0,\dots,d_n)$ be such a path. To prove Lemma E.3 we show by induction on n that there exists a $j_t \in \{1,2,\dots\}$ such that w_{j_t} contains the sub-word $\mathrm{enc}(d_n)$ (which is the same as $\mathrm{enc}(d)$ because $d_n = d$).

Base case ($n = 0$): Since there exists a seed URI $u \in S$ such that $adoc(u) = d_0$, it is easy to verify that w_1 contains the sub-word $\mathrm{enc}(d_0)$ (cf. line 1 in Algorithm 4.2).

Induction step ($n > 0$): Suppose there exists a $j \in \{1,2,\dots\}$ such that w_j contains sub-word $\mathrm{enc}(d_{n-1})$. We will show that there exists a $j' \in \{j, j+1,\dots\}$ such that $w_{j'}$ contains the sub-word $\mathrm{enc}(d_n)$. We distinguish two cases: either $\mathrm{enc}(d_n)$ is not contained in w_j or it is already contained in w_j. In the first case we have $j' > j$; in the latter case we have $j' = j$. Hence, we have to discuss the first case only.

Due to equation (E.2) there exist an RDF triple $t^* \in data(d_{n-1})$ and a URI $u^* \in \mathrm{uris}(t^*)$ such that $adoc(u^*) = d_n$ and $c(t^*,u^*,P) = \mathrm{true}$. Then, by Algorithm 4.2, there exists a $\delta \in \{1,2,\dots\}$ such that machine M finds t^* and u^* in the $(j{+}\delta)$-th iteration. During this iteration, M calls subroutine *lookup* for u^* (cf. line 5 in Algorithm 4.2). Hence, word $w_{j+\delta+1}$ contains $\mathrm{enc}(d_n)$ and, thus, $j' = j + \delta + 1$. ■

We now prove Lemma 4.6. This proof resembles the proof of the corresponding lemma for full-Web machines (cf. Lemma 3.2, page 54).

Since $\mathcal{Q}_c^{P,S}(W) = [\![P]\!]_{\mathsf{AllData}(R)}$ (cf. Definition 4.4, page 63), we may prove Lemma 4.6 by showing the following: For each solution $\mu \in [\![P]\!]_{\mathsf{AllData}(R)}$ there exists some $j_\mu \in \{1,2,\dots\}$ such that during the execution of Algorithm 4.2 by ER machine M it holds that $\mu \in [\![P]\!]_{T_j}$ for all $j \in \{j_\mu, j_\mu{+}1,\dots\}$. We note that, by Proposition 4.9 (cf. page 81), SPARQL expression P is monotonic because $\mathrm{SPARQL}_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ is monotonic and reachability criterion c does not ensure finiteness.

For the proof we use an induction on the possible structure of SPARQL expression P.

Base case: Suppose SPARQL expression P is a triple pattern tp. W.l.o.g., let valuation μ be an arbitrary solution for P in $\mathsf{AllData}(R)$; i.e., $\mu \in [\![P]\!]_{\mathsf{AllData}(R)}$. According to the definition of the standard SPARQL semantics (as given in Section 3.2.1, page 38ff) it holds that (i) $\mathrm{dom}(\mu) = \mathrm{vars}(tp)$ and (ii) there exists an RDF triple $t \in \mathsf{AllData}(R)$ such that $t = \mu[tp]$. By Lemma E.3, there exists a $j_\mu \in \{1,2,\dots\}$ such that $t \in T_j$ for all $j \in \{j_\mu, j_\mu{+}1,\dots\}$. Since P is monotonic, we conclude $\mu \in [\![P]\!]_{T_j}$ for all $j \in \{j_\mu, j_\mu{+}1,\dots\}$.

Induction step: Let P_1 and P_2 be SPARQL expressions. By induction, assume for each $i \in \{1,2\}$:

- For each valuation $\mu \in [\![P_i]\!]_{\mathsf{AllData}(R)}$ there exists a $j_\mu \in \{1,2,\dots\}$ such that, during the execution of Algorithm 4.2 by M, $\mu \in [\![P_i]\!]_{T_j}$ holds for all $j \in \{j_\mu, j_\mu{+}1,\dots\}$.

For any SPARQL expression P that can be constructed using P_1 and P_2 we will show:

- For each valuation $\mu \in [\![P]\!]_{\mathsf{AllData}(R)}$ there exists a $j_\mu \in \{1, 2, \dots\}$ such that, during the execution of Algorithm 4.2 by M, $\mu \in [\![P]\!]_{T_j}$ holds for all $j \in \{j_\mu, j_\mu{+}1, \dots\}$.

Let valuation μ be an arbitrary solution for P in $\mathsf{AllData}(R)$; i.e., $\mu \in [\![P]\!]_{\mathsf{AllData}(R)}$. We need to consider the following cases:

- P is $(P_1 \text{ AND } P_2)$. In this case there exist two valuations $\mu_1 \in [\![P_1]\!]_{\mathsf{AllData}(R)}$ and $\mu_2 \in [\![P_2]\!]_{\mathsf{AllData}(R)}$ such that $\mu_1 \sim \mu_2$ and $\mu = \mu_1 \cup \mu_2$. By induction, there exist $j_{\mu_1}, j_{\mu_2} \in \{1, 2, \dots\}$ such that, for each $i \in \{1, 2\}$, $\mu_i \in [\![P_i]\!]_{T_j}$ for all $j \in \{j_{\mu_i}, j_{\mu_i}{+}1, \dots\}$. Let $j_\mu = \max(\{j_{\mu_1}, j_{\mu_2}\})$. Due to the monotonicity of P we have $\mu \in [\![P]\!]_{T_j}$ for all $j \in \{j_\mu, j_\mu{+}1, \dots\}$.

- P is $(P_1 \text{ FILTER } R)$. In this case there exists a valuation $\mu' \in [\![P_1]\!]_{\mathsf{AllData}(R)}$ such that $\mu = \mu'$. By induction, there exists a $j_{\mu'} \in \{1, 2, \dots\}$ such that $\mu' \in [\![P_1]\!]_{T_j}$ for all $j \in \{j_{\mu'}, j_{\mu'}{+}1, \dots\}$. Due to the monotonicity of P we have $\mu \in [\![P]\!]_{T_j}$ for all $j \in \{j_{\mu'}, j_{\mu'}{+}1, \dots\}$.

- P is $(P_1 \text{ OPT } P_2)$. We distinguish two cases:
  1. There exist valuations $\mu_1 \in [\![P_1]\!]_{\mathsf{AllData}(R)}$ and $\mu_2 \in [\![P_2]\!]_{\mathsf{AllData}(R)}$ such that $\mu_1 \sim \mu_2$ and $\mu = \mu_1 \cup \mu_2$. This case corresponds to the case where P is $(P_1 \text{ AND } P_2)$ (see above).
  2. There exists a valuation $\mu_1 \in [\![P_1]\!]_{\mathsf{AllData}(R)}$ such that (i) $\mu = \mu_1$ and (ii) $\mu_1 \not\sim \mu_2$ for all $\mu_2 \in [\![P_2]\!]_{\mathsf{AllData}(R)}$. By induction, there exists a $j_{\mu_1} \in \{1, 2, \dots\}$ such that $\mu_1 \in [\![P_1]\!]_{T_j}$ for all $j \in \{j_{\mu_1}, j_{\mu_1}{+}1, \dots\}$. Since P is monotonic we have $\mu \in [\![P]\!]_{T_j}$ for all $j \in \{j_{\mu_1}, j_{\mu_1}{+}1, \dots\}$.

- P is $(P_1 \text{ UNION } P_2)$. We distinguish two cases:
  1. There exists a valuation $\mu' \in [\![P_1]\!]_{\mathsf{AllData}(W)}$ such that $\mu' = \mu$. By induction, there exists a $j_{\mu'} \in \{1, 2, \dots\}$ such that $\mu' \in [\![P_1]\!]_{T_j}$ for all $j \in \{j_{\mu'}, j_{\mu'}{+}1, \dots\}$.
  2. There exists a valuation $\mu' \in [\![P_2]\!]_{\mathsf{AllData}(W)}$ such that $\mu' = \mu$. By induction, there exists a $j_{\mu'} \in \{1, 2, \dots\}$ such that $\mu' \in [\![P_2]\!]_{T_j}$ for all $j \in \{j_{\mu'}, j_{\mu'}{+}1, \dots\}$.

  In both cases we have $\mu \in [\![P]\!]_{T_j}$ for all $j \in \{j_{\mu'}, j_{\mu'}{+}1, \dots\}$, because P is monotonic.

■

E.9. Proof of Lemma 4.7 (page 93)

Lemma 4.7. *Let M be the ER machine for a SPARQL$_{\mathsf{LD(R)}}$ query $\mathcal{Q}_c^{P,S}$ and let W be a Web of Linked Data encoded on the Web input tape of M. During the execution of Algorithm 4.2, M completes each iteration of the loop in Algorithm 4.2 after a finite number of computation steps.*

Let $W = (D, data, adoc)$. First, we emphasize the following properties:

1. Each call of subroutine *lookup* in Algorithm 4.2 terminates because the encoding of the queried Web of Linked Data on the Web input tape of any LD machine (including the ER machine M) is ordered (cf. Appendix B, page 193f).

2. The ER machine M completes the initialization in line 1 of Algorithm 4.2 in a finite number of computation steps because the set of seed URIs S is finite.

3. At any point in the computation, the word on the lookup tape of ER machine M is finite because the machine only gradually appends (encoded) LD documents to that tape (one document per iteration) and the encoding of each document is finite (recall that the set of RDF triples $data(d)$ for each LD document d is finite).

It remains to show that each iteration of the loop also only requires a finite number of computation steps: Due to the finiteness of the word on the lookup tape, every set $[\![P]\!]_{T_j}$ (for $j = \{1, 2, \ldots\}$) is finite. Due to this finiteness, lines 3 and 4 can be completed in a finite number of computation steps (during any iteration). The scan in line 5 also finishes after a finite number of computation steps because of the finiteness of the word on the lookup tape. ■

E.10. Proof of Lemma 6.1 (page 123)

Lemma 6.1. *Let $\mathcal{Q}^{B,S}_{c_{\mathsf{Match}}}$ be a $C_{\mathsf{LD(R)}}$ query (under c_{Match}-semantics); let R denote the $(S, c_{\mathsf{Match}}, B)$-reachable subweb of a Web of Linked Data W; and let $R_{\mathfrak{D}}$ be a discovered subweb of R. For any RDF triple t with (i) $t \in \mathsf{AllData}(R_{\mathfrak{D}})$ and (ii) t is a matching triple for a triple pattern $tp \in B$, it holds that $\mathsf{EXP}(R_{\mathfrak{D}}, t, W)$ is a discovered subweb of R.*

Let $W = (D, data, adoc)$, $R = (D_{\mathfrak{R}}, data_{\mathfrak{R}}, adoc_{\mathfrak{R}})$, $R_{\mathfrak{D}} = (D_{\mathfrak{D}}, data_{\mathfrak{D}}, adoc_{\mathfrak{D}})$, and $\mathsf{EXP}(R_{\mathfrak{D}}, t, W) = (D'_{\mathfrak{D}}, data'_{\mathfrak{D}}, adoc'_{\mathfrak{D}})$. Furthermore, let $t \in \mathsf{AllData}(R_{\mathfrak{D}})$ be an arbitrary RDF triple such that t is a matching triple for a triple pattern $tp \in B$. To prove that $\mathsf{EXP}(W_{\mathfrak{D}}, t, W)$ is a discovered subweb of R we have to show that the three properties in Definition 6.4 (cf. page 118) hold for $D'_{\mathfrak{D}}$, $data'_{\mathfrak{D}}$, and $adoc'_{\mathfrak{D}}$ w.r.t. R (i.e., in this case the Web of Linked Data referred to in Definition 6.4 is R).

Property 1 requires that $D'_{\mathfrak{D}} = D_{\mathfrak{D}} \cup \Delta^W(t)$ is a finite subset of $D_{\mathfrak{R}}$. Since $W_{\mathfrak{D}}$ is a discovered subweb of R, $D_{\mathfrak{D}}$ is a finite subset of $D_{\mathfrak{R}}$. Hence, we only have to show (i) $\Delta^W(t) \subseteq D_{\mathfrak{R}}$ and (ii) $\Delta^W(t)$ is finite (cf. Definition 6.6, page 120). The latter follows from the finiteness of $\mathrm{uris}(t)$. To show the former we use a proof by contradiction, that is, we assume there exists an LD document $d \in \Delta^W(t)$ such that $d \notin D_{\mathfrak{R}}$.

By Definition 6.6, there exists a URI $u \in \mathrm{uris}(t)$ such that $adoc(u) = d$. Let $u^* \in \mathrm{uris}(t)$ be this URI. Since t is a matching triple for a triple pattern $tp \in B$, it must hold that $c_{\mathsf{Match}}(t, u^*, B) = \text{true}$.

From $t \in \mathsf{AllData}(R_{\mathfrak{D}})$ we also know that there exists an LD document $d' \in D_{\mathfrak{D}}$ such that $t \in data_{\mathfrak{D}}(d')$. Let $d^* \in D_{\mathfrak{D}}$ be this document. Since $R_{\mathfrak{D}}$ is a discovered subweb of R, we have $D_{\mathfrak{D}} \subseteq D_{\mathfrak{R}}$ (cf. Definition 6.4). Hence, LD document d^* is (c_{Match}, B)-reachable from S in W. Finally, we may use the fact that R is a reachable subweb (and, thus, an induced subweb) of W to show $d^* \in D$ and $data_{\mathfrak{D}}(d^*) = data_{\mathfrak{R}}(d^*) = data(d^*)$ (cf. Definition 4.3, page 63, and Definition 2.4, page 21).

Putting everything together, we have $d^* \in D$, $t \in data(d^*)$, and $u^* \in \text{uris}(t)$, and we know (i) d^* is (c_{Match}, B)-reachable from S in W, (ii) $c_{\mathsf{Match}}(t, u^*, B) = \text{true}$, and (iii) $adoc(u^*) = d$. Thus, d must be (c_{Match}, B)-reachable from S in W (cf. Definition 4.2, page 62); i.e., $d \in D_{\mathfrak{R}}$. This contradicts our assumption $d \notin D_{\mathfrak{R}}$.

Property 2 follows from Definition 6.6 (cf. page 120), Definition 4.3 (cf. page 63), and the fact that R is a reachable subweb (and, thus, an induced subweb) of W.

We omit showing that *Property 3* holds for $(D'_{\mathfrak{D}}, data'_{\mathfrak{D}}, adoc'_{\mathfrak{D}})$ w.r.t. R; the proof ideas are the same as those that we use in the proof of Proposition 6.5 (cf. page 120). ■

E.11. Proof of Lemma 6.2 (page 126)

Lemma 6.2. *Let W be a Web of Linked Data and let $\mathcal{Q}^{B,S}$ be a $C_{\mathsf{LD(M)}}$ query. At any point during an (arbitrary) execution of tbExec(B, S, W) it holds that (i) each $\sigma \in \mathfrak{P}$ is a partial solution for $\mathcal{Q}^{B,S}$ in W and (ii) $\mathfrak{D}$ is a discovered subweb of the $(S, c_{\mathsf{Match}}, B)$-reachable subweb of W.*

Let $R = (D_{\mathfrak{R}}, data_{\mathfrak{R}}, adoc_{\mathfrak{R}})$ be the $(S, c_{\mathsf{Match}}, B)$-reachable subweb of W. The proof is by induction on the iterations of the main processing loop in *tbExec*(B, S, W) (i.e., lines 3 to 9 in Algorithm 6.1).

Base case: Before the first iteration, *tbExec*(B, S, W) initializes $\mathfrak{P}$ as a set that contains a single element, namely the empty partial solution $\sigma_{\emptyset}$ (cf. line 1 in Algorithm 6.1). The empty partial solution $\sigma_{\emptyset}$ is trivially a partial solution for $\mathcal{Q}^{B,S}$ in W.

$\mathfrak{D}$ is initialized using $\mathfrak{D}_{\mathsf{init}(S,W)}$ (cf. line 2 in Algorithm 6.1), where $\mathfrak{D}_{\mathsf{init}(S,W)}$ is the S-seed subweb of W (cf. Definition 6.5, page 119). Let $\mathfrak{D}_{\mathsf{init}(S,W)} = (D_0, data_0, adoc_0)$. To prove that $\mathfrak{D}_{\mathsf{init}(S,W)}$ is a discovered subweb of R we have to show that the three properties in Definition 6.4 (cf. page 118) hold for $(D_0, data_0, adoc_0)$ w.r.t. R (i.e., in this context the Web of Linked Data referred to in Definition 6.4 is R). Property 1 requires that (i) D_0 is finite and (ii) $D_0 \subseteq D_{\mathfrak{R}}$. The former holds because the set of seed URIs S is finite. The latter holds because each LD document $d \in D_0$ satisfies case 1 in Definition 4.2 (cf. page 62). Properties 2 and 3 follow from (i) $D_0 \subseteq D_{\mathfrak{R}}$, (ii) the definition of $\mathfrak{D}_{\mathsf{init}(S,W)}$ (cf. Definition 6.5), and (iii) the fact that R is a reachable subweb of W and, thus, an induced subweb of W (recall, Definition 4.3, page 63, introduces the concept of reachable subwebs while Definition 2.4, page 21, introduces induced subwebs).

Induction step: Let τ be the open AE task selected in the i-th iteration (cf. line 4 in Algorithm 6.1). The current QE state (for $\mathcal{Q}^{B,S}$ over W) in which the i-th iteration performs τ consists of (versions of) $\mathfrak{P}$ and $\mathfrak{D}$ for which we know from our induction hypothesis that (i) each $\sigma \in \mathfrak{P}$ is a partial solution for $\mathcal{Q}^{B,S}_{c_{\mathsf{Match}}}$ in W and (ii) $\mathfrak{D}$ is a discovered subweb of R. By Proposition 6.6 (cf. page 123), the same holds after performing AE task τ. ■

E.12. Proof of Lemma 6.3 (page 127)

Lemma 6.3. *Let $W = (D, data, adoc)$ be a Web of Linked Data and let $\mathcal{Q}^{B,S}$ be a $\mathcal{C}_{\mathsf{LD(M)}}$ query. There exist executions of tbExec(B, S, W) that have the following two properties:*

(1.) For each LD document $d \in D$ that is (c_{Match}, B)-reachable from S in W there exists an iteration (of the loop in tbExec) after which d is part of $\mathfrak{D}$.

(2.) For each partial solution $\sigma \in \mathcal{O}(\mathcal{Q}^{B,S}, W)$ there exists an iteration after which $\sigma \in \mathfrak{P}$.

As a preliminary for our proof of Lemma 6.3 we introduce the notion of *FIFO-based executions* of *tbExec*, that are, executions of *tbExec* that use a FIFO strategy to choose an open AE task at line 4 of Algorithm 6.1. More precisely, such an execution always chooses an open AE task $\tau \in \mathrm{Open}(\mathfrak{P}, \mathfrak{D})$ for which there does *not* exist another $\tau' \in \mathrm{Open}(\mathfrak{P}, \mathfrak{D})$ such that (i) τ was hidden in an earlier QE state of the execution and (ii) τ' was not hidden in that state. Informally, we note that such an execution resembles a breadth-first search over the link graph of the corresponding reachable subweb of W.

In addition to the symbols introduced in Lemma 6.3, let R denote the $(S, c_{\mathsf{Match}}, B)$-reachable subweb of W. For the proof we assume a FIFO-based execution of our abstract procedure *tbExec*(B, S, W). At any point during this execution let $D_{\mathfrak{D}}$ denote the set of LD documents discovered so far.

(1.) We begin by proving the first claim of Lemma 6.3. Let $d \in D$ be an arbitrary LD document such that d is (c_{Match}, B)-reachable from S in W. We show that during a FIFO-based execution of *tbExec*(B, S, W) there exists an iteration after which $d \in D_{\mathfrak{D}}$ holds. Based on Definition 4.2 (cf. page 62) and since d is (c_{Match}, B)-reachable from S in W, the link graph for W contains a finite path $(d_0, \dots, d_n)$ of (c_{Match}, B)-reachable LD documents $d_i \in D$ (where $0 \leq i \leq n$ and $n \geq 0$) such that (i) $\exists\, u \in S : adoc(u) = d_0$, (ii) $d_n = d$, and (iii) for all $i \in \{1, \dots, n\}$,

$$\exists\, t \in data(d_{i-1}) : \Big(\exists\, u \in \mathrm{uris}(t) : \big(adoc(u) = d_i \text{ and } c_{\mathsf{Match}}(t, u, B) = \text{true}\big)\Big). \quad \text{(E.3)}$$

Let $(d_0, \dots, d_n)$ be such a path. In the following, we show by induction on n that there exists an iteration (during any possible FIFO-based execution of *tbExec*(B, S, W)) after which $D_{\mathfrak{D}}$ contains $d_n = d$.

Base case $(n = 0)$: For this case there exists a seed URI $u \in S$ such that $adoc(u) = d$. Therefore, $d \in D_0$ where D_0 denotes the set of LD document in the S-seed subweb of W (cf. Definition 6.5, page 119). Any execution of *tbExec*(B, S, W) initializes $\mathfrak{D}$ with this S-seed subweb (cf. line 2 in Algorithm 6.1). Thus, even before the first iteration, $d \in D_{\mathfrak{D}}$ holds.

Induction step $(n > 1)$: During a FIFO-based execution of *tbExec*(B, S, W) there exists an iteration it_j after which $d_{n-1} \in D_{\mathfrak{D}}$ holds. We show that there also exists an iteration after which $d_n \in D_{\mathfrak{D}}$ holds. While $\mathfrak{P}$ and $\mathfrak{D}$ are variables in Algorithm 6.1, we denote the particular snapshot of $\mathfrak{P}$ and $\mathfrak{D}$ at the begin of the next iteration after it_j by $\mathfrak{P}_{j+1}$ and $\mathfrak{D}_{j+1}$, respectively.

Due to equation (E.3) there exist an RDF triple $t \in data(d_{n-1})$ and a URI $u \in \text{uris}(t)$ such that $adoc(u) = d_n$ and $c_{\mathsf{Match}}(t, u, B) = \text{true}$. Hence, there must exist a triple pattern $tp \in B$ such that t is a matching triple for tp. Let $tp^* \in B$ be such a triple pattern and let $t^* \in data(d_{n-1})$ be the corresponding, matching triple. Since $\mathfrak{D}_{j+1}$ is a discovered subweb of R (cf. Lemma 6.2), d_{n-1} is (c_{Match}, B)-reachable from S in W and, thus, $t^* \in \mathsf{AllData}(R)$. Therefore, $\tau = (\sigma_\emptyset, t^*, tp^*)$ is an AE task for $\mathcal{Q}^{B,S}$ over W (cf. Definition 6.8, page 122). The partial solution in this task is the empty partial solution $\sigma_\emptyset$. Since $\sigma_\emptyset \in \mathfrak{P}_{j+1}$ (cf. line 1 in Algorithm 6.1) and $t^* \in \mathsf{AllData}(\mathfrak{D}_{j+1})$ (recall our induction hypothesis due to which $d_{n-1} \in D_{\mathfrak{D}}$ after iteration it_j), we know that AE task τ is *not* hidden in QE state $st = (\mathfrak{P}_{j+1}, \mathfrak{D}_{j+1})$ (cf. Definition 6.11, page 124). We distinguish two cases: Either τ is open in st or it is not open (the latter case exists because there may exist other AE tasks with RDF triple t^* and one of them may have been performed in iteration it_j or before).

If AE task τ is not open in QE state st, then $st = \tau[st]$ (cf. Definition 6.12, page 125). Hence, in this case, $d_n \in D_{\mathfrak{D}}$ holds after iteration it_j (or even before).

It remains to discuss the case in which AE task τ is open in QE state st. Since τ is open there exists an iteration $it_{j+\delta}$ (after it_j) in which the FIFO-based execution selects an (open) AE task $\tau' = (\sigma, t, tp)$ where $t = t^*$. This task may either be τ or another AE task with t^*. After selecting τ' the execution performs this task and, thus, expands $\mathfrak{D}$ to $\mathsf{EXP}(\mathfrak{D}, t^*, W)$. This expansion operation results in adding each LD document $d \in \Delta^W(t^*)$ to $D_{\mathfrak{D}}$ (cf. Definition 6.6, page 120). Since there exists a URI $u \in \text{uris}(t^*)$ such that $adoc(u) = d_n$, $d_n \in \Delta^W(t^*)$ holds. Hence, LD document d_n will be added to $D_{\mathfrak{D}}$ in iteration $it_{j+\delta}$ (if it has not been added before).

(2.) We now prove the second claim of Lemma 6.3. Let $\sigma \in \boldsymbol{\sigma}(\mathcal{Q}^{B,S}, W)$ be an arbitrary partial solution for $\mathcal{Q}^{B,S}$ in W. We show that during our FIFO-based execution of $tbExec(B, S, W)$ there exists an iteration after which $\sigma \in \mathfrak{P}$ holds. The construction of σ comprises the iterative construction of a finite sequence $\sigma_0, \dots, \sigma_n$ of partial solutions $\sigma_i = (E_i, \mu_i) \in \boldsymbol{\sigma}(\mathcal{Q}^{B,S}, W)$ (where $0 \leq i \leq n$ and $0 \leq n \leq |B|$) where (i) σ_0 is the empty partial solution $\sigma_\emptyset$, (ii) $\sigma_n = \sigma$, and (iii) for all $i \in \{1, \dots, n\}$,

$$\mu_{i-1}[E_{i-1}] = \mu_i[E_{i-1}] \qquad \text{and} \qquad \exists\, tp \in B \setminus E_{i-1} : E_i = E_{i-1} \cup \{tp\}\,.$$

In the following, we show by induction on n that there exists an iteration (during a FIFO-based execution of $tbExec(B, S, W)$) after which $\mathfrak{P}$ contains $\sigma_n = \sigma$.

Base case ($n = 0$): Any execution of $tbExec(B, S, W)$ adds the empty partial solution $\sigma_\emptyset$ to $\mathfrak{P}$ before it starts the first iteration (cf. line 1 in Algorithm 6.1).

Induction step ($n > 1$): During any FIFO-based execution of $tbExec(B, S, W)$ there exists an iteration it_j after which $\sigma_{n-1} = (E_{n-1}, \mu_{n-1}) \in \mathfrak{P}$ holds. Based on this induction hypothesis we show that there also exists an iteration after which $\sigma_n = (E_n, \mu_n) \in \mathfrak{P}$ holds. We distinguish two cases: Either after iteration it_j it already holds that $\sigma_n \in \mathfrak{P}$ or it still holds that $\sigma_n \notin \mathfrak{P}$. We have to discuss the latter case only.

Let $tp \in B$ be the triple pattern such that $E_n = E_{n-1} \cup \{tp\}$. Since $\sigma_n = (E_n, \mu_n)$ is a partial solution for $\mathcal{Q}^{B,S}$ in W, $\mu_n \in [\![E_n]\!]_{\mathsf{AllData}(R)}$ holds (cf. Definition 6.1, page 113) and, thus, there exists a (c_{Match}, B)-reachable LD document $d \in D$ such that $\mu_n[tp] \in data(d)$.

Let $d^* \in D$ be this document and let $t = \mu_n[tp]$ be the corresponding RDF triple. According to the (previously shown) first claim of Lemma 6.3, there exists an iteration after which $d^* \in D_{\mathfrak{D}}$ holds. By then, either σ_n has already been constructed and added to $\mathfrak{P}$ or there exists an open AE task $\tau = (\sigma_{n-1}, t, tp)$. Again, we have to discuss the latter case only.

Recall that we assume a FIFO-based execution. This execution guarantees the performance of open AE task τ. This performance constructs σ_n and adds it to $\mathfrak{P}$. ■

Bibliography

[1] Daniel J. Abadi, Adam Marcus, Samuel Madden, and Katherine J. Hollenbach. Scalable Semantic Web Data Management Using Vertical Partitioning. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, 2007.

[2] Serge Abiteboul. Querying Semi-Structured Data. In *Proceedings of the 6th International Conference on Database Theory (ICDT)*, 1997.

[3] Serge Abiteboul and Victor Vianu. Queries and Computation on the Web. *Theoretical Computer Science*, 239(2):231–255, 2000.

[4] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[5] Maribel Acosta, Maria Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. ANAPSID: An Adaptive Query Processing Engine for SPARQL Endpoints. In *Proceedings of the 10th International Semantic Web Conference (ISWC)*, 2011.

[6] Ben Adida, Mark Birbeck, Shane McCarron, and Ivan Herman. RDFa Core 1.1 – Syntax and Processing Rules for Embedding RDF through Attributes. W3C Recommendation, Online: `http://www.w3.org/TR/rdfa-core/`, June 2012.

[7] Renzo Angles and Claudio Gutierrez. The Expressive Power of SPARQL. In *Proceedings of the 7th International Semantic Web Conference (ISWC)*, 2008.

[8] Gustavo O. Arocena and Alberto O. Mendelzon. WebOQL: Restructuring Documents, Databases and Webs. In *Proceedings of the 14th International Conference on Data Engineering (ICDE)*, 1998.

[9] Gustavo O. Arocena, Alberto O. Mendelzon, and George A. Mihaila. Applications of a Web Query Language. *Computer Networks and ISDN Systems*, 29(8-13): 1305–1316, 1997.

[10] Sören Auer, Jens Lehmann, and Sebastian Hellmann. LinkedGeoData – Adding a Spatial Dimension to the Web of Data. In *Proceedings of the 8th International Semantic Web Conference (ISWC)*, 2009.

[11] Cosmin Basca and Abraham Bernstein. Avalanche: Putting the Spirit of the Web back into Semantic Web Querying. In *Proceedings of the 6th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2010)*, 2010.

[12] Sotiris Batsakis, Euripides G. M. Petrakis, and Evangelos Milios. Improving the Performance of Focused Web Crawlers. *Data & Knowledge Engineering*, 68(10): 1001–1013, 2009.

[13] Rudolf Bayer and Edward M. McCreight. Organization and Maintenance of Large Ordered Indices. *Acta Informatica*, 1:173–189, 1972.

[14] Tim Berners-Lee. Design Issues: Linked Data. Online at http://www.w3.org/DesignIssues/LinkedData.html, July 2006.

[15] Tim Berners-Lee, Roy Fielding, and Larry Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, online at http://tools.ietf.org/html/rfc3986, January 2005.

[16] Tim Berners-Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, and David Sheets. Tabulator: Exploring and Analyzing linked data on the Semantic Web. In *Proceedings of the 3rd Semantic Web User Interaction Workshop (SWUI)*, 2006.

[17] John Bernier. Announcing BBYOpen Metis Alpha: Best Buy Product Catalog via Semantic Endpoints. Blog post, Online at http://bbyopen.com/announcing-bbyopen-metis-alpha-best-buy-product-catalog-semantic-endpoints, 2012.

[18] Diego Berrueta and Jon Phipps. Best Practice Recipes for Publishing RDF Vocabularies. W3C Working Group Note, Online at http://www.w3.org/TR/swbp-vocab-pub/, August 2008.

[19] Christian Bizer and Andreas Schultz. The berlin sparql benchmark. *International Journal on Semantic Web & Information Systems*, 5(2):1–24, 2009.

[20] Christian Bizer, Tobias Gauß, Richard Cyganiak, and Olaf Hartig. Semantic Web Client Library. Project homepage, Online at http://www4.wiwiss.fu-berlin.de/bizer/ng4j/semwebclient/.

[21] Christian Bizer, Richard Cyganiak, and Tobias Gauss. The RDF Book Mashup: From Web APIs to a Web of Data. In *Proceedings of the 3rd Workshop on Scripting for the Semantic Web (SFSW)*, 2007.

[22] Christian Bizer, Tom Heath, Danny Ayers, and Yves Raymond. Linking Open Data. In *Proceedings of the Poster Session at the 4th European Semantic Web Conference (ESWC)*, 2007.

[23] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data – The Story So Far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009.

[24] Christian Bizer, Anja Jentzsch, and Richard Cyganiak. State of the LOD Cloud. Version 0.3 of this report is online at http://lod-cloud.net/state/2011-09-19/, September 2011.

[25] Christof Bornhövd, Mehmet Altinel, C. Mohan, Hamid Pirahesh, and Berthold Reinwald. Adaptive Database Caching with DBCache. *IEEE Data Engineering Bulletin*, 27(2):11–18, 2004.

[26] Paolo Bouquet, Chiara Ghidini, and Luciano Serafini. Querying The Web Of Data: A Formal Approach. In *Proceedings of the 4th Asian Semantic Web Conference (ASWC)*, 2009.

[27] Brian E. Brewington and George Cybenko. Keeping Up with the Changing Web. *Computer*, 33(5):52–58, 2000.

[28] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, Online at http://www.w3.org/TR/rdf-schema/, February 2004.

[29] Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. SPARQL Web-Querying Infrastructure: Ready for Action? In *Proceedings of the 12th International Semantic Web Conference (ISWC)*, 2013.

[30] Stefan Büttcher and Charles L. A. Clarke. A Hybrid Approach to Index Maintenance in Dynamic Text Retrieval Systems. In *Proceedings of the 28th European Conference on Advances in Information Retrieval (ECIR)*, 2006.

[31] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.

[32] Chia-Hui Chang, Mohammed Kayed, Moheb R. Girgis, and Khaled F. Shaalan. A Survey of Web Information Extraction Systems. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411–1428, 2006.

[33] Surajit Chaudhuri and Umeshwar Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1):65–74, 1997.

[34] Ming-Syan Chen, Ming-Ling Lo, Philip S. Yu, and Honesty C. Young. Using Segmented Right-Deep Trees for the Execution of Pipelined Hash Joins. In *Proceedings of the 18th International Conference on Very Large Data Bases (VLDB)*, 1992.

[35] Alex Coley. Linked Data Pilot Implementations Update. Blog post, Online at http://data.gov.uk/blog/linked-data-pilot-implementations-update, 2012.

[36] Gianluca Correndo, Ian Millard, Hugh Glaser, Nigel Shadbolt, and Manuel Salvadores. SPARQL Query Rewriting for Implementing Data Integration over Linked Data. In *Proceedings of the 1st International Workshop on Data Semantics (DataSem)*, 2010.

[37] Shaul Dar, Michael J. Franklin, Björn THór Jónsson, Divesh Srivastava, and Michael Tan. Semantic Data Caching and Replacement. In *Proceedings of 22th International Conference on Very Large Data Bases (VLDB)*, 1996.

[38] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2nd edition, 2002.

[39] Brian D. Davison. A Web Caching Primer. *IEEE Internet Computing*, 5(4):38–45, 2001.

[40] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused Crawling Using Context Graphs. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)*, 2000.

[41] Li Ding, Lina Zhou, Timothy W. Finin, and Anupam Joshi. How the Semantic Web is Being Used: An Analysis of FOAF Documents. In *Proceedings of the 38th Hawaii International Conference on System Sciences (HICSS)*, 2005.

[42] Li Ding, Joshua Shinavier, Zhenning Shangguan, and Deborah L. McGuinness. SameAs Networks and Beyond: Analyzing Deployment Status and Implications of owl:sameAs in Linked Data. In *Proceedings of the 9th International Semantic Web Conference (ISWC)*, 2010.

[43] Xin Dong, Alon Halevy, and Jayant Madhavan. Reference Reconciliation in Complex Information Spaces. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, 2005.

[44] Eduard C. Dragut, Weiyi Meng, and Clement T. Yu. *Deep Web Query Interface Understanding and Integration*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[45] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1), 2007. doi: http://doi.ieeecomputersociety.org/10.1109/TKDE.2007.9.

[46] Lee Feigenbaum, Gregory Todd Williams, Kendall Grant Clark, and Elias Torres. SPARQL 1.1 Protocol. W3C Recommendation, Online at `http://www.w3.org/TR/sparql11-protocol/`, March 2013.

[47] Roy Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, online at `http://tools.ietf.org/html/rfc2616`, June 1999.

[48] Valeria Fionda, Claudio Gutierrez, and Giuseppe Pirró. Semantic Navigation on the Web of Data: Specification of Routes, Web Fragments and Actions. In *Proceedings of the 21th International World Wide Web Conference (WWW)*, 2012.

[49] Daniela Florescu, Alon Y. Levy, and Alberto O. Mendelzon. Database Techniques for the World-Wide Web: A Survey. *SIGMOD Record*, 27(3):59–74, 1998.

[50] Kevin Ford. LC Classification as Linked Data. *Italian Journal of Library and Information Science*, 4(1), 2013.

[51] Charles Forgy. Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem. *Artificial Intelligence*, 19(1):17–37, 1982.

[52] Hugh Glaser, Afraz Jaffri, and Ian Millard. Managing Co-reference on the Semantic Web. In *Proceedings of the 1st Linked Data on the Web Workshop (LDOW)*, 2009.

[53] Birte Glimm and Chimezie Ogbuji. SPARQL 1.1 Entailment Regimes. W3C Recommendation, Online at `http://www.w3.org/TR/sparql11-entailment/`, March 2013.

[54] Olaf Görlitz and Steffen Staab. SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In *Proceedings of the 2nd International Workshop on Consuming Linked Data (COLD)*, 2011.

[55] Olaf Görlitz and Steffen Staab. Federated Data Management and Query Optimization for Linked Open Data. In Athena Vakali and Lakhmi C. Jain, editors, *New Directions in Web Data Management 1*, pages 109–137. 2011.

[56] Goetz Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2):73–169, 1993.

[57] Jim Gray, Raymond A. Lorie, Gianfranco R. Putzolu, and Irving L. Traiger. Granularity of Locks and Degrees of Consistency in a Shared Data Base. In *Proceedings of the IFIP Working Conference on Modelling in Data Base Management Systems*, 1976.

[58] Tao Guan, Miao Liu, and Lawrence V. Saxton. Structure-Based Queries over the World Wide Web. In *Proceedings of the 17th International Conference on Conceptual Modeling (ER)*, 1998.

[59] Ashish Gupta and Iderpal Singh Mumick. *Materialized Views*, chapter Maintenance of Materialized Views: Problems, Techniques, and Applications, pages 145–157. MIT Press, Cambridge, MA, USA, 1999.

[60] Claudio Gutierrez. Modeling the Web of Data (Introductory Overview). In *Tutorial Lectures of the 7th Reasoning Web Summer School*, 2011.

[61] Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, 1984.

[62] Alon Y. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, 10 (4):270–294, 2001.

[63] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. SPARQL 1.1 Query Language. W3C Recommendation, Online at http://www.w3.org/TR/sparql11-query/, March 2013.

[64] Andreas Harth. Billion Triples Challenge 2011 Data Set. Online at http://km.aifb.kit.edu/projects/btc-2011/, 2011.

[65] Andreas Harth and Stefan Decker. Optimized Index Structures for Querying RDF from the Web. In *Proceedings of the 3rd Latin American Web Congress (LA-Web)*, 2005.

[66] Andreas Harth and Sebastian Speiser. On Completeness Classes for Query Evaluation on Linked Data. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 2012.

[67] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. Data Summaries for On-Demand Queries over Linked Data. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, 2010.

[68] Olaf Hartig. SQUIN. Project homepage, Online at http://squin.org/.

[69] Olaf Hartig. Querying Trust in RDF Data with tSPARQL. In *Proceedings of the 6th Extended Semantic Web Conference (ESWC)*, 2009.

[70] Olaf Hartig. Towards a Data-Centric Notion of Trust in the Semantic Web. In *Proceedings of the 2nd Workshop on Trust and Privacy on the Social and Semantic Web (SPOT)*, 2010.

[71] Olaf Hartig. How Caching Improves Efficiency and Result Completeness for Querying Linked Data. In *Proceedings of the 4th Linked Data on the Web Workshop (LDOW)*, 2011.

[72] Olaf Hartig. Zero-Knowledge Query Planning for an Iterator Implementation of Link Traversal Based Query Execution. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC)*, 2011.

[73] Olaf Hartig. SQUIN: A Traversal Based Query Execution System for the Web of Linked Data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013.

[74] Olaf Hartig. An Overview on Execution Strategies for Linked Data Queries. *Datenbank-Spektrum*, 13(2):89–99, 2013.

[75] Olaf Hartig and Johann-Christoph Freytag. Foundations of Traversal Based Query Execution over Linked Data. In *Proceedings of the 23rd ACM Conference on Hypertext and Social Media (HT)*, 2012.

[76] Olaf Hartig and Ralf Heese. The SPARQL Query Graph Model for Query Optimization. In *Proceedings of the 4th European Semantic Web Conference (ESWC)*, 2007.

[77] Olaf Hartig and Frank Huber. A Main Memory Index Structure to Query Linked Data. In *Proceedings of the 4th Linked Data on the Web Workshop (LDOW)*, 2011.

[78] Olaf Hartig and Andreas Langegger. A Database Perspective on Consuming Linked Data on the Web. *Datenbank-Spektrum*, 10(2):57–66, 2010.

[79] Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. Executing SPARQL Queries over the Web of Linked Data. In *Proceedings of the 8th International Semantic Web Conference (ISWC)*, 2009.

[80] Oktie Hassanzadeh and Mariano P. Consens. Linked Movie Data Base. In *Proceedings of the 2nd Linked Data on the Web Workshop (LDOW)*, 2009.

[81] Patrick Hayes. RDF Semantics. W3C Recommendation, Online at `http://www.w3.org/TR/rdf-mt/`, February 2004.

[82] Bin He, Mitesh Patel, Zhen Zhang, and Kevin Chen-Chuan Chang. Accessing the Deep Web. *Communications of the ACM*, 50(5):94–101, 2007.

[83] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 1st edition, 2011.

[84] Tom Heath and Enrico Motta. Revyu: Linking Reviews and Ratings into the Web of Data. *Journal of Web Semantics*, 6(4):266–273, 2008.

[85] Joseph M. Hellerstein, Michael Stonebraker, and James Hamilton. Architecture of a Database System. *Foundations and Trends in Databases*, 1(2):141–259, 2007.

[86] Rainer Himmeröder, Georg Lausen, Bertram Ludäscher, and Christian Schlepphorst. On a Declarative Semantics for Web Queries. In *Proceedings of the 5th International Conference on Deductive and Object-Oriented Databases (DOOD)*, 1997.

[87] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 Web Ontology Language Primer (Second Edition). W3C Recommendation, Online at `http://www.w3.org/TR/owl2-primer/`, December 2012.

[88] Aidan Hogan, Andreas Harth, Juergen Umrich, Sheila Kinsella, Axel Polleres, and Stefan Decker. Searching and Browsing Linked Data with SWSE: the Semantic Web Search Engine. *Journal of Web Semantics*, 9(4), 2012.

[89] Jiewen Huang, Daniel J. Abadi, and Kun Ren. Scalable SPARQL Querying of Large RDF Graphs. *Proceedings of the VLDB Endowment*, 4(11):1123–1134, 2011.

[90] Amit Krishna Joshi, Prateek Jain, Pascal Hitzler, Peter Z. Yeh, Kunal Verma, Amit P. Sheth, and Mariana Damova. Alignment-based Querying of Linked Open Data. In *Proceedings of the 11th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, 2012.

[91] Ritu Khare, Yuan An, and Il-Yeol Song. Understanding Deep Web Search Interfaces: A Survey. *SIGMOD Record*, 39(1):33–40, September 2010.

[92] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, Online at `http://www.w3.org/TR/rdf-concepts/`, February 2004.

[93] Georgi Kobilarov, Tom Scott, Yves Raimond, Silver Oliver, Chris Sizemore, Michael Smethurst, Christian Bizer, and Robert Lee. Media Meets Semantic Web – How the BBC Uses DBpedia and Linked Data to Make Connections. In *Proceedings of the 6th European Semantic Web Conference (ESWC)*, 2009.

[94] David Konopnicki and Oded Shmueli. W3QS: A Query System for the World-Wide Web. In *Proceedings of 21th International Conference on Very Large Data Bases (VLDB)*, 1995.

[95] David Konopnicki and Oded Shmueli. Information Gathering in the World-Wide Web: The W3QL Query Language and the W3QS System. *ACM Transactions on Database Systems*, 23(4):369–410, 1998.

[96] David Konopnicki and Oded Shmueli. WWW Exploration Queries. In *Proceedings of the 4th International Workshop on Next Generation Information Technologies and Systems (NGITS)*, 1999.

[97] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D. Sivakumar, Andrew Tomkins, and Eli Upfal. The Web as a Graph. In *Proceedings of the 19th Symposium on Principles of Database Systems (PODS)*, 2000.

[98] Zoe Lacroix, Arnaud Sahuguet, R. Chandrasekar, and B. Srinivas. A Novel Approach to Querying the Web: Integrating Retrieval and Browsing. In *Proceedings of the ER Workshop on Conceptual Modeling for Multimedia Information Seeking*, 1997.

[99] Günter Ladwig and Duc Thanh Tran. Linked Data Query Processing Strategies. In *Proceedings of the 9th International Semantic Web Conference (ISWC)*, 2010.

[100] Günter Ladwig and Duc Thanh Tran. SIHJoin: Querying Remote and Local Linked Data. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC)*, 2011.

[101] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira. A Brief Survey of Web Data Extraction Tools. *SIGMOD Record*, 31(2), June 2002.

[102] Laks V. S. Lakshmanan, Fereidoon Sadri, and Iyer 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.

[103] Rob Larson and Evan Sandhaus. NYT to Release Thesaurus and Enter Linked Data Cloud. Blog post, Online at `http://open.blogs.nytimes.com/2009/06/26/nyt-to-release-thesaurus-and-enter-linked-data-cloud/`, 2009.

[104] Wangchao Le, Anastasios Kementsietsidis, Songyun Duan, and Feifei Li. Scalable Multi-Query Optimization for SPARQL. In *Proceedings of the 28th IEEE International Conference on Data Engineering (ICDE)*, 2012.

[105] Nicholas Lester, Justin Zobel, and Hugh E. Williams. In-Place versus Re-Build versus Re-Merge: Index Maintenance Strategies for Text Retrieval Systems. In *Proceedings of the 27th Australasian Conference on Computer Science (ACSC)*, 2004.

[106] Wen-Syan Li, Junho Shim, and K. Selçuk Candan. WebDB: A System for Querying Semi-structured Data on the Web. *Journal of Visual Languages and Computing*, 13(1):3–33, 2002.

[107] Yan Liang, Haofen Wang, Qiaoling Liu, Thanh Tran, Thomas Penin, and Yong Yu. Efficient Index Maintenance for Frequently Updated Semantic Data. In *Proceedings of the 3rd Asian Semantic Web Conference (ASWC)*, 2008.

[108] Lipyeow Lim, Min Wang, Sriram Padmanabhan, Jeffrey Scott Vitter, and Ramesh Agarwal. Efficient Update of Indexes for Dynamically Changing Web Documents. *World Wide Web*, 10(1), 2007.

[109] Mengchi Liu and Tok Wang Ling. A Conceptual Model and Rule-Based Query Language for HTML. *World Wide Web*, 4(1-2):49–77, 2001.

[110] Konstantinos Makris, Nektarios Gioldasis, Nikos Bikakis, and Stavros Christodoulakis. Ontology Mapping and SPARQL Rewriting for Querying Federated RDF Data Sources. In *Proceedings of OTM Conferences*, 2010.

[111] Giansalvatore Mecca, Alberto O. Mendelzon, and Paolo Merialdo. Efficient Queries over Web Views. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT)*, 1998.

[112] Alberto O. Mendelzon and Tova Milo. Formal Models of Web Queries. *Information Systems*, 23(8):615–637, 1998.

[113] Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web. *International Journal on Digital Libraries*, 1(1):54–67, 1997.

[114] Peter Mika and Tim Potter. Metadata Statistics for a Large Web Corpus. In *Proceedings of the 5th Linked Data on the Web Workshop (LDOW)*, 2012.

[115] Daniel P. Miranker, Rodolfo K. Depena, Hyunjoon Jung, Juan F. Sequeda, and Carlos Reyna. Diamond: A SPARQL Query Engine, for Linked Data Based on the Rete Match. In *Proceedings of the Workshop on Artificial Intelligence meets the Web of Data (AImWD) at ECAI*, 2012.

[116] Jeffrey C. Mogul. Squeezing More Bits Out of HTTP Caches. *IEEE Network*, 14 (3):6–14, 2000.

[117] Gabriela Montoya, Luis Daniel Ibáñez, Hala Skaf-Molli, Pascal Molli, and Maria-Esther Vidal. GUN: An Efficient Execution Strategy for Querying the Web of Data. In *Proceedings of the 24th International Conference on Database and Expert Systems Applications (DEXA)*, 2013.

[118] Hannes Mühleisen and Christian Bizer. Web Data Commons – Extracting Structured Data from Two Large Web Corpora. In *Proceedings of the 5th Linked Data on the Web Workshop (LDOW)*, 2012.

[119] Sergio Muñoz, Jorge Pérez, and Claudio Gutierrez. Simple and Efficient Minimal RDFS. *Journal of Web Semantics*, 7(3):220–234, 2009.

[120] M. Muralikrishna and David J. DeWitt. Equi-Depth Histograms For Estimating Selectivity Factors For Multi-Dimensional Queries. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, 1988.

[121] Felix Naumann. *Quality-Driven Query Answering for Integrated Information Systems*. Springer Verlag, 2002.

[122] Thomas Neumann and Guido Moerkotte. Characteristic Sets: Accurate Cardinality Estimation for RDF Queries with Multiple Joins. In *Proceedings of the 27th IEEE International Conference on Data Engineering (ICDE)*, 2011.

[123] Thomas Neumann and Gerhard Weikum. RDF-3X: a RISC-style Engine for RDF. In *Proceedings of the 34th International Conference on Very Large Data Bases (VLDB)*, 2008.

[124] Emmy Noether. Idealtheorie in Ringbereichen. *Mathematische Annalen*, 83:24–66, 1921. URL http://eudml.org/doc/158855.

[125] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1993.

[126] Elien Paret, William Van Woensel, Sven Casteleyn, Beat Signer, and Olga De Troyer. Efficient Querying of Distributed RDF Sources in Mobile Settings based on a Source Index Model. *Procedia CS*, 5:554–561, 2011.

[127] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics of SPARQL. Technical Report TR/DCC-2006-17, Universidad de Chile, October 2006.

[128] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3), 2009.

[129] Eric Prud'hommeaux and Carlos Buil-Aranda. SPARQL 1.1 Federated Query. W3C Recommendation, Online at http://www.w3.org/TR/sparql11-federated-query/, March 2013.

[130] Bastian Quilitz and Ulf Leser. Querying Distributed RDF Data Sources with SPARQL. In *Proceedings of the 5th European Semantic Web Conference (ESWC)*, 2008.

[131] Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4):334–350, 2001.

[132] Yves Raimond, Christopher Sutton, and Mark B. Sandler. Interlinking Music-Related Data on the Web. *IEEE MultiMedia*, 16(2):52–63, 2009.

[133] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, 3rd edition, 2003.

[134] Yanbo Ru and Ellis Horowitz. Indexing the Invisible Web: A Survey. *Online Information Review*, 29(3):249–265, 2005.

[135] Gerard Salton and Michael McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, 1984.

[136] Matthias Samwald, Anja Jentzsch, Christopher Bouton, Claus Kallesøe, Egon L. Willighagen, Janos Hajagos, M. Scott Marshall, Eric Prud'hommeaux, Oktie Hassanzadeh, Elgar Pichler, and Susie Stephens. Linked Open Drug Data for Pharmaceutical Research and Development. *Journal of Cheminformatics*, 3(19), 2011.

[137] Monica Scannapieco, Paolo Missier, and Carlo Batini. Data Quality at a Glance. *Datenbank-Spektrum*, 5(14):6–14, 2005.

[138] Sebastian Schaffert, Christoph Bauer, Thomas Kurz, Fabian Dorschel, Dietmar Glachs, and Manuel 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.

[139] Florian Schmedding. Incremental SPARQL Evaluation for Query Answering on Linked Data. In *Proceedings of the 2nd International Workshop on Consuming Linked Data (COLD) at ISWC*, 2011.

[140] Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of SPARQL Query Optimization. In *Proceedings of the 13th International Conference on Database Theory (ICDT)*, 2010.

[141] Michael Schmidt, Olaf Görlitz, Peter Haase, Günter Ladwig, Andreas Schwarte, and Thanh Tran. FedBench: A Benchmark Suite for Federated Semantic Data Query Processing. In *Proceedings of the 10th International Semantic Web Conference (ISWC)*, 2011.

[142] Michael Schneider. OWL 2 Web Ontology Language, RDF-Based Semantics (Second Edition). W3C Recommendation, Online at `http://www.w3.org/TR/owl2-rdf-based-semantics/`, December 2012.

[143] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. FedX: Optimization Techniques for Federated Query Processing on Linked Data. In *Proceedings of the 10th International Semantic Web Conference (ISWC)*, 2011.

[144] Francois-Paul Servant and Edouard Chevalier. Describing Customizable Products on the Web of Data. In *Proceedings of the 6th Linked Data on the Web Workshop (LDOW)*, 2013.

[145] Amit P. Sheth and James A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[146] Joshua Shinavier. Ripple: Functional Programs as Linked Data. In *Proceedings of the 3rd Workshop on Scripting for the Semantic Web (SFSW)*, 2007.

[147] Oded Shmueli and Alon Itai. Maintenance of Views. *SIGMOD Record*, 14(2): 240–255, 1984.

[148] Pavel Shvaiko and Jérôme Euzenat. Ontology Matching: State of the Art and Future Challenges. *IEEE Transactions on Knowledge Data Engineering*, 25(1): 158–176, 2013.

[149] Lefteris Sidirourgos, Romulo Goncalves, Martin Kersten, Niels Nes, and Stefan Manegold. Column-Store Support for RDF Data Management: Not All Swans are White. *Proceedings of the VLDB Endowment*, 1:1553–1563, 2008.

[150] Michael Sipser. *Introduction to the Theory of Computation.* PWS Publishing, 2nd edition, 2006.

[151] John Miles Smith and Philip Yen-Tang Chang. Optimizing the Performance of a Relational Algebra Database Interface. *Communications of the ACM*, 18(10): 568–579, 1975.

[152] Ellen Spertus and Lynn Andrea Stein. Squeal: A Structured Query Language for the Web. *Computer Networks*, 33(1-6):95–103, 2000.

[153] Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. SPARQL Basic Graph Pattern Optimization Using Selectivity Estimation. In *Proceedings of the 17th International World Wide Web Conference (WWW)*, 2008.

[154] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms.* Pearson Education, 2nd edition, 2007.

[155] Olivier Thereaux, Sofia Angeletou, Jeremy Tarling, and Michael Smethurst. Opening up the BBC's Data to the Web. In *Proceedings of the W3C Open Data on the Web Workshop*, 2013.

[156] Yuan Tian, Jürgen Umbrich, and Yong Yu. Enhancing Source Selection for Live Queries over Linked Data via Query Log Mining. In *Proceedings of the Joint International Semantic Technology Conference (JIST)*, 2011.

[157] Petros Tsialiamanis, Lefteris Sidirourgos, Irini Fundulaki, Vassilis Christophides, and Peter Boncz. Heuristics-based Query Optimisation for SPARQL. In *15th International Conference on Extending Database Technology (EDBT)*, 2012.

[158] Alan Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2*, 42: 230–265, 1936.

[159] Jürgen Umbrich, Katja Hose, Marcel Karnstedt, Andreas Harth, and Axel Polleres. Comparing Data Summaries for Processing Live Queries over Linked Data. *World Wide Web*, 14(5–6):495–544, 2011.

[160] Jürgen Umbrich, Aidan Hogan, Axel Polleres, and Stefan Decker. Improving the Recall of Live Linked Data Querying through Reasoning. In *Proceedings of the 6th International Conference on Web Reasoning and Rule Systems (RR)*, 2012.

[161] Denny Vrandecić, Markus Krötzsch, Sebastian Rudolph, and Uta Lösch. Leveraging Non-Lexical Knowledge for the Linked Open Data Web. In *The 5th Review of April Fool's day Transactions (RAFT)*, 2010.

[162] Andreas Wagner, Thanh Tran, Günter Ladwig, and Andreas Harth. Top-K Linked Data Query Processing. In *Proceedings of the 9th Extended Semantic Web Conference (ESWC)*, 2012.

[163] Jia Wang. A Survey of Web Caching Schemes for the Internet. *ACM Computer Communication Review*, 29(5):36–46, 1999.

[164] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. Hexastore: Sextuple Indexing for Semantic Web Data Management. In *Proceedings of the 34th International Conference on Very Large Data Bases (VLDB)*, 2008.

[165] Reynold S. Xin, Oktie Hassanzadeh, Christian Fritz, Shirin Sohrabi, and Renée J. Miller. Publishing Bibliographic Data on the Semantic Web using BibBase. *Semantic Web Journal*, 4(1):15–22, 2013.

[166] S. Bing Yao. Optimization of Query Evaluation Algorithms. *ACM Transactions on Database Systems*, 4(2):133–155, 1979.

[167] Richard Zanibbi, Dorothea Blostein, and James R. Cordy. A Survey of Table Recognition: Models, Observations, Transformations, and Inferences. *International Journal on Document Analysis and Recognition*, 7(1):1–16, March 2004.

[168] Yue Zhuge, Héctor García-Molina, Joachim Hammer, and Jennifer Widom. View Maintenance in a Warehousing Environment. *SIGMOD Record*, 24(2):316–327, 1995.

[169] Lei Zou, Jinghui Mo, Lei Chen, M. Tamer Özsu, and Dongyan Zhao. gStore: Answering SPARQL Queries via Subgraph Matching. *Proceedings of the VLDB Endowment*, 4(8):482–493, 2011.

Selbstständigkeitserklärung

Ich erkläre hiermit, dass

- ich die vorliegende Dissertationsschrift „Querying a Web of Linked Data" selbständig und ohne unerlaubte Hilfe angefertigt habe;
- ich mich nicht bereits anderwärts um einen Doktorgrad beworben habe oder einen solchen besitze;
- mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin bekannt ist, gemäß amtliches Mitteilungsblatt Nr. 34/2006.

Berlin, den 15. Januar 2014 Olaf Hartig