FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

# Complementary Methods for the Enrichment of Linked Data

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

**Doktor der Technischen Wissenschaften**

by

**Dipl.-Ing. Stefan Bischof**
Registration Number 00327033

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.-Prof. Dr. Axel Polleres

The dissertation has been reviewed by:

| | |
|---|---|
| Prof Dr. Oscar Corcho | Dr. Claudia d'Amato |

Vienna, 12th October, 2017

Stefan Bischof

# Contents

Part I

# Foundations

# 1 Introduction

The futurist Naisbitt wrote in 1982, "we are drowning in information, but we are starved for knowledge" [Naisbitt 1982]. With the advent of the World Wide Web (www) [Berners-Lee 1989], the amount of information in the form of hypertext documents is even more vast and is continuously increasing.

To gain knowledge from an ocean of data we can perform *data analysis*. Regardless of the concrete process a data analyst or data scientist follows—for example Han [2012], Pyle [1999] and Schutt and O'Neil [2013]—one of the first necessary and most time intensive tasks of data analysis is *data preparation*, which includes *data cleaning* and *data integration*. Experts agree that around 60% of the time in data analysis is spent on data preparation alone [Cabena et al. 1998; CloudFlower 2016; Pyle 1999]. Therefore efficient data preparation approaches are crucial.

Search engines were built to automatically structure and find knowledge in this information ocean. But the search engines could not *understand* the contents of the documents they were linking to. This *web of documents* was then augmented and extended with the *web of data*; a process ignited by the seminal idea of the Semantic Web [Berners-Lee, Hendler and Lassila 2001]. The Semantic Web aims to provide a framework of languages and methods to build "a new form of Web content that is *meaningful* (emphasis added) to computers" [Berners-Lee, Hendler and Lassila 2001] and thus eventually allowing machines to automatically produce and ingest information [Heath and Bizer 2011; Berners-Lee 2006; Bernstein, Hendler and Noy 2016]. Nevertheless, the web of data only contributes to the explosion of the amount of information and does not save us from drowning.

Semantic Web technologies aim to provide means to perform data integration and preparation more efficiently and—eventually—in an automated fashion [Janowicz et al. 2015]. In practice, however, even getting comparable data is not trivial; that is different kinds of heterogeneity problems and challenges arise [Bernstein, Hendler and Noy 2016]. Converting data to Semantic Web data formats alone does therefore not resolve heterogeneity, and actual *meaningful* interlinking can only be partially automated and still involves manual intervention. Even though data published following Semantic Web principles constitutes a prime source of openly available data in a unified format, this data still needs preprocessing before being ready for data analysis.

In this work we focus on automatic taxonomic and numerical reasoning

to get more complete query answers necessary for data analysis.

THE MAIN USE CASE we present in this thesis to exemplify the addressed problems and our approach, is the collection, integration and analysis of openly available statistical data of cities worldwide, with the goal to score and compare cities based on this data. City data is an interesting domain because of the societal and economical implications of more and more people living and working in cities: since 2014 more than 50% of the population lives in cities—this ratio is still increasing [United Nations 2015b]. The United Nations (UN) defined in their 2030 Agenda for Sustainable Development [United Nations 2015a] one goal specific to cities "Sustainable Development Goal 11: Make cities and human settlements inclusive, safe, resilient and sustainable". When evaluating progress on this goal the UN also uses statistical city data. Different organisations also acknowledge the importance of cities and city data by publishing analyses of cities such as the Mercer's Quality of Living Ranking [Mercer 2017], the Economist Intelligence Unit's Global Liveability Ranking[The Economist Intelligence Unit 2017] or the Siemens Green City Index [The Economist Intelligence Unit 2012]. These erprts are used by city administration for decision support or for example for companies to aid in computing premium allowances for expatriate employees.

However, these existing reports have several disadvantages: (i) the underlying data is often already outdated when the analysis is published, (ii) the underlying data is not publicly available, making an evaluation of validity or reliability of the results hardly possible; additionally the computation of the resulting scores and rankings is often not documented well enough to reproduce the results and (iii) the underlying data and the resulting scores and rankings are often not reusable, either because of technical or legal reasons.

OUR HYPOTHESIS in the this thesis is that in fact a lot of the underlying data used to generate these reports is already available in one or the other form publicly as *open data* and that—by exploiting Semantic Web technologies—we can collect and integrate this data to eventually enable the up-to-date generation of such reports in an automated fashion while retaining data lineage.

## 1.1 Challenges and Research Questions

Data integration is an important part of data preparation and it is the main use case for Semantic Web technologies. While integration is not always simple to achieve, Semantic Web technologies give us different tools to address it. Apart from data integration other steps of data preparation, e.g. data cleaning, data reduction, data transformation [Han 2012], are hard to achieve with Semantic Web technologies alone. In particular the manipulation of numbers can be cumbersome and is thus often left to dedicated data analysis tool

which are not integrated into a Linked Data based workflow. Therefore information about the data origin and processing is lost in the process. While reasoning about and processing of taxonomic knowledge is well addressed within Semantic Web research, reasoning about and processing numerical data is—with few exceptions—still a widely neglected area.

When building a Semantic Web analysis platform several problems arise, which can not be solved efficiently by engineering but pose interesting research problems. In this work we are concerned with three of such challenges.

### *Reasoning capabilities of SPARQL endpoints unknown*

Linked Data and other RDF data is often accessible via open SPARQL endpoints where data is modelled after OWL ontologies. However, these endpoints often do not or only partially support ontological reasoning (for example on certain fragments of OWL such as RDFS only) or it is unclear whether or to which extent reasoning already was applied to the data. Thus, we need means to assess if reasoning is performed. If not, we still want to be able to do reasoning with the tools SPARQL provides.

In the case of OWL QL ontologies which we are considering in this thesis, a user can still get complete query answers by using *ontology mediated query answering* (OMQA) techniques [Bienvenu and Ortiz 2015]. In OMQA the query is first rewritten into a new query, by taking the ontology into account. For our main use-case however, OMQA has two disadvantages: (i) RDF triple stores do not distinguish between terminological and assertional knowledge like Description Logic and OMQA, thus the ontology must be extracted beforehand for the rewriting step and (ii) the rewritten query is exponentially large in the size of the input query in the worst case [Calvanese, Giacomo et al. 2007].

**Research question 1.** *How can we produce rewritings of SPARQL queries which are independent of the ontology and avoid the exponential blowup of standard query rewriting techniques?*

We aim to push the boundaries of enabling OWL QL on off-the-shelf SPARQL endpoints by query rewriting and without knowing the ontology upfront.

### *Equational background knowledge unusable for reasoning*

Even if ontological reasoning is complete, certain "ontological" background knowledge, such as equations on numeric values, are out of scope of ontological reasoners.

Data analysts working with statistical dataa (or other numeric data in general) are interested in computing indicators to represent some characteristic. For example the *number of unemployed persons* is an important indicator when evaluating the economic situation of a region. But only when normalising this indicator to the population number (usually including only persons

of employable age) of the region and thus computing the *unemployment rate* regions then become comparable.

These computations are usually defined as functions or equations. *Equations* describe relations between numerical attributes. While OWL provides terminological reasoning services, knowledge about attributes (concrete values like numbers or strings) is only weakly supported. Equations are not expressible by OWL 2 alone. Extending OWL to express equations and a corresponding semantics for computation would give us the means to compute new values like the GDP per capita or check numerical attributes for consistency.

**Research question 2.** *How can we express and use equational knowledge about numerical values of instances along with RDFS and OWL to derive new values?*

By using equational knowledge we can transparently compute new numerical values for missing value prediction, for computing derived indicators or simply for converting values between units.

### *Ontological reasoning insufficient for predicting missing values*

Real-world datasets are often missing a part of the data which would be needed for data analysis, e.g. to be able to compute indicators and scores for all desired cities. Ontological reasoning alone is insufficient to impute missing values for analysis because it lacks significant reasoning capabilities for numeric values. Automatic computation of new values based on equational knowledge alone will often still not give enough data to conduct an analysis. Standard statistical missing data methods are also challenged by the high rate of missing values in such datasets. A combination of equationally derived values and statistical methods could improve the overall missing value estimation quality.

**Research question 3.** *How can we combine statistical inference with OWL and equational knowledge to improve missing value imputation?*

With this combined workflow we expect higher quality missing value predictions as well as a higher number of missing value predictions.

OVERALL WE ARE interested to find out to what extent we can use standard off-the-shelf tools such as SPARQL engines to answer the research questions.

## 1.2 Contributions and Structure

We now introduce four contributions of this thesis to address the three research questions.

### *Contribution to Research question 1*

*Schema-agnostic rewriting* We introduce schema-agnostic query rewriting in SPARQL 1.1 as a way to transparently, i.e. independently of the ontology, rewrite a SPARQL query into a new SPARQL query. The rewriting exploits SPARQL 1.1 property path expressions to navigate the ontology at runtime and thus turns an off-the-shelf SPARQL 1.1 engine into an OWL QL reasoner.

We introduced schema-agnostic query rewriting in SPARQL 1.1 at the International Semantic Web Conference 2014 [Bischof, Krötzsch et al. 2014a], and also published a paper at the Description Logic Workshop 2015 [Bischof, Krötzsch et al. 2015]. A journal submission containing an extensive evaluation of schema-agnostic querying is currently in preparation [Bischof, Krötzsch et al. 2017]. Schema-agnostic rewriting is presented in Chapter 3.

### *Contributions to Research question 2*

In general RDF allows two ways to model numerical data: (i) as a numeric literal (attribute) in a binary relation in RDF—the equivalent in OWL being *DataProperty* and (ii) as part of an n-ary (or reified) statement, as for example modelled by the measure dimension in the Data Cube vocabulary [Cyganiak, Reynolds and Tennison 2014]. The following two contributions describe expressing and reasoning with equational knowledge for these two cases.

*RDF attribute equations* We define an extension of the Description Logic equivalent to RDFS to express relations between numerical attributes (equations) for RDF attributes. We give an algorithm for SPARQL query rewriting to transparently compute new values for backward chaining reasoning.

We presented RDF attribute equations at the Extended Semantic Web Conference 2013 [Bischof and Polleres 2013]. These results are presented in Chapter 4.

*QB equations* We introduce QB equations, an approach to compute new numerical (statistical) data by exploiting equational knowledge for multidimensional databases. QB equations use the Data Cube vocabulary and provide detailed provenance information on how values are computed and can propagate error estimates of input values. Error estimates quantify how much reported indicator values potentially deviate from the real value. Such error estimates can stem from observed values or from statistical missing value imputation algorithms. We give an RDF syntax and semantics and present experimental results of a prototype implementation.

A journal submission containing the QB equations as well as an improved version of the Open City Data Pipeline[1] is accepted for publication in the Special Issue on *Semantic Statistics* of the Journal of Web Semantics [Bischof, Harth et al. 2017]. QB equations are described in Chapter 6.

1 http://citydata.wu.ac.at/ocdp

### *Contribution to Research question 3*

*Combined missing value prediction* We introduce a process to fill in missing values which usually occur in Linked Data, especially when integrating data in one domain from different sources. The described process combines statistical missing value imputation applied on data given in the Data Cube vocabulary with QB equations.

We introduced the Open City Data Pipeline and specifically the first part of the combined missing value prediction—two machine learning methods for missing value prediction—at the Know@LOD workshop [Bischof, Martin et al. 2015b] co-located with ESWC 2015. We presented an extension of the above paper on the Open City Data Pipeline at the International Semantic Web Conference 2015 [Bischof, Martin et al. 2015a], where we additionally evaluated the cross-dataset-predictions and automatic learning of indicator mappings between datasets. A journal submission containing the QB equations as well as an improved version of the Open City Data Pipeline is accepted for publication in the Special Issue on *Semantic Statistics* of the Journal of Web Semantics [Bischof, Harth et al. 2017] and served as a basis for Chapter 7.

APART FROM the research questions, our use-case motivates the four previous technical contributions with a concrete application scenario. The Open City Data Pipeline is a platform to collect, integrate, enrich and republish Linked Data about cities based on off-the-shelf Semantic Web technologies.[2]

2 available at http://citydata.wu.ac.at

### *Outline*

Chapter 2 explains shared definitions used in the following chapters. This mainly includes Semantic Web foundations, including description logics and Semantic Web technologies as well definitions of basic data science terms.

Chapter 3 introduces schema-agnostic rewriting. We give formal definitions, some applications and an extensive evaluation.

Chapter 4 introduces RDF attribute equations. We extend a common description logic to support reasoning with equational knowledge and give an experimental evaluation.

Chapter 6 describes QB equations, which express equational knowledge for the Data Cube vocabulary. We specifically introduce an RDF syntax, a semantics based on the evaluation of SPARQL queries until a fixpoint is reached, as well as a experimental evaluation.

Chapter 7 we present a combined workflow integrating statistical machine learning methods with reasoning based on equational knowledge. We show how the combinated method can help in building a system to process data for analysis and further publication in the domain of city data.

Chapter 8 summarises this work and gives concluding remarks. We finish with a list of open questions interesting for future work.

# 2 Preliminaries

This chapter provides the preliminaries needed for the following chapters. In particular this chapter includes an introduction to basic Semantic Web technology specifications such as W3C standards as well as their formal underpinnings. We keep this chapter to the necessary minimum and refer the interested reader to more extensive resources at the end of each section.

Section 2.1 introduces the underlying Semantic Web data model RDF and Section 2.2 describes the corresponding query language SPARQL. OWL 2, a knowledge representation language along with reasoning formalisms applicable to OWL are described in Section 2.3. Section 2.4 describes relevant Linked Data Vocabularies.

## 2.1 Resource Description Framework

The foundation of Semantic Web technologies consists of a layer for data representation. The Resource Description Framework (RDF) specifies the data model of Semantic Web Technologies. With Linked Data we introduce a means to make this data available through web technologies. Vocabularies, we describe two external vocabularies used in later chapters, are shared and reusable domain models, which themselves are described in RDF.

The Resource Description Framework (RDF) is primarily a definition of a graph-based data model to describe any kind of things (called *resources*). RDF distinguishes three types of resources: IRIs, blank nodes and literals. *Internationalized Resource Identifiers* (IRI) are used to *name* resources. A resource could be any physical thing like a rock or a person, or a immaterial thing like a city or the concept of a smart city itself. If, for some reason, it is unnecesary to give a resource an explicit name, a *blank node* can be used; formally a blank node can be seen as an anonymous existential variable. Lastly a *literal* is a data value which can be of one of several types: a simple string like `"Vienna"`, a string with a language tag like `"Wien"@de` where `de` represents the German language, literals with a datatype like `"1741246"^^xsd:int` which represents an integer number [Klyne, Carroll and McBride 2014].

The basic building block of the RDF data model is an RDF triple (or simply *triple*) consisting of a *subject*, *predicate* and *object*, where subject and object are often depicted as nodes and the predicate as the label of the directed link

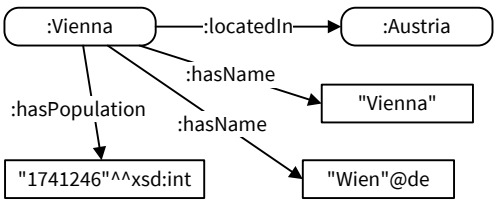
connecting the subject to the object. Each of the three can be an IRI. Subject and object can also be a blank node, i.e. an unlabelled entity. The object can also consist of a literal, i.e. an atomic value in a simple datatype such as string or integer.

**Definition 2.1** (RDF term, RDF triple, RDF graph). Let $\mathbf{I}$ be the set of all IRIs, $\mathbf{L}$ be the set of all literals and $\mathbf{B}$ the set of all blank nodes where all these three sets are pairwise disjoint. The set $\mathbf{L}$ is comprised of all literals with language tags, literals with datatype IRIs and plain literals, i.e., literals with no language tag or datatype IRI. Then the set of *RDF terms* $\mathbf{T}$ is the union of these sets $\mathbf{I} \cup \mathbf{L} \cup \mathbf{B}$. An *RDF triple* is a triple $(s, p, o) \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times \mathbf{T}$ where $s$ is the *subject*, $p$ the *predicate* and $o$ the *object*. A set of RDF triples is an *RDF graph*.

The RDF specificatication originally specified only the RDF/XML syntax—an XML based syntax for the RDF data model [Dave Beckett 2004]. In this work we will use the more compact and human-friendly Turtle syntax instead to denote RDF triples and RDF graphs [David Beckett et al. 2014]. Turtle provides two syntaxes for an IRI, either as IRI enclosed in a pair of angle brackets <http://dbpedia.org/resource/Vienna> or as CURIE [Birbeck and McCarron 2010], a syntax for expressing compact IRIs.[1] With a prefix declaration such as @prefix dbpedia: <http://dbpedia.org/resource/> we can express the same IRI as before as dbpedia:Vienna. Blank nodes can be expressed either as CURIE with the underscore '_' as prefix or as a pair of square brackets. Literals are either given as plain literals, as literals with language tag or as literals with datatype annotation. RDF triples are then written by simply joining subject, predicate and object with white space and terminated by a dot. To reduce redundancy a semicolon can be used to implicitly repeat the subject and a comma to implicitly both subject and predicate.

1 Apart from this initial description we will usually also use the term *IRI* instead of CURIE.

*Example* 2.1. Rounded rectangles depict nodes—more specifically CURIEs in the following example—and rectangles literals. The following RDF graph shows a subject node :Vienna which is connected to the object node :Austria via the predicate :locatedIn. The other three triples show three different forms of literals as objects. While "Vienna" is a plain literal, "Wien"@de is a literal with a language tag for German and "1741246"^^xsd:int is a typed literal, an integer number in this case:




We will use IRIs and data from the DBpedia RDF graph.[2] This graph can then be represented in Turtle syntax as follows:

2 http://dbpedia.org/

```
@prefix dbr: <http :// dbpedia.org/resource/>
@prefix dbo: <http :// dbpedia.org/ontology/>
@prefix foaf : <http :// xmlns.com/foaf /0.1/>
dbr:Vienna a dbo:City .
dbr:Vienna dbo:country dbr: Austria .
dbr:Vienna foaf :name "Vienna" .
dbr:Vienna foaf :name "Wien"@de .
dbr:Vienna dbo:populationTotal "1852997"^^xsd:integer .
```

Using semicolons and commas we can write the same RDF graph more compactly as follows:

```
@prefix dbr: <http :// dbpedia.org/resource/>
@prefix dbo: <http :// dbpedia.org/ontology/>
@prefix foaf : <http :// xmlns.com/foaf /0.1/>
dbr:Vienna a dbo:City ;
  dbo:country dbr: Austria ;
  foaf :name "Vienna", "Wien"@de ;
  dbo:populationTotal "1852997"^^xsd:integer .
```

For a more compact representation we will use the prefix declarations from Appendix A instead of listing them in every example.

*Summary* The RDF as a data model can represent arbitrary resources which are named using IRIs, relations between resources also named using IRIs plus concrete atomic attribute values wich are represented by possibly typed literals. As syntax we use CURIEs and Turtle. The texbook by Hitzler, Krötzsch and Rudolph [2009] contains a complete introduction to RDF.

## 2.2 SPARQL

Just as for relational data, we need tools to access and process data given in RDF. Analogous to SQL for relational data, the query language for RDF is SPARQL [Harris and Seaborne 2013].

### *SPARQL Syntax*

The SPARQL language builds upon basic graph patterns (BGP) as the foundational patterns which map variables to RDF resources, as well as more complex graph patterns built on top of BGPs.

A triple pattern is a simple extension of the RDF triple by allowing a variable in each position and a path expression as predicate.

**Definition 2.2** (triple pattern, BGP). Let $\mathbf{V}$ be the set of variables, disjoint from $\mathbf{T}$. Then a *triple pattern* is a triple $(\mathbf{T} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{T} \cup \mathbf{V})$. A *basic graph pattern* (BGP) is a set of triple patterns.

Path expressions work like regular expressions over edges in RDF graphs and can be used in a SPARQL BGP on predicate position to form path patterns.

**Definition 2.3** (path expression, path pattern). The set of *path expressions* **E** is defined inductively as follows:: (i) Every IRI is a path expression and (ii) For $e$ and $f$ being property expressions, the following expressions are property expressions as well: ($\hat{}e$) for inverse, ($e$ / $f$) for sequence, ($e$ | $f$) for alternative and ($e^*$) for Kleene star. As usual, parentheses can be omitted if there is no danger of confusion. A *path pattern* is a triple $(\mathbf{T} \cup \mathbf{V}) \times \mathbf{E} \times (\mathbf{V} \cup \mathbf{V})$.

Additionally to triple patterns we also allow path patterns in BGPs.

More complex graph patterns allow value creation, filtering, conjunction, disjunction and optional matching.

**Definition 2.4** (graph pattern). The set of *graph patterns* is defined inductively as follows:

- a BGP is a graph pattern
- if $e$ is a SPARQL expression, $p$ is a graph pattern and $?v$ is a variable, then BIND($e$ AS$?v$) is a graph pattern
- if $e$ is a SPARQL expression and $p$ is a graph pattern, then $\{p\}$ FILTER($e$) is a graph pattern
- if $p$ and $q$ are graph patterns, then $\{p\}$ UNION $\{q\}$, $\{p\}$ OPTIONAL $\{q\}$ and $\{p\}$ . $\{q\}$ are graph patterns

SPARQL expressions can be used for filtering but also for value creation.

**Definition 2.5** (SPARQL expression). *SPARQL expressions* are defined inductively:

- a variable is a SPARQL expression
- an RDF term is a SPARQL expression
- if $p$ is a graph pattern, then EXISTS$\{p\}$ is a SPARQL expression
- if $e_1, \ldots, e_n$ are SPARQL expressions and $f$ is one of the SPARQL functions[3] CONCAT, REPLACE, IRI, ABS, NOW, NOT or IF, then $f(e_1, \ldots, e_n)$ is a SPARQL expression
- if $e_1$ and $e_2$ are SPARQL expressions, then $e_1 = e_2$, $e_1 < e_2$, $e_1 <= e_2$, $e_1$ != $e_2$, $e_1 > e_2$ and $e_1 >= e_2$ as well as $e_1$ && $e_2$ and $e_1$ || $e_2$ are SPARQL expressions

3 see Harris and Seaborne [2013] for a complete list of SPARQL functions, we restrict ourselves here to the ones used in the course of this thesis

SPARQL allows three result forms for querying RDF or one form for updating RDF graphs.[4]

4 the other result forms are not relevant for this thesis [Harris and Seaborne 2013]

**Definition 2.6** (SPARQL query, result form). A *SPARQL query* consists of a triple $(R, P, G)$ where $P$ is a graph pattern, $G$ is an RDF graph and $R$ is a *result form* defined as one of four cases as follows:

- SELECT $?v_1 \ldots ?v_n$ is a result form for the variables $v_1, \ldots v_n$
- CONSTRUCT $\{T\}$ is a result form
- ASK is a result form
- INSERT $\{T\}$ is a result form

where $T$ is a set of triple patterns called *graph template*.

A template, as needed for CONSTRUCT and INSERT consists of a set of triple patterns.

SPARQL defines a *dataset* which consists of one so called *default graph* and a set of named graphs, where a *named graph* is a pair $(i, G)$ where $i \in \mathbf{I}$ and $G$ is an RDF graph. Since this practical feature is not necessary in this thesis, we operate on a single graph $G$ instead.

*Example* 2.2. The following query is a SELECT query containing one path pattern

```
SELECT ?thing ?name
WHERE { ?thing dbo:country?/foaf:name ?name }
```

The next example is a CONSTRUCT query and contains two triple patterns in a BGP, a BIND pattern for value creation and a triple pattern in the CONSTRUCT template:

```
CONSTRUCT { ?thing dbo:populationDensity ?populationDensity }
WHERE { ?thing dbo:populationTotal  ?population ;
               dbo:areaTotal  ?area  .
        BIND(?population/?area AS ?populationDensity)  }
```

### *SPARQL Semantics*

We define the semantics of SPARQL queries extending upon J. Pérez, Arenas and Gutierrez [2009]. We thus define a set semantics in contrast of the bag semantics of the SPARQL specification or in other words we ignore mapping cardinalities. First we define the semantics of path expressions.

**Definition 2.7.** We define the *evaluation* of path expressions with respect to an RDF graph $G$ as a binary relation over $\mathbf{I} \cup \mathbf{B}$ in an inductive way: for $p \in \mathbf{I}$, $[\![p]\!]_G = \{(u_1, u_2) \mid u_1\ p\ u_2 \in G\}$, inverse $[\![\hat{}p]\!]_G = \{(u_2, u_1) \mid (u_1, u_2) \in [\![p]\!]_G\}$, sequence $[\![p\ /\ q]\!]_G = \{(u_1, u_3) \mid (u_1, u_2) \in [\![p]\!]_G, (u_2, u_3) \in [\![q]\!]_G\}$, alternative $[\![p \mid q]\!]_G = [\![p]\!]_G \cup [\![q]\!]_G$, Kleene star $[\![p^*]\!]_G = \bigcup_{n \geq 0} [\![p^n]\!]_G$ where $[\![p^0]\!]_G = \{(u, u) \mid u \in \mathbf{I} \cup \mathbf{B}$ occurs in $G\}$ and $[\![p^{n+1}]\!]_G = [\![p^n]\!]_G \circ [\![p]\!]_G$.

Next we define the semantics of triple patterns and BGPs. Blank nodes in BGPs work like scoped variables and are instantiated via a mapping $\sigma$. The mapping $\mu$ then is a partial mapping from variables to RDF terms.

**Definition 2.8.** The *evaluation* $[\![b]\!]_G$ of a basic graph pattern $b$ with respect to an RDF graph $G$ is the set of all partial mappings $\mu$ from variables in $b$ to IRIs, blank nodes or literals of $G$, such that there exists some mapping $\sigma$ from all blank nodes in $b$ to terms of $G$ for which $\mu(\sigma(b)) \in G$. Two mappings $\mu_1$ and $\mu_2$ are *compatible* if for all $?v \in \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2)$ the mappings are the same $\mu_1(?v) = \mu_2(?v)$. The domain $\mathrm{dom}(\mu)$ of a mapping $\mu$ is the set of variables where $\mu$ is defined.

We can now define the evaluation of the other graph patterns over a graph.

**Definition 2.9.** The *evaluation* $[\![p]\!]_G$ of a graph pattern $p$ over an RDF graph $G$ is defined recursively as follows:

- $[\![p_1 \,.\, p_2]\!]_G = \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![p_1]\!]_G, \mu_2 \in [\![p_2]\!]_G \text{ and } \mu_1, \mu_2 \text{ are compatible}\}$
- $[\![p_1 \text{ UNION } p_2]\!]_G = \{\mu \mid \mu \in [\![p_1]\!]_G \text{ or } \mu \in [\![p_2]\!]_G\}$
- $[\![p_1 \text{ OPTIONAL } p_2]\!]_G = [\![p_1 \,.\, p_2]\!]_G \cup \{\mu \in [\![p_1]\!]_G \mid \text{for all } \mu' \in [\![p_2]\!]_G, \mu, \mu' \text{ are not compatible}\}$
- $[\![p \text{ FILTER } e]\!]_G = \{\mu \mid \mu \in [\![p]\!]_G \text{ and } [\![\mu(e)]\!]_G = \mathit{true}\}$
- $[\![p \text{ BIND}(e \text{ AS} ?v)]\!]_G = \{\mu_1 \cup \mu_2 \mid \mu_1 \in [\![p]\!]_G, \mu_2 = \{?v \mapsto [\![\mu_1(e)]\!]_G\}\}$

We say a graph pattern $gp$ has a *match* into a graph $G$ if $[\![gp]\!]_G \neq \emptyset$.

We slightly abuse the notation by extending the mapping $\mu$ to all SPARQL expressions, so the mapping $\mu(e)$ maps a SPARQL expression $e$ to a SPARQL expression $e'$ where all variables are replaced according to $\mu$.[5]

5 Similarly to J. Pérez, Arenas and Gutierrez [2009] we use a two-valued logic here for SPARQL expressions. However the semantics would be easily generalisable to add the third value *error* to the official three-valued logic, see Polleres and Wallner [2013].

**Definition 2.10.** The *evaluation* $[\![e]\!]_G$ of a SPARQL expression $e$ over an RDF graph $G$ is defined recursively as follows:

- for an RDF term $t \in \mathbf{T}$, $[\![t]\!]_G = t$
- for a graph pattern $p$, $[\![\text{EXISTS}\{p\}]\!]_G = \mathit{true}$ if $[\![p]\!]_G \neq \emptyset$ and *false* otherwise
- $[\![\text{CONCAT}(e_1, \ldots, e_n)]\!]_G$ returns the concatenation of the strings $[\![e_1]\!]_G, \ldots, [\![e_n]\!]_G$ to one string
- $[\![\text{REPLACE}(s, p, r)]\!]_G$ returns the string $[\![s]\!]_G$ where the string $[\![r]\!]_G$ replaces matches of the regular pattern $p$ in $s$
- $[\![\text{IRI}(e)]\!]_G$ converts the string $[\![e]\!]_G$ into an IRI
- $[\![\text{ABS}(e)]\!]_G$ returns the absolute value of a number $[\![e]\!]_G$
- $[\![\text{NOW}()]\!]_G$ returns the current date and time
- $[\![\text{NOT}(e)]\!]_G$ returns *true* if $[\![e]\!]_G = \mathit{false}$ and *false* otherwise
- $[\![\text{IF}(c, i, e)]\!]_G$ returns $[\![i]\!]_G$ if $[\![c]\!]_G = \mathit{true}$ and $[\![e]\!]_G$ otherwise
- for the comparison operators $\circ \in \{=, <, <=, >, >=, !=\}$, the evaluation $[\![e_1 \circ e_2]\!]_G$ returns the boolean value of the corresponding comparisons for numbers $[\![e_1]\!]_G$ and $[\![e_2]\!]_G$
- $[\![e_1 \;\&\&\; e_2]\!]_G$ returns *true* if $[\![e_1]\!]_G = \mathit{true}$ and $[\![e_2]\!]_G = \mathit{true}$ and *false* otherwise
- $[\![e_1 \;||\; e_2]\!]_G$ returns *true* if $[\![e_1]\!]_G = \mathit{true}$ or $[\![e_2]\!]_G = \mathit{true}$ and *false* otherwise

Whenever the type of one of the subexpressions does not match the expected type in the definition, the result is undefined.

Finally we define the answers of a complete SPARQL query.

**Definition 2.11.** The set of *answers* of a SELECT query $(\text{SELECT } ?v_1 \ldots ?v_n, P, G)$ is the set obtained by restricting every partial function $\mu \in [\![P]\!]_G$ to the variables $?v_1, \ldots ?v_n$.

The set of *answers* of a CONSTRUCT query $(\text{CONSTRUCT } T, P, G)$ returns the RDF graph obtained by applying each mapping $\mu$ in the evaluation $[\![P]\!]_G$ to each triple in the template $T$. For each blank node in $T$ a fresh blank node is created for each mapping $\mu \in [\![P]\!]_G$.

The *answer* of an ASK query $(\mathsf{ASK}, P, G)$ returns *true* if $[\![P]\!]_G \neq \emptyset$ and *false* otherwise.

The INSERT query $(\mathsf{INSERT}\ T, P, G)$ replaces the RDF graph $G$ with a new RDF graph $G'$ which is the union of $G$ and the RDF graph obtained by applying each mapping $\mu$ in the evaluation $[\![P]\!]_G$ to each triple in the template $T$. As for the CONSTRUCT query, for each blank node in $T$ a fresh blank node is created for each mapping $\mu \in [\![P]\!]_G$.

*Example* 2.3. The first query of Example 2.2 returns the set of mappings (often visualised as a table) with ?thing being mapped to an IRI or blank node representing a city, and ?name being mapped to the name of the city or the country the city belongs to. For the example graph from Example 2.1 we will get the following two results:

| ?thing | ?name |
|---|---|
| dbr:Vienna | "Vienna" |
| dbr:Vienna | "Wien"@de |

If the country dbr:Austria had a triple with the predicate foaf:name as well, we would get a third result.

For the second query of Example 2.2 and the RDF graph from Example 2.1 plus the triple dbr:Vienna dbo:areaTotal 414650000.0"^^xsd:double we would get the following RDF graph as a result:

```
dbr:Vienna dbo:populationDensity 0.0044688 .
```

This gives us the population density of Vienna per $m^2$.

*Summary* SPARQL is an expressive query language for the RDF data model. The textbook by DuCharme [2013] gives a good introduction to SPARQL 1.1. While Hitzler, Krötzsch and Rudolph [2009] cover only the older SPARQL specification [Prud'hommeaux and Seaborne 2008].

## 2.3 OWL and Description Logics

To increase the expressiveness of RDF the W3C also published the RDFS and later the OWL specifications [Brickley and Guha 2014; OWL Working Group 2009].

RDF allows resources to be assigned to classes. RDFS extends RDF with mainly four constructs which are defined below with the other OWL constructs: rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain and rdfs:range.

The Web Ontology Language (OWL) is a language to represent ontologies, which in turn are descriptions about resources, groups of resources and their relations to each other [Hitzler, Krötzsch et al. 2009]. This section introduces first different OWL constructs with the corresponding RDF syntax. Then we

**Table 2.1**: Conversion from OWL to DL syntax

| OWL RDF syntax | DL syntax |
|---|---|
| owl:Thing | $\top$ |
| owl:Nothing | $\bot$ |
| owl:someValuesFrom owl:Thing ; owl:onProperty P | $\exists P$ |
| owl:someValuesFrom C ; owl:onProperty P | $\exists P.C$ |
| B rdfs:subClassOf C | $B_1 \sqsubseteq C_2$ |
| $B_1$ owl:equivalentClass $B_2$ | $B_1 \equiv B_2$ |
| B owl:disjointWith $B_1$ | $B_1 \sqsubseteq \neg\mathbb{B}_2$ |
| $P_1$ owl:inverseOf $P_2$ | $P_1 \equiv P_2^-$ |
| $P_1$ rdfs:subPropertyOf $P_2$ | $P_1 \sqsubseteq P_2$ |
| $Q_1$ owl:equivalentProperty $Q_2$ | $Q_1 \equiv Q_2$ |
| P rdfs:domain A | $\exists P \sqsubseteq A$ |
| P rdfs:range A | $\exists P^- \sqsubseteq A$ |
| $P_1$ owl:propertyDisjointWith $P_2$ | $P_1 \sqsubseteq \neg P_2$ |
| $Q_1$ owl:complementOf $Q_2$ | $Q_1 \equiv \neg Q_2$ |
| owl:AllDisjointClasses | |
| owl:AllDisjointProperties | |
| owl:AllDifferent | |
| x rdf:type A | $A(x)$ |
| x R y | $R(x, y)$ |
| x owl:differentFrom y | $x \neq y$ |

specify the semantics of OWL via a so called description logic. Eventually we introduce *ontology-mediated query answering*.

In this work we restrict ourselves to a subset of the OWL 2 language called the *OWL QL profile*. This profile is one of three distinguished fragments of the OWL language which allow more efficient reasoning for different use-cases.

### *Description Logic*

The semantics if OWL can either be given by the RDF semantics or by a mapping to description logics. The definition we choose is the latter one called the *direct semantics*.

Table 2.1 shows the conversion from OWL to the DL syntax where the equivalence axiom $X \equiv Y$ is used as a short hand for the two corresponding inclusion axioms $X \sqsubseteq Y$ and $Y \sqsubseteq X$. The mapping is based on the OWL to RDF mapping [Patel-Schneider and Motik 2009]. Qualified existential restriction $B' \sqsubseteq \exists R.B$ is only used as syntactic sugar since it can also be written with a fresh property name $R_B$ according to our defined DL as follows:

$$B' \sqsubseteq \exists R_B, \quad \exists R_B^- \sqsubseteq B, \quad R_B \sqsubseteq R$$

In the literature concept disjointness is sometimes alternatively written as $B \sqcap B' \sqsubseteq \bot$; the same applies to property disjointness.

We first define the description logic (DL) DL-Lite $_{\mathcal{R}}$ [Calvanese, Giacomo et al. 2007].

**Definition 2.12.** Let $A$ be an atomic concept name and $P$ be an atomic property name. $B$ is a basic concept and $Q$ is a basic property. $C$ is a complex concept expression and $R$ is a complex property expression. Then concepts and properties are defined as follows:

$$B \coloneqq A \mid \exists Q \qquad\qquad Q \coloneqq P \mid P^{-}$$
$$C \coloneqq B \mid \neg B \qquad\qquad R \coloneqq Q \mid \neg Q$$

**Definition 2.13.** A DL-Lite *knowledge base* (KB) $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. The *TBox* is a set of *class inclusion axioms* $B \sqsubseteq C$ and *property inclusion axioms* $Q \sqsubseteq R$. The *ABox* is a set of *class membership axioms* $A(a)$ and *property membership axioms* $P(a, b)$.

**Definition 2.14.** An *interpretation* $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ consists of a non-empty set $\Delta^{\mathcal{I}}$ called the *object domain* and an *interpretation function* $\cdot^{\mathcal{I}}$ which is defined as follows:

| | |
|---|---|
| $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ | Class assertion |
| $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ | Role assertion |
| $(\exists Q)^{\mathcal{I}} = \left\{ x \mid \exists y.(x, y) \in R^{\mathcal{I}} \right\}$ | Existential restriction |
| $(P^{-})^{\mathcal{I}} = \left\{ (x, y) \mid (y, x) \in R^{\mathcal{I}} \right\}$ | Inverse |
| $(\neg B)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}}$ | Class negation |
| $(\neg Q)^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus Q^{\mathcal{I}}$ | Role negation |

An interpretation $\mathcal{I}$ *satisfies* an axiom $\alpha$, denoted $\mathcal{I} \models \alpha$, in the following cases:

$$\mathcal{I} \models C \sqsubseteq A \text{ if } C^{\mathcal{I}} \subseteq A^{\mathcal{I}}$$
$$\mathcal{I} \models P_1 \sqsubseteq P_2 \text{ if } P_1^{\mathcal{I}} \subseteq P_2^{\mathcal{I}}$$
$$\mathcal{I} \models C(a) \text{ if } a^{\mathcal{I}} \in C^{\mathcal{I}}$$
$$\mathcal{I} \models P(a, b) \text{ if } (a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$$

Finally, an interpretation $\mathcal{I}$ is called a *model* of a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, written $\mathcal{I} \models \mathcal{K}$, if $\mathcal{I}$ satisfies all axioms in $\mathcal{T}$ and $\mathcal{A}$.

### *Ontology-Mediated Query Answering*

Since we are rewriting SPARQL queries to SPARQL queries and are not concerned with providing access to non-RDF databases we use OMQA instead of the more specific *ontology based data access* (OBDA).

The OWL QL profile allows conjunctive query answering in LogSpace data complexity [Motik, Cuenca Grau et al. 2009].

The function Algorithm - rewrities a conjunctive query and a TBox to a union (disjunction) of conjunctive queries.

*Summary* Krötzsch [2012] gives an extensive introduction and discussion of the OWL profiles. Bienvenu and Ortiz [2015] and Kontchakov, Rodriguez-Muro and Zakharyaschev [2013] explain OMQA and OBDA.

**Function** PerfectRef($q$, $\mathcal{T}$)

**Input**: Conjunctive query $q$, TBox $\mathcal{T}$
**Output**: Union of conjunctive queries

$P := \{q\}$
**repeat**
  $P' := P$
  **foreach** $q \in P'$ **do**
    **foreach** $g$ *in* $q$ **do** // expansion
      **foreach** *inclusion axiom* $I$ *in* $\mathcal{T}$ **do**
        **if** $I$ *is applicable to* $g$ **then**
          $P := P \cup \{q[g/\mathrm{gr}(g, I)]\}$
    **foreach** $g_1, g_2$ *in* $q$ **do** // reduction
      **if** $g_1$ *and* $g_2$ *unify* **then**
        $P := P \cup \{\tau(\mathrm{reduce}(q, g_1, g_2))\}$
**until** $P' = P$
**return** $P$

**Function** gr($g$, $I$)

**if** $g = A(x)$ **then**
  **if** $I = A_1 \sqsubseteq A$ **then return** $A_1(x)$
  **else if** $I = \exists P \sqsubseteq A$ **then return** $P(x, \_)$
  **else if** $I = \exists P^- \sqsubseteq A$ **then return** $P(\_, x)$
**else if** $g = P(x, \_)$ **then**
  **if** $I = A \sqsubseteq \exists P$ **then return** $A(x)$
  **else if** $I = \exists P_1 \sqsubseteq \exists P$ **then return** $P_1(x, \_)$
  **else if** $I = \exists P_1^- \sqsubseteq \exists P$ **then return** $P_1(\_, x)$
**else if** $g = P(\_, x)$ **then**
  **if** $I = A \sqsubseteq \exists P^-$ **then return** $A(x)$
  **else if** $I = \exists P_1 \sqsubseteq \exists P^-$ **then return** $P_1(x, \_)$
  **else if** $I = \exists P_1^- \sqsubseteq \exists P^-$ **then return** $P_1(\_, x)$
**else if** $g = P(x, y)$ **then**
  **if** $I = P_1 \sqsubseteq P$ **or** $I = P_1^- \sqsubseteq P^-$ **then return** $P_1(x, y)$
  **else if** $I = P_1 \sqsubseteq P^-$ **or** $I = P_1^- \sqsubseteq P$ **then return** $P_1(y, x)$

## 2.4 Linked Data Vocabularies

The idea of Linked Data (LD) (or the web of data) aims to enable easier publication of RDF data on the web by relying on the Hypertext Transfer Protocol (HTTP) and thus exploiting the existing infrastructure of the WWW to publish and access data. To this end Berners-Lee introduced the following so called *Linked Data principles* [Berners-Lee 2006]:

1. Use URI s as names for things
2. Use HTTP URI s so that people can look up those names.
3. When someone looks up a URI, provide useful information,

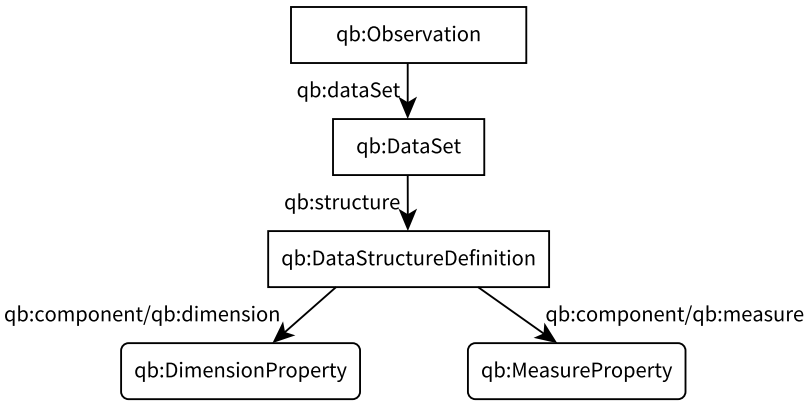



**Figure 2.1**: Illustration of most important classes of the RDF Data Cube Vocabulary with properties (or property chains) between instances of concepts; adapted from "Outline of the vocabulary" in the QB specification.

using the standards (e.g. RDF, SPARQL)

4. Include links to other URI s, so that they can discover more things

Following the W3C specifications and the definitions above we use the term *resource* instead of *thing*. Furthermore *Universal Resource Identifiers* (URI) used in the Linked Data principles where subsumed by IRIs in the later versions of the Semantic Web standards: RDF 1.1 [Klyne, Carroll and McBride 2014], RDFS 1.1 [Brickley and Guha 2014], OWL 2 [OWL Working Group 2009] and SPARQL 1.1 [Harris and Seaborne 2013].

The amount of data published following the Linked Data principles is continuously growing [Abele et al. 2017; Ermilov et al. 2016]. Linked Data sources use lightweight ontologies called *vocabularies* to structure their data. Repositories such as *Linked Open Vocabularies* document a wide range of vocabularies [Vandenbussche et al. 2017]. Most of the top used ontology constructs are covered by OWL QL [Glimm, Adian Hogan et al. 2012]. In this work we reuse two existing vocabularies: the RDF Data Cube Vocabulary for modelling multidimensional numerical data and the PROV vocabulary for modelling provenance information.

### *RDF Data Cube Vocabulary*

The RDF Data Cube Vocabulary (QB) [Cyganiak, Reynolds and Tennison 2014] is a widely-used vocabulary to describe numeric data using a multidimensional data model [Gray et al. 1997]. QB, as a W3C recommendation has established itself as the standard for aggregating and (re-)publishing statistical observations on the web, with off-the-shelf tools to process and visualise QB data. Figure 2.1 provides an overview of QB with the most important classes and properties.

QB allows to describe datasets/cubes (instances of qb:DataSet) with ob-

servations (instances of qb:Observation). We use the terms (statistical) dataset, QB dataset and cube synonymously. Every dataset has a certain structure (instance of qb:DataStructureDefinition (DSD) that—using a chain of properties qb:component before qb:measure or qb:dimension—defines measures (instances of qb:MeasureProperty) and dimensions (qb:DimensionProperty). Attributes (qb:AttributeProperty) allow to add information to observations to qualify and interpret the observed values. A qb:DataSet provides all necessary information about a cube. The qb:DataSet IRI gives the name of the relation in the tabular representation defined by the cube. The qb:DataStructureDefinition—the metadata of the cube/dataset—defines the independent and dependent attributes of the relation as well as their possible attribute values. The qb:Observation instances describe the entities in the relation.

In the following, we illustrate how the metadata of a population dataset can be modelled using QB.

```
eurostat:id/urb_cpop1#ds a qb:DataSet ;
  rdfs:label "Population␣on␣1␣January␣by␣age␣groups␣and␣sex−cities␣and␣greater␣cities";
  qb:structure </dsd/urb_cpop1#dsd>.

eurostat:dsd/urb_cpop1#dsd a qb: DataStructureDefinition ;
  qb:component [ qb:dimension dcterms:date ] ;
  qb:component [ qb:dimension estatwrap:cities ] ;
  qb:component [ qb:dimension estatwrap:indic_ur ] ;
  qb:component [ qb:measure sdmx−measure:obsValue ].
```

In the example, a data structure definition (DSD) defines the independent, categorical properties of the dataset, so-called *dimensions*: date, city and indicator. Also, the DSD defines one dependent numeric property, so-called *measure*. The data structure definition could also include all valid dimension values, such as all city IRIs for the dimension estatwrap:cities.

Now, we give an example of how one data point can be modelled using QB:

```
_:obs1 a qb:Observation ;
  qb:dataSet eurostat:id/urb_cpop1#ds ;
  estatwrap:cities eurostat−cities:Vienna ;
  estatwrap:indic_ur eurostat−indic_ur:Population ;
  dcterms:date "2013" ;
  sdmx−measure:obsValue "1741246".
```

The example describes an observation of 1 741 246 inhabitants of Vienna in 2013 in the population dataset of Eurostat. The observation is modelled with a blank node.

We use the terms (statistical) dataset, QB dataset and (data) cube synonymously. The QB specification defines the notion of *well-formed cubes*[6] based on constraints that need to hold on a dataset. When generating and publishing QB datasets, we ensure that these constraints are fulfilled. For instance, when we later generate new observations via predictions and computations we also generate new datasets containing these values.

6 https://www.w3.org/TR/vocab-data-cube/#wf

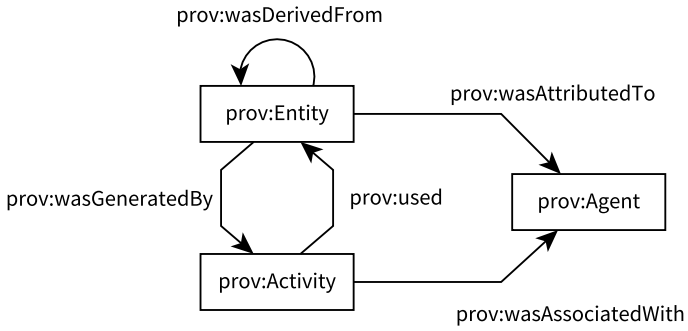



**Figure 2.2:** Overview of the PROV ontology

### *Provenance Annotations*

To make observations more traceable and allow to judge the trustworthiness of data, we go beyond the lightweight approach of using Dublin Core properties such as dc:publisher to refer from a dataset to its publisher. We use the PROV ontology [Lebo, McGuinness and Sahoo 2013] to add provenance annotations, such as the agents and activities that were involved in generating observations from other observations (e.g., predicting, inferencing); Figure 2.2 gives a highlevel overview of the three main concepts. On a high level PROV distinguishes between entities, agents, and activities. A prov:Entity can be all kinds of things, digital or not, which are created or modified. *Activities* are the processes which create or modify entities. An prov:Agent is something or someone who is responsible for an activity (and indirectly also for an entity). A prov:Activity is something that happens and, in our case, generates new observations from other observations. PROV also defines *plans* which can be understood as any kind of predefined workflow which a prov:Activity might follow to create or modify an entity. Additionally PROV also allows to tag certain activities with *time*, for example a timestamp when an entity was created. The following RDF fragment shows a PROV example of two observations, where a QB observation ex:obs123 was derived from another observation ex:obs789 via an activity ex:activity456 on the 15th of January 2017 at 12:37. This derivation was executed according to the rule ex:rule937 with an agent ex:fred being responsible.

```
ex:obs123 prov:generatedAtTime "2017−01−15T12:37:00" ;
  prov:wasDerivedFrom ex:obs789 ;
  prov:wasGeneratedBy ex:activity456  .
ex: activity456  prov: qualifiedAssociation  [
  prov:wasAssociatedWith ex:fred  ]  ;
  prov:hadPlan ex:rule937  .
```

*Summary* Wood et al. [2013] and Heath and Bizer [2011] both introduce Linked Data. Moreau and Groth [2013] describe provenance with the PROV ontology while the best resource for the RDF Data Cube Vocabulary is still

the official specification [Cyganiak, Reynolds and Tennison 2014].

Part II

# Theory

# 3 Schema-Agnostic Query Rewriting

As a first contribution we describe our idea of exploiting SPARQL 1.1 for query rewriting to implement OWL QL reasoning in a single query. Parts of this chapter have been published as Bischof, Krötzsch et al. [2014a], Bischof, Krötzsch et al. [2014b] and Bischof, Krötzsch et al. [2015].

Section 3.1 gives an introduction of schema-agnostic query rewriting and compares it to ontology-based techniques. Section 3.2 introduces OWL QL and relate its semantics to a *chase* procedure. Section 3.3 develops queries for implementing basic OWL QL reasoning in SPARQL 1.1. Section 3.4 extends these queries into a schema-agnostic query rewriting procedure for conjunctive queries. Section 3.5 defines three approaches to optimize query evaluation of schema-agnostic query rewriting. Section 3.6 evaluates schema-agnostic rewriting via a prototype implementation. The prototype is compared against an ontology-based implementation and the different optimization approaches are evaluated as well.

## 3.1 Introduction

SPARQL 1.1, the recent revision of the W3C SPARQL standard, introduces significant extensions to the capabilities of the RDF query language [Harris and Seaborne 2013]. Even at the very core of the query language, we can find many notable new features, including *property paths*, *value creation* (BIND), inline data (VALUES), negation, and extended filtering capabilities. In addition, SPARQL 1.1 now supports query answering over OWL ontologies, taking full advantage of ontological information in the data [Glimm and Ogbuji 2013].

Query answering in the presence of ontologies is known as *ontology-based data access* (OBDA) or *ontology-mediated query answering* (OMQA)[1] and has long been an important topic in applied and foundational research. Even before SPARQL provided support for this feature, several projects have used ontologies to integrate disparate data sources or to provide views over legacy databases, e.g. [Calvanese, Giacomo et al. 2007; Pérez-Urbina, Motik and Horrocks 2010; Rodriguez-Muro, Kontchakov and Zakharyaschev 2013; Di Pinto et al. 2013; Kontchakov, Rodriguez-Muro and Zakharyaschev 2013]. The W3C OWL 2 Web Ontology Language includes the OWL QL language profile, which was specifically designed for this application [Motik, Cuenca Grau et

1 we will use OBDA and OMQA synonymously in this thesis

al. 2009]. With the arrival of SPARQL 1.1, every aspect of OBDA is thus supported by tailor-made W3C technologies.

In practice, however, SPARQL and OWL QL are rarely integrated. Most works on OBDA address the problem of answering *conjunctive queries* (CQs), which correspond to SELECT-PROJECT-JOIN queries in SQL, and (to some degree) to BGPs in SPARQL. The most common approach for OBDA is *query rewriting*, where a given CQ is rewritten into a (set of) CQs that fully incorporate the schema information of the ontology. The answers to the rewritten queries (obtained without considering the ontology) are guaranteed to agree with the answers of the original queries (over the ontology). This approach separates the ontology (used for query rewriting) from the rest of the data (used for query answering), and it is typical that the latter is stored in a relational database. Correspondingly, the rewritten queries are often transformed into SQL for query answering. SPARQL and RDF do not play a role in this.

In this chapter, we thus take a fresh look on the problem of OBDA query rewriting with SPARQL 1.1 as our target query language. The additional expressive power of SPARQL 1.1 allows us to introduce a new paradigm of *schema-agnostic query rewriting*, where the ontological schema is not needed for rewriting queries. Rather, the ontology is stored *together with the data* in a single RDF database. This is how many ontologies are managed today, and it corresponds to the W3C view on OWL and RDF, which does not distinguish schema and data components. The fact that today's OBDA approaches separate both parts testifies to their focus on relational databases. Our work, somewhat ironically, widens the scope of OWL QL to RDF-based applications, which have hitherto focused on OWL RL as their ontology language of choice.

Another practical advantage of schema-agnostic query rewriting is that it supports frequent updates of both data and schema. The rewriting system does not need any information on the content of the database under query, while the SPARQL processor that executes the query does not need any support for OWL. This is particularly interesting if a database can only be accessed through a restricted SPARQL query interface that does not support reasoning. For example, we have used our approach to check the consistency of DBpedia under OWL semantics, using only the public Live DBpedia SPARQL endpoint[2] (it is inconsistent: every library is inferred to belong to the mutually disjoint classes "Place" and "Agent").

2 http://live.dbpedia.org/sparql

Worst-case reasoning complexity remains the same in all cases, yet our approach is much more practical in the case of standard reasoning and BGP rewriting. For general CQs, the rewritten queries are usually too complex for today's RDF databases to handle. Nevertheless, we think that our "SPARQL 1.1 implementation" of OWL QL query answering is a valuable contribution, since it reduces the problem of supporting OWL QL in an RDF database to the task of optimizing a single (type of) query. Since OWL QL subsumes RDFS, one can also apply our insights to implement query answering under RDFS ontologies, which again leads to much simpler queries.

## 3.2 OWL QL: RDF Syntax and Rule-Based Semantics

OWL QL is one of the OWL 2 profiles, which restricts the OWL DL ontology language to ensure that reasoning is tractable [Motik, Cuenca Grau et al. 2009]. To ensure compatibility with SPARQL, we work only with the RDF representation of OWL QL here [Patel-Schneider and Motik 2009]. Like OWL DL, OWL QL requires "standard use" of RDFS and OWL vocabulary, i.e., special vocabulary that is used to encode ontology axioms in RDF is strictly distinct from the ontology's vocabulary, and can only occur in specific triple patterns [Bruijn and Heymans 2007; Muñoz, J. Pérez and Gutiérrez 2007]. Only a few special IRIs, such as owl:Thing, can also be used like ontology vocabulary in axioms.

In the RDF serialisation of OWL, classes, properties and individuals are represented by RDF elements, where complex class expressions and complex property expressions are represented by blank nodes. Whether an expression is represented by an IRI or a blank node does not have an impact on ontological entailment, so we ignore this distinction in most cases. OWL DL allows us to use a single IRI to represent an individual, a class and a property in the same ontology; owing to the restrictions of standard use, it is always clear which meaning applies in a particular case.[3] Hence we will also work with one single set of IRIs.

3 The OWL specification uses the term *punning* for this feature [Hitzler, Krötzsch et al. 2009].

Next, we define the constrains that an RDF graph has to satisfy to represent an OWL QL ontology. We thus transfer the syntactical constraints of DL-Lite in Section 2.3 to the RDF representation of OWL. To this end, consider a fixed RDF graph $G$. A *property expression in* $G$ is an IRI or a blank node $x$ that occurs in a pattern $\{x\ \mathsf{owl{:}inverseOf}\ P\}$ with $P \in \mathbf{I}$. We use **PRP** for the set of all property elements in a given RDF graph. OWL QL further distinguishes two types of class expressions with different syntactic constraints. The set **SBC** of *subclasses in* $G$ consists of all IRIs and all blank nodes $x$ that occur in a pattern $\{x\ \mathsf{owl{:}onProperty}\ P; \mathsf{owl{:}someValuesFrom}\ \mathsf{owl{:}Thing}\}$, where $P \in \mathbf{PRP}$. The set **SPC** of *superclasses in* $G$ is defined recursively as follows. An element $x$ is in **SPC** if it is in **I**, or if it is in **B** and $G$ contains one of the following patterns:

- $\{x\ \mathsf{owl{:}onProperty}\ \mathbf{PRP}; \mathsf{owl{:}someValuesFrom}\ y\}$ where $y \in \mathbf{SPC}$;
- $\{x\ \mathsf{owl{:}intersectionOf}\ (y_1, \ldots, y_n)\}$ where $y_1, \ldots, y_n \in \mathbf{SPC}$;
- $\{x\ \mathsf{owl{:}complementOf}\ y\}$ where $y \in \mathbf{SBC}$.

$G$ is an *OWL QL ontology* and may use the following triple patterns to encode *axioms*:

- $\{\mathbf{I}\ \mathbf{PRP}\ \mathbf{I}\}$
- $\{\mathbf{I}\ \mathsf{rdf{:}type}\ \mathbf{SPC}\}$
- $\{\mathbf{SBC}\ \mathsf{rdfs{:}subClassOf}\ \mathbf{SPC}\}$
- $\{\mathbf{SBC}\ \mathsf{owl{:}equivalentClass}\ \mathbf{SBC}\}$
- $\{\mathbf{SBC}\ \mathsf{owl{:}disjointWith}\ \mathbf{SBC}\}$
- $\{\mathbf{PRP}\ \mathsf{rdfs{:}range}\ \mathbf{SPC}\}$

- {**PRP** rdfs:domain **SPC**}
- {**PRP** rdfs:subPropertyOf **PRP**}
- {**PRP** owl:equivalentProperty **PRP**}
- {**PRP** owl:inverseOf **PRP**}
- {**PRP** owl:propertyDisjointWith **PRP**}
- {**I** owl:differentFrom **I**}
- {**B** rdf:type owl:AllDisjointClasses; owl:members (**SBC**, . . . , **SBC**)}
- {**B** rdf:type owl:AllDisjointProperties; owl:members (**PRP**, . . . , **PRP**)}
- {**B** rdf:type owl:AllDifferent; owl:members (**I**, . . . , **I**)}

$G$ is an OWL QL ontology if every triple in $G$ is part of a unique axiom or a unique complex class or property definition used in such axioms. For simplicity, we ignore triples used in annotations or ontology headers. Moreover, we do not consider the OWL QL property characteristics symmetry, asymmetry and global reflexivity. Asymmetry and reflexivity are not a problem, but their explicit treatment would inflate our presentation considerably. Symmetry, in contrast, cannot be supported with SPARQL 1.1, as we have shown in Bischof, Krötzsch et al. [2014a]. This is no major limitation of our approach, since symmetry can be expressed using inverses. This shows that rewritability of an ontology language does not depend on ontological expressiveness alone.

The semantics of OWL QL is inherited from OWL DL. However, since the OWL QL profile does not support any form of disjunctive information, one can also describe the semantics by defining a *universal model*, i.e., a structure that realizes all entailments of an ontology but no additional entailments. Such a "least model" exactly captures the semantics of an ontology.

To define a universal model for OWL QL, we define a set of RDF-based inference rules, similar to the rules given for OWL RL in the standard [Motik, Cuenca Grau et al. 2009]. In contrast to OWL RL, however, the application of rules can introduce new elements to an RDF graph, and the universal model that is obtained in the limit is not finite in general. Indeed, our goal is not to give a practical reasoning algorithm, but to define the semantics of OWL QL in a way that is useful for analysing the correctness of the rewriting algorithms we introduce.

The main rules for reasoning in OWL QL are defined in Table 3.1. A rule is *applicable* if the premise on the left matches the current RDF graph and the conclusion on the right does not match the current graph; in this case, the conclusion is added to the graph. In case of rule (3.2), this requires us to create a fresh blank node. In all other cases, we only add new triples among existing elements. Rules like (3.3) are actually schemas for an infinite number of rules for lists of any length $n$ and any index $i \in \{1, \ldots, n\}$. Rules (3.15)–(3.16) cover owl:Thing and owl:topObjectProperty, which lead to conclusions that are true for "all" individuals. To ensure standard use, we cannot simply assert $x$ rdf:type owl:Thing for *every* IRI $x$, and we restrict instead to IRIs that are used as individuals in the ontology.

**Table 3.1**: RDF inference rules for OWL QL

| | | |
|---|---|---|
| | → [] rdf:type owl:Thing | (3.1) |
| ?X rdf:type [owl:onProperty ?P; owl:someValuesFrom ?C] | → ?X ?P [rdf:type ?C] | (3.2) |
| ?X rdf:type [owl:intersectionOf ($?C_1, \ldots, ?C_i, \ldots, ?C_n$)] | → ?X rdf:type $?C_i$ | (3.3) |
| ?X rdf:type ?C . ?C rdfs:subClassOf ?D | → ?X rdf:type ?D | (3.4) |
| ?X rdf:type ?C . ?C owl:equivalentClass ?D | → ?X rdf:type ?D | (3.5) |
| ?X rdf:type ?C . ?D owl:equivalentClass ?C | → ?X rdf:type ?D | (3.6) |
| ?X ?P ?Y . ?C owl:onProperty ?P; owl:someValuesFrom owl:Thing | → ?X rdf:type ?C | (3.7) |
| ?X ?P ?Y . ?P rdfs:domain ?C | → ?X rdf:type ?C | (3.8) |
| ?X ?P ?Y . ?P rdfs:range ?C | → ?Y rdf:type ?C | (3.9) |
| ?X ?P ?Y . ?P owl:inverseOf ?Q | → ?Y ?Q ?X | (3.10) |
| ?X ?P ?Y . ?Q owl:inverseOf ?P | → ?Y ?Q ?X | (3.11) |
| ?X ?P ?Y . ?P rdfs:subPropertyOf ?Q | → ?X ?Q ?Y | (3.12) |
| ?X ?P ?Y . ?P owl:equivalentProperty ?Q | → ?X ?Q ?Y | (3.13) |
| ?X ?P ?Y . ?Q owl:equivalentProperty ?P | → ?X ?Q ?Y | (3.14) |
| INDIVIDUAL(?X) | → ?X rdf:type owl:Thing | (3.15) |
| ?X rdf:type owl:Thing . ?Y rdf:type owl:Thing | → ?X owl:topObjectProperty ?Y | (3.16) |

We define INDIVIDUAL($x$) to be the following SPARQL pattern:

{$x$ rdf:type owl:NamedIndividual} UNION
{$x$ rdf:type ?C . ?C rdf:type owl:Class} UNION
{$x$ ?P ?Y . ?P rdf:type owl:ObjectProperty} UNION
{?Y ?P $x$ . ?P rdf:type owl:ObjectProperty}

Note that this also covers any newly introduced individuals.

**Definition 3.1.** The *chase* $G'$ of an OWL QL ontology $G$ is a possibly infinite RDF graph obtained from $G$ by fair application of the rules of Table 3.1, meaning that every rule that is applicable has eventually been applied.

Finally, some features of OWL QL can only make the ontology inconsistent, but not introduce any other kinds of positive entailments, according patterns are shown in Table 3.2. If any of these match, the ontology is inconsistent, every OWL axiom is a logical consequence, and there is no universal model.

The following corollary follows straightforwardly from Definition 3.1 and entailment for the two cases of a consistent and of an inconsistent ontology.

**Corollary 3.1.** *Consider an OWL QL ontology $G$ with chase $G'$ and a basic graph pattern $P$. A variable mapping $\mu$ is a solution for $P$ over $G$ under the OWL DL entailment regime iff either (1) $\mu$ is a solution for $P$ over $G'$ under simple entailment, or (2) one of the patterns of Table 3.2 matches $G'$.*

**Table 3.2:** RDF inference patterns for inconsistency in OWL QL

| | |
|---|---|
| ?X owl:bottomObjectProperty ?Y | (3.17) |
| ?X rdf:type owl:Nothing | (3.18) |
| ?X rdf:type ?C . ?X rdf:type [owl:complementOf ?C] | (3.19) |
| ?X rdf:type ?C . ?X rdf:type ?D . ?C owl:disjointWith ?D | (3.20) |
| ?X rdf:type $?C_i$ . ?X rdf:type $?C_j$. | |
| _:b rdf:type owl:AllDisjointClasses; owl:members $(?C_1, \ldots, ?C_i, \ldots, ?C_j, \ldots, ?C_n)$ | (3.21) |
| ?X ?P ?Y . ?X ?Q ?Y . ?P owl:propertyDisjointWith ?Q | (3.22) |
| ?X $?P_i$ ?Y . ?X $?P_j$ ?Y . _:b rdf:type owl:AllDisjointProperties; owl:members $(?P1, \ldots, ?P_i, \ldots, ?P_j, \ldots, ?P_n)$ | (3.23) |
| ?X owl:differentFrom ?X | (3.24) |
| _:b rdf:type owl:AllDifferent; owl:members $(?I_1, \ldots, ?X, \ldots, ?X, \ldots, ?I_n)$ | (3.25) |

## 3.3 OWL QL Reasoning with SPARQL Path Expressions

Next, we define SPARQL 1.1 queries to solve standard reasoning tasks of the OWL QL profile. We start with simple cases and then consider increasingly complex reasoning problems.

We first focus on the property hierarchy. An axiom of the form $p$ rdfs:subPropertyOf $q$ is entailed by an ontology $G$ if, for newly introduced individuals $a$ and $b$, $G \cup \{a\ p\ b\}$ entails $\{a\ q\ b\}$. By Corollary 3.1, the rules of Section 3.2 represent all possibilities for deriving this information. In this particular case, we can see that only rules (3.10)–(3.14) in Table 3.1 can derive a triple of the form $a\ q\ b$, where $q$ is a regular property. The case $q$ = owl:topObjectProperty is easy to handle, since $p$ rdfs:subPropertyOf owl:topObjectProperty is always true (which is also shown by rules (3.15) and (3.16)). In addition, it might be that $G \cup \{a\ p\ b\}$ is inconsistent, implied by rules of Table 3.2; we will ignore this case for now, since it requires more powerful reasoning.

**Definition 3.2.** We introduce sPO, invOf and eqP as abbreviations for rdfs:subPropertyOf, owl:inverseOf and owl:equivalentProperty, respectively, and define the following composite property path expressions

$$\begin{aligned} \textsc{SpoEqp} &:= (\text{sPO} \mid \text{eqP} \mid \hat{}\text{eqP}) \\ \textsc{Inv} &:= (\text{invOf} \mid \hat{}\text{invOf}) \\ \textsc{subPropertyOf} &:= (\textsc{SpoEqp} \mid (\textsc{Inv} \,/\, \textsc{SpoEqp}^* \,/\, \textsc{Inv}))^* \\ \textsc{subInvPropertyOf} &:= \textsc{SpoEqp}^* \,/\, \textsc{Inv} \,/\, \textsc{subPropertyOf} \end{aligned}$$

Moreover, for an arbitrary term $x$, let $\textsc{univProperty}[x]$ be the following pattern: $\{\text{owl:topObjectProperty } (\textsc{SpoEqp} \mid \textsc{Inv})^*\ x\}$.

The pattern subPropertyOf does not check for property subsumption that is caused by the inconsistency rules in Table 3.2, but it can be used to check for subsumptions related to owl:topObjectProperty. This relies on the following correctness property of the pattern $\textsc{univProperty}[p]$. We provide a par-

ticularly detailed proof here, since many of our later correctness properties will rely on similar arguments.

**Lemma 3.1.** *Consider a consistent* OWL QL *ontology $G$ with property $p \in$* **PRP**. *Then $G$ entails* owl:topObjectProperty rdfs:subPropertyOf $p$ *if and only if the pattern* univProperty$[p]$ *matches $G$.*

*Proof.* For the "if" direction, we assume that the pattern univProperty$[p]$ matches $G$. We need to show that $G$ entails owl:topObjectProperty rdfs:subPropertyOf $p$. Using Corollary 3.1, this is equivalent to the claim: the triple _:a $p$ _:b can be derived by applying the deduction rules of Table 3.1 to $G \cup \{$_:a owl:topObjectProperty _:b$\}$. In particular, we know that the latter is consistent, since otherwise $G$ would clearly be inconsistent as well.

Thus assume a path $(\text{SpoEqp} \mid \text{Inv})^n$ of length $n \geq 0$ from owl:topObjectProperty to $p$. We show the claim by induction on $n$. For $n = 0$, $p =$ owl:topObjectProperty and the claim is immediate. For $n > 0$, let $p'$ be the element in the that is reached after $n - 1$ steps in the path (and for which the claim was already shown by induction). We distinguish cases according to which of the optional properties $q$ in the pattern connects $p'$ to $p$:

- If $q =$ rdfs:subPropertyOf, then we can apply rule (3.12) to derive _:a $p$ _:b from _:a $p'$ _:b. Since the latter can be derived from $G \cup \{$_:a owl:topObjectProperty _:b$\}$ by the induction hypothesis, the claim follows.
- The cases $q =$ owl:equivalentProperty and $q =$ ^owl:equivalentProperty are similar using rules (3.13) and (3.14), respectively.
- If $q =$ owl:inverseOf, we can use the same argument as before to obtain a derivation of _:a $p$ _:b from $G \cup \{$_:b owl:topObjectProperty _:a$\}$, using rule (3.10) in the last step. Note that we apply the induction hypothesis to an input with _:a and _:b swapped. To get the desired derivation, we note that _:b owl:topObjectProperty _:a can be derived from _:a owl:topObjectProperty _:b by applying rule (3.15) to _:a and _:b, followed by rule (3.16).
- The case $q =$ ^owl:inverseOf is again similar, using rule (3.11).

For the "only if" direction, assume that _:a $p$ _:b can be derived from the ontology $G \cup \{$_:a owl:topObjectProperty _:b$\}$ by applying the deduction rules. This can only be accomplished by applying rules (3.12)–(3.16). Moreover, we can assume without loss of generality that (3.15) and (3.16) are only applied at the beginning of the derivation to obtain {_:b owl:topObjectProperty _:a} from {_:a owl:topObjectProperty _:b} (the latter being the only interesting derivation that rule (3.16) could produce here). Thus, to simplify our claim, consider a derivation of _:a $p$ _:b can be derived from the following:

$$G \cup \{\text{\_:a owl:topObjectProperty \_:b}, \text{\_:b owl:topObjectProperty \_:a}\}$$

The proof is by induction on the length $\ell$ of this derivation We claim that there is a path of the form $(\text{SpoEqp} \mid \text{Inv})^n$ of length $n \geq 0$ from owl:topObjectProperty to $p$.

If $\ell = 0, p =$ owl:topObjectProperty and the claim is immediate (with $n = 0$). For $\ell > 0$, we distinguish cases according to the rule applied in the last step of the derivation:

- Rule (3.12), (3.13) or (3.14) applied to a previous consequence {_:a $p'$ _:b}. Then $G$ contains a triple $p'$ $q$ $p$ for $q =$ rdfs:subPropertyOf, $q =$ owl:equivalentProperty, or $q =$ ^owl:equivalentProperty, respectively. By the induction hypothesis, there is a path as in the claim from owl:topObjectProperty to $p'$. We can extend this path by $p'$ $q$ $p$.
- Rule (3.10) or (3.11) applied to a previous consequence {_:b $p'$ _:a}. Then $G$ contains a triple $p'$ $q$ $p$ for $q =$ owl:inverseOf or $q =$ ^owl:inverseOf, respectively. The induction hypothesis applies since we can always swap _:a and _:b in a derivation. Thus there is a path as in the claim from owl:topObjectProperty to $p'$. We can extend this path by $p'$ $q$ $p$.
- Rules (3.15) cannot occur by our assumption on the derivation. □

The following result shows the essential correctness property of subPropertyOf on consistent ontologies.

**Proposition 3.1.** *Consider an OWL QL ontology $G$ with properties $p, q \in$ PRP such that $G \cup$ {_:a $p$ _:b} is consistent. Then $G$ entails $p$* rdfs:subPropertyOf *$q$ iff the pattern* {$p$ subPropertyOf $q$} UNION univProperty[$q$] *matches $G$.*

*Proof.* For the "if" direction, we have to show that the above described calculus allows to derive the triple _:a $q$ _:b from $G \cup$ {_:a $p$ _:b} whenever the pattern {$p$ subPropertyOf $q$} UNION univProperty[$q$] matches $G$. We consider both cases of the UNION expression.

First, let $p$ subPropertyOf $q$ be the matching pattern of the query, that is, we find some $n \in \mathbb{N}$ and a path from $p$ to $q$ matching the regular expression (SpoEqp | (Inv / SpoEqp* / Inv))$^n$. We show the claim via an induction over $n$. For $n = 0$ we obtain $p = q$, therefore _:a $q$ _:b holds by assumption.

For the induction step, assume the claim holds for $n$ and consider a path matching the expression (SpoEqp | (Inv / SpoEqp* / Inv))$^{n+1}$, which means that there is an individual $p'$ such that there is a path matching (SpoEqp | (Inv / SpoEqp* / Inv))$^n$ from $p$ to $p'$ and a path matching (SpoEqp | (Inv / SpoEqp* / Inv)) from $p'$ to $q'$. By induction hypothesis, there is a derivation for _:a $p'$ _:b (†). Now we further analyze the path from $p'$ to $q'$:

- If SpoEqp matches this path then, for each of the possible sub-cases of SpoEqp (viz. rdfs:subPropertyOf, owl:equivalentProperty, or ^owl:equivalentProperty) we find an appropriate rule (namely rule (3.12), (3.13), or (3.14) of Table 3.1, respectively) to derive _:a $q$ _:b from _:a $p'$ _:b.
- If the path from $p'$ to $q$ is matched by (Inv / SpoEqp* / Inv), there are individuals $q'$ and $q''$, such that there are (i) an Inv path from $p'$ to $q'$, (ii) a path from $q'$ to $q''$ matching SpoEqp$^k$ for some $k \geq 0$, and (iii) an Inv path from $q''$ to $q$.

From (†) and (i), we can obtain _:b $q'$ _:a (‡) via rule 3.10 or 3.11. Given (‡) and (ii), we can perform another induction over $k$, recalling the above argument regarding SpoEqp, to arrive at _:b $q''$ _:a. Now, exploiting (iii) and rule 3.10 or 3.11 once more, we finally obtain _:a $q'$ _:b as claimed.

For the second part of the UNION expression, we note that from _:a $p$ _:b, we can infer _:a owl:topObjectProperty _:b by means of rule 3.15 and 3.16. Then, we can invoke Lemma 3.1 to arrive at _:a $q$ _:b as claimed.

For the "only if" direction, assume $G$ is such that _:a $q$ _:b can be derived from $G \cup \{$_:a $p$ _:b$\}$ by applying the deduction rules. If _:a $q$ _:b can be derived from $G \cup \{$_:a owl:topObjectProperty _:b$\}$, then the claim follows from Lemma 3.1. For the remaining case, we can restrict to derivations of _:a $q$ _:b using rules (3.10)–(3.14). Clearly, any such derivation is linear, with each rule applying to a triple in $G$ and a triple of the form _:a $q'$ _:b or _:b $q'$ _:b. Let $p = q_0, \ldots, q_n = q$ be the sequence of properties used in the latter. Only rules (3.10) and (3.11) can swap the order of _:a and _:b, hence there must be an even number of applications of these rules in the derivation. It is easy to see that the expression subPropertyOf hatches exactly these sequences of properties $q_0 \ldots q_n$. □

We will extend this to cover the inconsistent case in Theorem 3.1 below. First, however, we look at entailments of class subsumptions. In this case, the main rules are (3.2)–(3.9). However, several of these rules also depend on property triples derived by rules (3.10)–(3.14), and we apply our results on property subsumption to take this into account.

**Definition 3.3.** Let eqC and sCO abbreviate owl:equivalentClass and rdfs:subClassOf, respectively. We define property path expressions

```
intListMember := (owl:intersectionOf / rdf:rest* / rdf:first)
     someProp := (owl:onProperty / subPropertyOf/
                    (^owl:onProperty | rdfs:domain))
  somePropInv := (owl:onProperty / subInvPropertyOf / rdfs:range)
   subClassOf := (sCO | eqC | ^eqC | intListMember |
                    someProp | somePropInv)*
```

Moreover, we let univClass$[x]$ denote the following pattern:

```
{owl:Thing subClassOf x} UNION
          {owl:topObjectProperty ((SpoEqp | Inv)* /
           (^owl:onProperty | rdfs:domain | rdfs:range)/subClassOf) x}
```

We can use subClassOf to check if a superclass expression in $G$ is subsumed by a subclass expression in $G$; in particular, this applies to class names. As before, we exclude the possibility that one of the classes is incoherent (i.e., entailed to be equivalent to owl:Nothing).

**Proposition 3.2.** *Consider an* OWL QL *ontology* $G$ *with classes* $c \in$ **SPC** *and* $d \in$ **SBC** *such that* $G \cup \{\_\text{:a rdf:type } c\}$ *is consistent. Then* $G$ *entails* $c$ rdfs:subClassOf $d$ *iff the pattern* $\{c$ SUBCLASSOF $d\}$ UNION UNIVCLASS$[d]$ *matches* $G$.

*Proof.* For the "if" direction, we have to show that the above described calculus allows to derive the triple _:a rdf:type $d$ from $G \cup \{\_\text{:a rdf:type } c\}$ whenever the pattern $\{c$ SUBCLASSOF $d\}$ UNION UNIVCLASS$[d]$ matches $G$. We consider both cases of the UNION expression.

First, let $c$ SUBCLASSOF $d$ be the matching pattern of the query, that is, we find some $n \in \mathbb{N}$ and a path from $c$ to $d$ matching the regular expression (sCO | eqC | ^eqC | INTLISTMEMBER | SOMEPROP | SOMEPROPINV)$^n$. We show the claim via an induction over $n$. For $n = 0$ we obtain $a = b$, therefore _:a rdf:type $d$ holds by assumption. For the induction step, assume the claim holds for $n$ and consider a path matching the expression (sCO | eqC | ^eqC | INTLISTMEMBER | SOMEPROP | SOMEPROPINV)$^{n+1}$ which means that there is a class $c'$ such that there is a path matching (sCO | eqC | ^eqC | INTLISTMEMBER | SOMEPROP | SOMEPROPINV)$^n$ from $c$ to $c'$ and a path matching (sCO | eqC | ^eqC | INTLISTMEMBER | SOMEPROP | SOMEPROPINV) from $c'$ to $d'$. By induction hypothesis, we can deduce that _:a rdf:type $c'$ must hold (†). Now we further analyze the path from $c'$ to $d'$ by separately considering the 6 disjunctive options:

- $c'$ sCO $d'$: we can use the induction hypothesis and rule (3.4) to infer the triple _:a rdf:type $d'$.
- $c'$ eqC $d'$: we can use the induction hypothesis and rule (3.5) to infer the triple _:a rdf:type $d'$.
- $c'$ ^eqC $d'$: we can use the induction hypothesis and rule (3.6) to infer _:a rdf:type $d'$.
- $c'$ INTLISTMEMBER $d'$: presuming $G$ to be a well-formed OWL QL graph, the regular expression INTLISTMEMBER does only match inside a structure of the shape $x$ owl:intersectionOf $(x_1, \ldots, x_n)$ and connects $x$ with some $x_i$. Then by the induction hypothesis and rule (3.3) we can infer _:a rdf:type $d'$.
- $c'$ SOMEPROP $d'$: in this case there must exist $p$ and $q$ connected by a path matching SUBPROPERTYOF such that $G$ also contains the triples $c'$ owl:onProperty $p$ and either $d'$ owl:onProperty$p$ or $p$rdfs:domain $d'$. Again assuming well-formedness of $G$ we can apply rule (3.2) to infer _:a $p$ _:b for some fresh bnode _:b. Consequently, due to Proposition 3.1, we can infer _:a $q$ _:b. Finally applying either rule (3.7) or rule (3.8), we arrive at _:a rdf:type $d'$.
- $c'$ SOMEPROPINV $d'$: in this case there must exist $p$ and $q$ connected by a path matching SUBINVPROPERTYOF such that $G$ also contains the triples $c'$ owl:onProperty$p$ and $p$rdfs:range $d'$. We can apply rule (3.2) to infer _:a $p$ _:b for some fresh bnode _:b. Consequently, exploiting the argument in the proof of Proposition 3.1, we can infer _:b $q$ _:a. Finally applying rule (3.9), we arrive at _:a rdf:type $d'$.

**Table 3.3:** Pattern EMPTYCLASS[$x$] for detecting empty classes.

```
x (sCO | eqC | ^eqC | INTLISTMEMBER | owl:someValuesFrom |
   (owl:onProperty / (INV | SPOEQP)* / (^owl:onProperty | rdfs:domain | rdfs:range)))*
    ?C . {
  {?C SUBCLASSOF owl:Nothing} UNION
  {?C SUBCLASSOF ?D1 . {{?C SUBCLASSOF ?D2} UNION UNIVCLASS[?D2]} . {
    {?D1 DISJOINTCLASSES ?D2} UNION
    {?V rdf:type owl:AllDisjointClasses . TWOMEMBERS[?V, ?D1, ?D2]}
  }} UNION
  {?C (owl:onProperty / (INV | SPOEQP)*) ?P . {
    {?P SUBPROPERTYOF owl:bottomObjectProperty} UNION
    {?P SUBPROPERTYOF ?Q1
      {{?P SUBPROPERTYOF ?Q2} UNION UNIVPROPERTY[?Q2]} . {
      {?Q1 (owl:PropertyDisjointWith | ^owl:PropertyDisjointWith) ?Q2} UNION
      {?V rdf:type owl:AllDisjointProperties . TWOMEMBERS[?V, ?Q1, ?Q2]}
    }}
  }
}}
```

The case for UNIVCLASS[$d$] being the matching pattern can be shown in a way analogous to the one above, additionally using Rule (3.15) and rule (3.16) for the base cases.

For the "only if" direction, we have to analyze all possible proofs. For this, it is helpful to distinguish two cases: one where a proof can be found that directly applies the rule (3.15) to all occurrences of proof-tree leafs carrying _:a rdf:type $d$. In such a case, a match to UNIVCLASS[$d$] can be constructed from the proof. In all other cases we can construct a match to {$c$ SUBCLASSOF $d$} in way very analogous to the argument in Proposition 3.2. □

It remains to identify classes that are incoherent, i.e., for which the triple $c$ rdfs:subClassOf owl:Nothing is entailed. To do this, we need to consider the patterns of Table 3.2.

**Definition 3.4.** For arbitrary terms $x$, $y$ and $z$, let TWOMEMBERS[$x, y, z$] be the pattern {$x$(owl:members/rdf:rest*)?W.?W rdf:first $y$.?W(rdf:rest+/rdf:first)$z$}, and let DISJOINTCLASSES be the property path expression (owl:disjointWith | ^owl:disjointWith | owl:complementOf | ^owl:complementOf). The query patterns EMPTYCLASS[$x$] and EMPTYPROPERTY[$x$] are defined as in Table 3.3 and Table 3.4 respectively.

As their name suggests, the patterns of the previous definition allow us to detect classes and properties that must be empty in every model of the ontology. To prove this, we first make some simpler observations:

**Table 3.4**: Pattern EMPTYPROPERTY[$x$] for detecting empty properties.

```
x (INV | SPOEQP |
   (^owl:onProperty / (sCO | eqC | ^eqC | INTLISTMEMBER | owl:someValuesFrom)* /
      owl:onProperty))* ?P . {
   {?P SUBPROPERTYOF owl:bottomObjectProperty} UNION
   {?P SUBPROPERTYOF ?Q1
      {{?P SUBPROPERTYOF ?Q2} UNION UNIVPROPERTY[?Q2]} {
         {?Q1 (owl:PropertyDisjointWith | ^owl:PropertyDisjointWith) ?Q2} UNION
         {?V rdf:type owl:AllDisjointProperties . TWOMEMBERS[?V, ?Q1, ?Q2]}
   }} UNION
   {?P ((^owl:onProperty | rdfs:domain | rdfs:range) / SUBCLASSOF) ?C . {
      {?C SUBCLASSOF owl:Nothing} UNION
      {?C SUBCLASSOF ?D1 {{?C SUBCLASSOF ?D2} UNION UNIVCLASS[?D2]} {
         {?D1 DISJOINTCLASSES ?D2} UNION
         {?V rdf:type owl:AllDisjointClasses . TWOMEMBERS[?V, ?D1, ?D2]}
      }
   }}
}}
```

**Lemma 3.2.** *The pattern* TWOMEMBERS[$x, y, z$] *matches an ontology $G$ iff $G$ contains an* RDF *list $x$ with two distinct elements $y$ and $z$.*

**Lemma 3.3.** *Consider a consistent* OWL QL *ontology $G$ with class $c$. Then $G \cup$* {_:a rdf:type $c$} *is inconsistent iff the pattern* EMPTYCLASS[$c$] *matches $G$.*

*Proof.* The general structure of the proof is as in Lemma 3.1, but with a lot more cases to consider. We sketch the arguments in order to avoid getting lost in details here.

First, we can show a property of the first two lines of the pattern in Table 3.3. Namely, the variable ?C in the pattern generally represents a class that must be non-empty whenever the class $x$ ($c$ in our claim) is non-empty. Formally: $G$ has a match for the pattern

```
c  (rdfs:subClassOf | owl:equivalentClass | ^owl:equivalentClass |
                    INTLISTMEMBER | owl:someValuesFrom |
                       (owl:onProperty / (INV | SPOEQP)* /
                                 (^owl:onProperty | rdfs:domain | rdfs:range))*  d
```

if and only if $G \cup \{d$ rdfs:subClassOf owl:Nothing$\}$ is consistent but the following is inconsistent: $G \cup \{$_:a rdf:type $c, d$ rdfs:subClassOf owl:Nothing$\}$.

This is shown by easy inductions as in Lemma 3.1. Most importantly, we need to observe that non-emptiness of classes can directly follow from the rules (3.1)–(3.9) and (3.15). The cases of (3.1) and (3.15) are not of interest, since they infer non-emptiness of owl:Thing: $d$ in our claim cannot be a superclass of owl:Thing as this would make $G \cup \{d$ rdfs:subClassOf owl:Nothing$\}$

inconsistent. Of the remaining rules, (3.2)–(3.6) are are covered by the options rdfs:subClassOf, owl:equivalentClass, ^owl:equivalentClass, INTLISTMEMBER and owl:someValuesFrom in the pattern, respectively.

For the remaining cases, we need to take derivations of property assertion triples into account. The only relevant rule to derive such triples from premises of the form _:a rdf:type $c'$ is (3.2). After this, further property triples are inferred as in Proposition 3.1 using rules (3.10)–(3.14), corresponding (as shown before) to the expression (INV | SPOEQP)*. Again, owl:topObjectProperty is not of interest here since we assume $G \cup \{d$ rdfs:subClassOf owl:Nothing$\}$ to be consistent. Finally, property assertion triples can be used to transfer new class assertion triples in rules (3.7)–(3.9), corresponding to the final options (^owl:onProperty | rdfs:domain | rdfs:range) in the pattern. This correspondence of rules and pattern can be exploited to obtain the desired result by two inductions, as before.

The remaining parts of the pattern in Table 3.3 lists relevant cases in which the non-emptiness of the class represented by ?C would lead to inconsistency. These cases correspond to the patterns in Table 3.2. The cases (3.18)–(3.21) are covered by the patterns in the third to seventh line of Table 3.2, where we use (reasoning similar to) Proposition 3.2 for the essential correctness of subpatterns UNIVCLASS[?D2] and SUBCLASSOF. Cases (3.17), (3.22) and (3.23) are covered by the remaining lines, where we use Lemma 3.1 and Proposition 3.1 for the essential correctness of subpatterns SUBPROPERTYOF and UNIVPROPERTY[?Q2]. Cases (3.24) and (3.25) can only be violated by $G$ initially and thus do not require checking here.

Lemma 3.2 provides the essential correctness of the pattern TWOMEMBERS$[x, y, z]$ used in several places. For cases (3.17), (3.22) and (3.23), we need to consider property assertion triples that are derived from the non-emptiness of ?C; the pattern used to find a non-empty property ?P is the same pattern that already occurred on the second line, and the same reasoning applies.

Using the, by now obvious, correspondences between inference rules and patterns, we can thus prove the overall claim. □

**Lemma 3.4.** *Consider a consistent OWL QL ontology $G$ with property $p$. Then $G \cup \{$_:a $p$ _:b$\}$ is inconsistent iff the pattern* EMPTYPROPERTY$[p]$ *matches $G$.*

*Proof.* The proof follows the same arguments as the proof of Lemma 3.3. Indeed, many of the subpatterns used are the same, with the main difference being that we now start the derivation from property assertion triples rather than from class assertion triples. □

We can now completely express OWL QL schema reasoning in SPARQL 1.1:

**Theorem 3.1.** *An OWL QL ontology $G$ is inconsistent iff it has a match for the*

*pattern*

$$\begin{aligned}&\{\text{?X rdf:type ?C . } \textsc{emptyClass}[\text{?C}]\} \text{ UNION}\\&\{\text{?X ?P ?Y . } \textsc{emptyProperty}[\text{?P}]\} \text{ UNION}\\&\{\text{?X owl:differentFrom ?X}\} \text{ UNION}\\&\{\text{?V rdf:type owl:AllDifferent . } \textsc{twoMembers}[\text{?V}, \text{?X}, \text{?X}]\}.\end{aligned} \tag{3.26}$$

*$G$ entails $c$* rdfs:subClassOf *$d$ for $c \in$* **SPC** *and $d \in$* **SBC** *iff $G$ is either inconsistent or has a match for the pattern*

$$\{c \textsc{ subClassOf } d\} \text{ UNION } \textsc{univClass}[d] \text{ UNION } \textsc{emptyClass}[c]. \tag{3.27}$$

*$G$ entails $x$* rdf:type *$c$ iff $G$ is either inconsistent or has a match for the pattern*

$$\begin{aligned}&\{\{x \text{ (rdf:type / } \textsc{subClassOf}) \ c\} \text{ UNION}\\&\ \{x \text{ ?P ?Y . ?P (} \textsc{subPropertyOf} \text{ / (\^{}owl:onProperty | rdfs:domain)/}\\&\qquad \textsc{subClassOf}) \ c\} \text{ UNION}\\&\ \{\text{?Y ?P } x \text{ . ?P (} \textsc{subPropertyOf} \text{ / rdfs:range / } \textsc{subClassOf}) \ c\}\\&\} \text{ UNION } \textsc{univClass}[c]\end{aligned} \tag{3.28}$$

*$G$ entails $p$* rdfs:subPropertyOf *$q$ for $p, q \in$* **PRP** *iff $G$ is either inconsistent or has a match for the pattern*

$$\{p \textsc{ subPropertyOf } q\} \text{ UNION } \textsc{univProperty}[q] \text{ UNION } \textsc{emptyProperty}[p]. \tag{3.29}$$

*$G$ entails $x \ p \ y$ iff $G$ is either inconsistent or has a match for the pattern*

$$\begin{aligned}&\{x \text{ ?R } y \text{ . ?R } \textsc{subPropertyOf } p\} \text{ UNION}\\&\{y \text{ ?R } x \text{ . ?R } \textsc{subInvPropertyOf } p\} \text{ UNION}\\&\textsc{univProperty}[p].\end{aligned} \tag{3.30}$$

*Proof.* In each of the cases, we can show correctness using similar techniques as in the proof of Lemma 3.1 and the subsequent proofs shown in this section. We consider each case individually.

For (3.26), correctness is an easy consequence of Lemmata 3.3 and 3.4, together with the observation that the two last lines of (3.26) correspond to the cases (3.24) and (3.25) in Table 3.2. We need to use rules (3.1) (for the first time) and (3.16) to see that (3.26) also covers the cases where owl:Thing is a subclass of owl:Nothing, or where owl:topObjectProperty is a subproperty of owl:bottomObjectProperty.

For (3.27), correctness follows from Proposition 3.2 and Lemma 3.3.

For (3.28), we see from Proposition 3.2 why the first and last line are correct. However, Proposition 3.2 only covers derivations that start from a class assertion triple. When checking for the type of an individual in (3.28), the derivation might also start at property assertion triples given in the ontology. Our arguments in the proof of Proposition 3.2 covered property assertion triples, but only as an intermediate stage of the derivation. It is not hard to see that the second and third line of (3.28) are similar to the respective expressions SOMEPROP and SOMEPROPINV in Definition 3.3, and correctness is shows using the same reasoning as in the proof of Proposition 3.2.

For (3.29), correctness follows from Proposition 3.1 and Lemma 3.4.

For (3.30), correctness is an easy consequence of the same reasoning as in the proof of Proposition 3.1, together with the easy observation that SUBINVPROPERTYOF is similar to SUBPROPERTYOF but swaps the sides. Note that only the rules (3.10)–(3.14) are relevant for normal derivations (not involving owl:topObjectProperty, which is covered by Lemma 3.1). □

## 3.4 OWL QL Query Rewriting with SPARQL 1.1

We now turn towards query answering over OWL QL ontologies using SPARQL 1.1. Research in OWL QL query answering typically considers the problem of answering *conjunctive queries* (CQs), which are conjunctions of OWL property and class assertions that use variables only in the place of individuals, not in the place of properties or classes. Conjunction can easily be represented by a Basic Graph Pattern in SPARQL, yet CQs are not a subset of SPARQL, since they also support existential quantification of variables. Normal query variables are called *distinguished* while existentially quantified variables are called *non-distinguished*. Distinguished variables can only bind to elements of the ontology, whereas for non-distinguished variables it suffices if the ontology implies that some binding must exist.

*Example* 3.1. Consider an OWL ontology with the assertion :peter rdf:type :Person and the following axiom:

:Person rdfs:subClassOf [owl:onProperty :father; owl:someValuesFrom :Person]

This implies that :peter has some :father but the ontology may not contain any element of which we know that it plays this role. In this case, the SPARQL pattern {?X :father ?Y} would not have a match with ?X = :peter under OWL DL entailment. In contrast, if the variable ?Y were non-distinguished, the query would match with ?X = :peter (and ?Y would not receive any binding).

SPARQL can only express CQs where all variables are distinguished. To define this fragment of SPARQL, recall that the OWL DL entailment regime of SPARQL 1.1 requires every variable to be *declared* for a certain type (individual, object property, datatype property, or class) [Glimm and Ogbuji 2013]. This requirement is the analogue of "standard use" on the level of query patterns, and it allows us to focus on instance retrieval here. We thus call a Basic Graph Pattern $P$ CQ-*pattern* if

1. $P$ does not contain any OWL, RDF, or RDFS IRIs other than rdf:type in property positions,
2. all variables in $P$ are declared as required by the OWL DL entailment regime,
3. property variables occur only in predicate positions,
4. class variables occur only as objects of triples with predicate rdf:type.

Rewriting CQ-patterns is an easy application of Theorem 3.1:

**Definition 3.5.** For a triple pattern $x$ rdf:type $c$, the rewriting $[\![x \text{ rdf:type } c]\!]$ is the graph pattern (3.28) as in Theorem 3.1; for a triple pattern $x\ p\ y$, the rewriting $[\![x\ p\ y]\!]$ is the graph pattern (3.30). The rewriting $[\![P]\!]$ of a CQ-pattern $P$ is obtained by replacing every triple pattern $s\ p\ o$ in $P$ by $\{[\![s\ p\ o]\!]\}$.

The following theorem follows from Definition 3.5, Theorem 3.1 and the proof of Theorem 3.1.

**Theorem 3.2.** *If $G$ is the RDF graph of a consistent OWL QL ontology, then the matches of a CQ-pattern $P$ on $G$ under OWL DL entailment are exactly the matches of $[\![P]\!]$ on $G$ under simple entailment.*

For this thesis we are not interested in rewriting general conjunctive queries with non-distinguished variables. However, in Bischof, Krötzsch et al. [2014a] we show how to obtain such a rewriting.

*Limits* We have seen that schema-agnostic query rewriting works for (almost) all of OWL QL, so it is natural to ask how far this approach can be extended. We outline the intuition of the natural limits of SPARQL 1.1 as a query rewriting language and point out extensions to overcome these limits.

In Section 3.2, we excluded owl:SymmetricProperty from our considerations, because SPARQL 1.1 lacks the necessary expressivity to handle the RDF encoding [Bischof, Krötzsch et al. 2014a]. However, one can write $p$ rdf:type owl:SymmetricProperty as $p$ rdfs:subPropertyOf [owl:inverseOf $p$] and thus indirectly allow symmetric properties. One possible approach to directly deal with owl:SymmetricProperty is nSPARQL, which has been proposed as an extension of SPARQL 1.0 with a form of path expression that can test for the presence of certain side branches in property paths [J. Pérez, Arenas and Gutierrez 2010]. Similar test expressions have been considered in OBDA recently [Bienvenu, Calvanese et al. 2014].

By complexity theoretic arguments we can furthermore exclude most extensions of OWL QL as well as other OWL profiles [Bischof, Krötzsch et al. 2014a]. These complexity-theoretic limitations can only be overcome by using a more complex query language. Many query languages with P-complete data complexity can be found in the Datalog family of languages (see for example Abiteboul, Hull and Vianu [1994]), which are supported by RDF databases like OWLIM and Oracle 11g that include rule engines.

## 3.5 Optimisation and Implementation Remarks

Theorem 3.2 gives a procedure to rewrite CQ-patterns under OWL DL entailment to SPARQL 1.1 graph patterns under simple entailment. With this rewriting procedure, called *BGP Schema-Agnostic Rewriting* (SAR) we can rewrite

SPARQL queries when considering practical issues. In this section we discuss these issues and introduce three orthogonal optimisation approaches to improve query evaluation times.

### *Implementation Remarks*

Blank nodes can not be shared across BGPs in SPARQL [Harris and Seaborne 2013, §4.1.4]. However, since SAR replaces each triple pattern with a UNION pattern this restriction would lead to invalid queries. For example the blank node _:x in the BGP {_:x rdf:type b . _:x c d} would be split across several BGPs. The mapping $\sigma$ from blank nodes to RDF terms (see Definition 2.8) is "lost" outside of the BGP. Therefore blank node mappings for the same blank node _:x from different BGPs can not be joined which contradicts the intended semantics. Since in SPARQL variables can substitute blank nodes (under simple entailment) [Gutierrez, Hurtado and Mendelzon 2004; Franconi, Bruijn and Tessaris 2005] we can retain the semantics by replacing each blank node in a BGP with a unique fresh variable. We call these variables *blank node variables*.

In the preceding theoretical sections we considered SPARQL 1.1 under set semantics for easier handling. Since the official SPARQL semantics [Harris and Seaborne 2013] uses multisets, an implementation of SAR has to cater for duplicate results.[4] To ensure correct answer cardinalities we could embed each BGP in a SPARQL 1.1 SELECT DISTINCT sub-query which projects the temporary variables away, but keeps the blank node variables which should only be projected away in the end. In practice however users rarely rely on such semantically exact behaviour but will either use DISTINCT for the whole query or accept duplicate results. Adding DISTINCT around each BGP could affect query evaluation performance negatively thus we accept duplicates. We note that this might lead to incoherent results considering features on top of basic SPARQL operators such as solution modifiers, grouping and aggregates.

4 This applies to other query rewriting techniques as well, if the target language, like for example SQL, has a multiset semantics.

An implementation has to consider two cases: (1) according to (3.28) a triple pattern $x$ rdf:type $c$ is rewritten as follows (expanding UNIVCLASS):

```
{{x (rdf:type / subClassOf) c} UNION
 {x ?P ?Y.
  ?P (subPropertyOf / (^owl:onProperty | rdfs:domain) / subClassOf) c} UNION
 {?Y ?P x . ?P (subPropertyOf / rdfs:range / subClassOf) c} UNION
 {owl:Thing subClassOf c} UNION
 {owl:topObjectProperty ((SpoEqp | Inv)*/
    (^owl:onProperty | rdfs:domain | rdfs:range)/subClassOf) c}}        (3.31)
```

And (2) for an arbitrary predicate $p$ the triple pattern $x$ $p$ $y$ is rewritten according to (3.30) as follows (expanding SUBINVPROPERTYOF):

```
{{x ?R y . ?R subPropertyOf p} UNION
 {y ?R x . ?R SpoEqp* / Inv / subPropertyOf p}
} UNION univProperty[p]                                                 (3.32)
```

SAR rewriting replaces each triple pattern of a SPARQL query by one of these complex graph patterns which makes the need for optimization apparent. To address this need we introduce three such optimization approaches.: (i) without relaxing the main assumption of SAR (no knowledge of the RDF graph required for rewriting) we can only implement query optimization heuristics using algebraic equivalences, (ii) by using information about the OWL and RDFS constructs used in the ontology we can simplify the path expressions and in some cases also remove even BGPs and (iii) by allowing updates to the TBox of the RDF graph we can materialise path expressions and thereby use a partial ontology saturation approach. In the remainder of this section we describe these three approaches in detail.

### *Algebraic Path Equivalences ($O_E$)*

A SPARQL sequence path could be translated to a BGP [Harris and Seaborne 2013, §18.4]. The path expression $x\ p/q\ y$ is thus translated to $x\ p$ ?v . ?v $q\ y$ (with ?v being a fresh variable). We can generalize this to extract a common path fragment $p$ over a UNION with the same object $y$.

**Lemma 3.5.** *For the IRIs $p, p_1, p_2$, and the RDF terms $x_1, x_2, y$:*

$$\{x_1\ p_1\ /\ p\ y\}\ \text{UNION}\ \{x_2\ p_2\ /\ p\ y\} \equiv \{?\text{v}\ p\ y\}\ .\ \{\{x_1\ p_1\ ?\text{V}\}\ \text{UNION}\ \{x_2\ p_2\ ?\text{V}\}\}$$

*where* ?V *is a fresh variable. A similar equivalence holds for distributivity of a shared prefix path over* UNION.

*Proof.* This follows from the definition of the sequence operator and the distributivity of the join (.) operator over UNION [J. Pérez, Arenas and Gutierrez 2009; Schmidt, Meier and Lausen 2010]. □

**Definition 3.6.** For (3.31) the *algebraic path equivalences optimisation* (OE) applies Lemma 3.5 from left to right to extract SUBCLASSOF. The triple pattern testing the class $c$ being a superclass of owl:Thing is replaced by a BIND pattern. The result of applying OE to (3.31) thus results in the following graph pattern:

```
{?V SUBCLASSOF c}.
{{x rdf:type ?V} UNION
 {x ?P ?Y . ?P SUBPROPERTYOF / (^owl:onProperty | rdfs:domain) ?V} UNION
 {?Y ?P x . ?P (SUBPROPERTYOF / rdfs:range) ?V} UNION
 {BIND(owl:Thing AS ?V)} UNION
 {owl:topObjectProperty (SPOEQP | INV)*/
    (^owl:onProperty | rdfs:domain | rdfs:range) ?V}}                (3.33)
```

For (3.32) OE applies the equivalence (3.5) is used to reuse SUBPROPERTYOF in the arbitrary predicate rewriting (note that in the first BGP we could

replace ?R by ?V):

$$\begin{aligned}&\{?V \text{ subPropertyOf } p\}.\\&\{\{x\ ?V\ y\} \text{ UNION}\\&\ \{y\ ?R\ x\ .\ ?R \text{ SpoEqp}^*\ /\ \text{Inv } ?V\}\\&\} \text{ UNION univProperty}[p]\end{aligned} \tag{3.34}$$

Additionally we will need the following equivalence later for the path materialization optimizations:

$$p^* \equiv (p^+)? \tag{3.35}$$

### *Remove Irrelevant Properties from Paths ($O_I$)*

If a property $p$ never occurs in a graph, then no triple pattern with predicate $p$ will evaluate to a solution mapping. Similarly when $p$ is used in a path pattern, then this path step will never evaluate to a solution mapping. Thus we can safely remove $p$ from the path pattern and reduce the length and, in some cases, an operator of the paths, while still evaluating to the same solution mappings.

In SAR we are interested in removing OWL/RDFS properties not occurring in the TBox, thus achieving a pay-as-you-go behaviour: ontologies using only few RDFS and OWL properties lead to shorter rewritten queries.

**Definition 3.7.** Let $\epsilon$ be the *empty reflexive property* of an arbitrary RDF graph $G$: each IRI in $G$ is connected to itself via $\epsilon$, which is disjoint from any other property. We use $\bot$ as an abbreviation for owl:bottomObjectProperty. For arbitrary path expressions $p_1, p_2$ and we define the following OI *rewriting rules* on property expressions:

$$\bot^* \to \epsilon \tag{3.36}$$
$$\epsilon^* \to \epsilon \tag{3.37}$$
$$\hat{}\bot \to \bot \tag{3.38}$$
$$\hat{}\epsilon \to \epsilon \tag{3.39}$$
$$p_1 \mid \bot \mid p_2 \to p_1 \mid p_2 \tag{3.40}$$
$$p_1 \mid \epsilon \mid \epsilon \mid p_2 \to p_1 \mid \epsilon \mid p_2 \tag{3.41}$$
$$p_1 \,/\, \bot \,/\, p_2 \to \bot \tag{3.42}$$
$$p_1 \,/\, \epsilon \,/\, p_2 \to p_1 \,/\, p_2 \tag{3.43}$$

An OI rule $p' \to p''$ is *applicable* on a path expression $p$ if the path expression $p'$ occurs in $p$; in this case $p'$ is replaced by $p''$. The *exhaustive application* of the OI rules until no rule is applicable anymore, is denoted as $p' \to^* p''$.

Optional patterns absorb $\bot$ in rule (3.40) while sequence patterns absorb $\epsilon$ in rule (3.43). A property removed from a sequence in rule (3.42) is an effective way to remove the whole sequence. Note that the optional pattern is commutative and both optional and sequence patterns are associative. These properties allow us to write the OI rules in such a succinct manner.

We define $I$ as the set of all OWL QL properties and $I' \subseteq I$ as the set of all OWL QL properties to ignore. In our case $I'$ is comprised of those OWL QL properties not occurring in $G$ (we could also selectively ignore properties despite them occurring in the ontology). Then for each $p \in I'$ we replace $p$ by $\perp$. The term rewriting rules are applied until no rule is applicable any more. Since each rule shortens the path by removing either a path operator or an IRI, exhaustive rule application is guaranteed to terminate.

Eventually the path expressions might still contain $\epsilon$ which we can not map directly to SPARQL path expressions. After exhaustively applying the term rewriting rules $\epsilon$ can only occur either (i) in one ore more optional patterns as a result of (3.41) or (ii) as the only remaining property of the whole path expression as a result of (3.36), (3.37) or (3.39). Case (i) is resolved by rewriting each such optional pattern to the SPARQL '?' optional operator:

$$p_1 \mid \epsilon \mid p_2 \rightarrow (p_1 \mid p_2)?$$

Case (ii), subject and object must represent or bind to the same RDF term, is resolved by the following rewriting rules, where $r$ is an RDF term and the expression $gp[x/x']$ replaces each occurrence of an RDF term or variable $x$ in the graph pattern $gp$ by the RDF term or variable $x'$.

$$gp \,.\, \{?\mathsf{V} \; \epsilon \; x\} \rightarrow gp[?\mathsf{V}/r] \tag{3.44}$$
$$gp \,.\, \{x \; \epsilon \; ?\mathsf{V}\} \rightarrow gp[?\mathsf{V}/r] \tag{3.45}$$
$$\{?\mathsf{V} \; \epsilon \; x\} \rightarrow \{\,\mathsf{BIND}(x \; \mathsf{AS} \; ?\mathsf{V})\} \tag{3.46}$$
$$\{x \; \epsilon \; ?\mathsf{V}\} \rightarrow \{\,\mathsf{BIND}(x \; \mathsf{AS} \; ?\mathsf{V})\} \tag{3.47}$$

It is sufficient to address the cases of either object or subject being a variable for the triple expressions in (3.33) and (3.34). The rules (3.44) and (3.45) are correct because in our rewritings ?V is always a temporary variable occurring only in $gp$.

Eventually the following rules define rewritings for special cases of triple patterns and a graph pattern $gp$ to further simpify graph patterns. Rules (3.48) and (3.49) propagate triple patterns with $\perp$ as predicate.

$$gp \,.\, \{s \perp o\} \rightarrow \{s \perp o\} \tag{3.48}$$
$$gp \; \mathsf{UNION} \; \{s \perp p\} \rightarrow gp \tag{3.49}$$

The OI optimization of a path expression $p$ considering the RDF graph $G$, denoted $\mathsf{OI}(p, G)$ is then defined as the exhaustive application of the OI rules on $p'$ where $p'$ is the path expression obtained when replacing all OWL properties not occurring in $G$ by $\perp$.

*Example.* When ignoring rdfs:range by replacing rdfs:range with $\perp$ in the paths of (3.33) the one BGP containing rdfs:range will be removed completely by applying the rules (3.42), (3.48) and (3.49). Another interesting case occurs when ignoring owl:inverseOf, thus replacing owl:inverseOf with $\perp$, then SUBCLASSOF is reduced from originally 36 properties (10 of which are owl:inverseOf) to 12

properties:

$$\begin{aligned}\text{subClassOf} \rightarrow\ & (\text{sCO} \mid \text{eqC} \mid \hat{}\text{eqC} \mid \\ & (\text{owl:intersectionOf} \,/\, \text{rdf:rest}^* \,/\, \text{rdf:first}) \mid \\ & (\text{owl:onProperty} \,/\, (\text{sPO} \mid \text{eqP} \mid \hat{}\text{eqP})^* / (\hat{}\text{owl:onProperty} \mid \text{rdfs:domain})))^*\end{aligned}$$

By further ignoring owl:onProperty would remove the third line (apart from the final ")*") while removing any of owl:intersectionOf or rdf:first would remove the second line of the path expression above.

In the degenerated case of ignoring *all* OWL and RDFS properties OI restores the original triple patterns, i.e., after applying OI to the result of SAR the patterns (3.31) and (3.32) collapse back to $x$ rdf:type $c$ and $x$ $p$ $y$ respectively. The same happens when applying OI to OE because in these cases subClassOf $\rightarrow \epsilon$ and subPropertyOf $\rightarrow \epsilon$ the rules (3.44) and (3.45) are applied.

### *Path Materialization*

Since SAR produces always the same path patterns regardless of the ontology it makes sense to materialize these paths to RDF predicates and adapt the rewriter accordingly by replacing the path expression by the according temporary predicate.

**Definition 3.8.** We define a *materialization rule* $c := p$ for an IRI $c$ and a path expression $p$. The path expression $p$ is *materialized* to an RDF graph $G$ according to the materialization rule $c := p$ by adding a triple $x$ $c$ $y$ to $G$ whenever $x$ $p$ $y$ matches $G$.

*Materializing a Kleene star expression* An issue arises if the outermost operator of a path pattern $p$ is a Kleene star (or the optional '?').

*Example* 3.2. If the WHERE part of a cache insert query of some path $p$ looks like ?s $p^*$ ?p then the cache property would be inserted as a (kind of) identity relation for every resource in the graph. In principle not only "ontology nodes" would be affected by also "ABox nodes". Thus the number of cache triples increases with the data, we have to recomputing the materialization for *every* update, not only TBox updates.

Materializing Kleene star paths in this manner makes separating TBox and ABox useless. We thus apply equivalence (3.35) to replace the Kleene star by the one-or-more operator + in this case, and in the query rewriting the cache property is used in an optional pattern.

Alternatively to the optional operator we could also create a UNION of a triple pattern and a pattern binding the original object term to the variable.

First we define two new classes c:UnivClass and c:UnivProp to mark universal classes and universal properties, respectively. For this we generalize the notion of *materialization rules* to allow an RDF triple on the left hand side

and a macro with a variable on the right hand side. Membership of these two new classes is computed with the corresponding macros as follows:

$$c \text{ rdf:type c:UnivClass} := \textsc{univClass}[c]$$
$$p \text{ rdf:type c:UnivProp} := \textsc{univProperty}[p]$$

For the materialization optimization we define two different variants:: (i) by materializing complete path expressions into the RDF graph and reducing each path pattern in the query to one triple pattern and (ii) by materializing the longest Kleene-star expression into the RDF graph and reducing this complex path expression with a single property. The first variant shortens the path maximally while the second variant materializes only the presumably most costly operator. We expect a better query evaluation times from the first variant and more space efficient materialization, i.e. less materialized triples, from the second variant.

*Materialize complete paths (OMA)* By materializing the complete path expressions we reduce the path pattern maximally, to a single triple pattern. We expect better query evaluation times from this optimization variant than from the following one.

**Definition 3.9.** For the path expression $p$ occurring in the patterns (3.28) and (3.30) we define the following materialization rules:

$$\text{c:dom} := \textsc{subPropertyOf} \,/\, (\hat{}\text{owl:onProperty} \mid \text{rdfs:domain}) \tag{3.50}$$
$$\text{c:rng} := \textsc{subPropertyOf} \,/\, \text{rdfs:range} \tag{3.51}$$
$$\text{c:sc} := (\text{sCO} \mid \text{eqC} \mid \hat{}\text{eqC} \mid \textsc{intListMember} \mid \textsc{someProp} \mid \textsc{somePropInv})^{+} \tag{3.52}$$
$$\text{c:sp} := (\textsc{SpoEqp} \mid (\textsc{Inv} \,/\, \textsc{SpoEqp}^{*} \,/\, \textsc{Inv}))^{+} \tag{3.53}$$
$$\text{c:spi} := \textsc{SpoEqp}^{*} \,/\, \textsc{Inv} \tag{3.54}$$
$$\text{c:spo} := \textsc{SpoEqp}^{+} \tag{3.55}$$

We can materialize *complete paths* created during SAR rewriting and thus reduce the path patterns in the rewriting to triple patterns and to optional path patterns.

In the rewriting the materialization issue discussed above affects the c:sc and the c:sp properties. Thus we apply equivalence (3.35) to the c:sc and the c:sp properties.

The pattern (3.33) is rewritten as follows:

$$\begin{aligned}
&\{?\text{V c:sc? } c\}.\\
&\{\{x \text{ rdf:type ?V}\} \text{ UNION}\\
&\ \{x \text{ ?P ?Y . ?P c:dom ?V}\} \text{ UNION}\\
&\ \{?\text{Y ?P } x \text{ . ?P c:rng ?V}\}\\
&\} \text{ UNION } \{c \text{ rdf:type c:UnivClass}\}\}
\end{aligned} \tag{3.56}$$

The pattern (3.34) is rewritten as follows:

```
{?V c:sp? p}.
{{s ?V o} UNION
 {o ?R s . ?R c:spi ?V}
} UNION {p rdf:type c:UnivProp}}                    (3.57)
```

*Materialize longest Kleene star expression (OMS)* A more economic approach, considering the number of materialized tripes, is to materialize only the *outermost Kleene star path fragments*, since these fragments are usually not covered by indexes in existing SPARQL engines and thus supposedly hard to evaluate.

Since all the paths materialized in this case have the Kleene star as outermost operator, all cache properties are affected by the materialization issue discussed above.

The cache properties cache:starSC and cache:subClassOf from OMA optimization are the same. The same applies to the properties cache:starSP and cache:subPropertyOf from OMA.

The optimizations $O_I$ and $O_{Mx}$ (meaning both OMA and OMS) are orthogonal and can be combined to $O_{IMx}$. The query rewriting for $O_{IMx}$ is the same as for $O_{Mx}$ while the property removal is applied to the materialization queries. The pattern (3.33) is rewritten as follows:

```
{?V c:sc? c}.
{{x rdf:type ?V} UNION
 {x ?P ?Y . ?P (c:sp? / (^owl:onProperty | rdfs:domain)) ?V} UNION
 {?Y ?P x . ?P (c:sp? / rdfs:range) ?V}
} UNION {c rdf:type c:UnivClass}}                   (3.58)
```

The pattern (3.34) is rewritten as follows:

```
{?V c:sp? p}.
{{s ?V o} UNION
 {o ?R s . ?R (c:spo / Inv) ?V}
} UNION {p rdf:type c:UnivProp}}                    (3.59)
```

This the semantics but might be infeasible in practice, because the whole materialization, including all cache properties, has to be recomputed for every TBox update. Depending on the actual changes of the TBox a more refined materialization implementation can deliver better performance. Examples for such improvements are: (i) re-materializing only paths affected by the TBox change and (ii) re-materializing only paths for classes or properties affected by the TBox change. We do not go into details of such an algorithm in this work.

### *Implementation*

We implemented the SAR rewriter as well as the optimization variants introduced in this section. The system takes an input SPARQL query and rewrites

it to a new SPARQL 1.1 query considering the chosen optimization strategy. The resulting query can then be evaluated by the SPARQL 1.1 engine.

The novel main component of a SAR system is the SAR query rewriter. We implemented the SAR rewriting component as a Java library built upon Apache Jena [Jena 2017] for parsing and manipulating SPARQL queries. We provide a command line application and a web user interface.[5]s The implementation can be configured to produce query rewritings for all of the optimizations described in this section.

5 both are linked from http://stefanbischof.at/sar

Additionally we implemented an ontology analyser which uses ASK queries to list which OWL/RDFS properties are not used by an ontology; this is needed for $O_I$.

## 3.6 Evaluation

Comparing a SAR implementation to other systems is hard because SAR makes other assumptions than other systems. Since SAR implements OWL QL we compare our implementation to a OWL QL reasoner. OWL QL reasoners usually also rewrite the queries and might even perform some materialization.

But it is important to note that query rewriting for OWL QL usually depends not only on the query but also the ontology. The ontology is not needed during query evaluation. SAR is independent of the ontology but needs the ontology during runtime. That means other OWL QL rewriters need to rewrite a query again when the ontology changes whereas the resulting SAR queries remain unchanged (except for ontology dependent optimizations).

Also other OWL QL rewriters often handle and produce only conjunctive queries and not SPARQL queries. Usually these queries are not meant to be evaluated on SPARQL engines although that would be possible in principle. The SAR implementation handles and produces specifically SPARQL queries.

As comparison we selected the REQUIEM rewriter [Pérez-Urbina, Motik and Horrocks 2010]. REQUIEM produces a rather small set of union of conjunctive queries (for OWL QL ontologies). REQUIEM provides three modes of operation: *N* for naive, i.e. no optimizations, *F* for full, i.e. query subsumption and dependency graph pruning or *G* for greedy, i.e. full plus greedy unfolding. We adapted the REQUIEM source code to be able to rewrite SPARQL queries and also produce SPARQL queries for the benchmark queries.

We evaluate performance of query rewriting, the path materialization and query evaluation.

*Evaluation System* We executed the query evaluation on a CentOS Linux server with a 2.4GHz CPU of 8 cores and 64 GB of main memory and Java 7.

**Table 3.5**: Number of triples for the different benchmarks

| Benchmark | TBox | OWL QL fragment | ABox |
|---|---|---|---|
| LUBM ($n$) | 306 | 246 | $n \times 10k$ |
| UOBM ($n$) | 977 | 539 | $n \times 25k$ |
| EUGEN ($n$,$m$) | $313 + m \times 80$ | $313 + m \times 80$ | $n \times 10k$ |
| IMDB-MO | 10 523 | 9 802 | 44 930 765 |
| FLY | 115 136 | 75 452 | 32 336 |
| DBpedia+ | 3 130 | 3007 | 29 730 164 |

### *Benchmarks*

Table 3.5 shows an overview of the different benchmark suites used in this evaluation.

LUBM [Guo, Z. Pan and Heflin 2005], although more than a decode old, is still the standard benchmark often used when benchmarking OWL reasoners. LUBM contains an ontology from the university domain, a data generator for creating scaled data sets and 14 queries. LUBM contains a data generator creating around 10k triples for each *university*.

UOBM [Ma et al. 2006] is based on LUBM and contains new ontologies and an improved data generator.

EUGEN [Lutz et al. 2013] is an extension of LUBM aimed at benchmarking OBDA systems. The benchmark contains ontologies with varying numbers of subclasses for the classes "Department", "Course", "Student" and "Professor". Furthermore it contains 6 queries which are longer (in the number of triple patterns) and thus harder to evaluate than LUBM.

IMDB-MO [Rodriguez-Muro, Kontchakov and Zakharyaschev 2013] uses the Movie Ontology (www.movieontology.org) and (the RDF version of) the data from IMDb (www.imdb.com/interfaces).

FLY (http://www.virtualflybrain.org) is an ontology modelling the anatomy of the fly. The ABox is rather small, but the TBox is complex.

DBpedia+ [Zhou et al. 2015] is a benchmark with DBpedia entities and an ontology of the tourism domain. The queries are only atomic queries, one query for each class and one for each property of the ontology.

We first created a OWL QL version of each of the used ontologies. The OWL QL extractor removes each OWL QL violation that OWLAPI [Horridge and Bechhofer 2011] reports. This step is necessary because REQUIEM can also rewrite ontology constructs not allowed OWL QL by using more complex query languages.

#### *SPARQL 1.1 engines*

The triple store implementation predominantly used in Semantic Web technologies research is OpenLink Virtuoso. Virtuoso is a relational database system which provides also a SPARQL engine. RDF is mapped to relational tables and

SPARQL queries are evaluated via SQL queries. Unfortunately the implementation of path patterns, especially transitivity, in Virtuoso (version 07.20.3214) is not compliant to the SPARQL 1.1 specification:

1. Virtuoso can not evaluate transitive path patterns with both subject and object being unbound variables (depending on the query plan) and reports only the following error: "transitive start not given".
2. Virtuoso will not evaluate long queries at all. For many queries generated by our SAR implementation Virtuoso reports the following error: "The SPARQL optimizer has failed to process the query with reasonable quality. The resulting SQL query is abnormally long. Please paraphrase the SPARQL query".

Furthermore Virtuoso forbids blank nodes as subjects in transitive path patterns with the following message: "Subject of transitive triple pattern should be variable or QName, not literal or blank node". The same applies to blank nodes as objects of transitive path patterns. Since the queries resulting from the SAR rewriting do not contain blank nodes as subjects or objects of path patterns we are not affected by this last limitation.

Let us illustrate these limitations using rewritings for the LUBM queries. Limitation 1 makes evaluation of the SAR and OI rewritings for the LUBM queries 1, 3, 5, 10, 11 and 13 impossible to evaluate on Virtuoso. Limitation 2 makes evaluation of the SAR and OI rewritings for the LUBM queries 2, 4, 7, 8, 9 and 12 as well as the OMS rewritings for the LUBM queries 2 and 9 impossible to evaluate on Virtuoso. Since for each query only one error is reported it is not automatically clear if or which queries would suffer from both limitations.

Thus of the SAR and OI rewritings Virtuoso can only evaluate the LUBM queries 6 and 14 which both consist only of a single type triple pattern. Therefore we assume that the rewriting of type triple patterns is not affected by limitation 1 but Virtuoso can not process the arbitrary predicate rewriting.

As an alternative, we use Blazegraph 1.5.3 for the evaluation.[6] Blazegraph is a graph database implementing interfaces for both SPARQL 1.1 and property graphs. We use Blazegraph in triples mode, and thus disable features unnecessary for this evaluation such as named graphs, inferencing and full-text index. Blazegraph includes the optional *Runtime Query Optimizer*[7] (RTO) which is based on ROX [Abdel Kader et al. 2009] to improve join order of high latency queries by sampling different join orderings.

Next we evaluate SAR and the optimization variations, considering query rewriting, path materialization and query answering.

6 http://www.blazegraph.com/

7 https://wiki.blazegraph.com/wiki/index.php/QueryOptimization#Runtime_Query_Optimizer

### *Rewriting*

First we evaluate the query rewriting step. We create 5 variations of all input queries (original and rewritten queries are available at http://stefanbischof.at/sar):

**Table 3.6**: Properties that are ignored by $O_I$ (○: property not used in OWL QL fragment ontology, ●: property not even used in the original ontology); all ontologies contained rdfs:domain, rdfs:range, rdfs:subClassOf, owl:onProperty and owl:inverseOf properties. The lower section of the table lists properties not used in the query rewriting but would only be used for consistency checking queries.

| OWL QL property | DBpedia | EUGEN | FLY | IMDB-MO | LUBM | UOBM |
|---|---|---|---|---|---|---|
| rdfs:domain | | | | | | |
| rdfs:range | | | | | | |
| rdfs:subPropertyOf | ● | | | | | |
| rdfs:subClassOf | | | | | | |
| owl:onProperty | | | | | | |
| owl:inverseOf | | | | | | |
| owl:equivalentClass | | ● | ○ | | ● | ○ |
| owl:equivalentProperty | ● | ● | ● | ● | ● | |
| rdf:first | | ● | ○ | | ○ | ○ |
| rdf:rest | | ● | ○ | | ○ | ○ |
| owl:intersectionOf | ○ | ● | ○ | ● | ○ | ○ |
| owl:disjointWith | ○ | ● | ○ | ○ | ● | ○ |
| owl:complementOf | ○ | ● | ● | ● | ● | ○ |
| owl:members | ● | ● | ● | ○ | ● | ● |
| owl:someValuesFrom | | | | | | |
| owl:propertyDisjointWith | ● | ● | ● | | ● | ● |

*SAR* plain Schema-Agnostic rewriting
R REQUIEM rewriting
*OI* rewriting ignoring non-occurring axioms
*OMS* rewriting with all non-simple star path fragments materialized
*OMA* rewriting with all paths materialized

*Rewriting times* Since the SAR rewriting depends only on the query and not the ontology, the rewriting times are dominated by parsing before and query serialization after rewriting. Thus we measure rewriting times for the internal rewriting step only measured internally by the Java API returning high precision times. The rewriting for each of the queries of LUBM, IMDB-MO and EUGEN took less than 10 ms. While rewriting for OMA and OMS is significantly faster than the rewriting for SAR and OI, all in all rewriting times are negligible compared to SPARQL query parsing times.

*Used RDFS/OWL properties* In the benchmark ontologies we found one to four ontology axiom properties we could ignore for the OI optimization as shown in Table 3.6. When considering only the OWL QL fragment, we could often ignore more axiom triples. Table 3.6 shows the OWL properties ignored by the OI optimization for the different ontologies. When reduced to OWL QL, LUBM reduces to RDFS plus owl:inverseOf. DBPedia+ was the only benchmark which did not include any kind of property equality or hierarchy.

**Table 3.7:** REQUIEM rewriting times in seconds and number of conjunctive clauses in the resulting UCQ for the EUGEN ontology with 10 subclasses for Department, Course, Student and Professor

| | | # of clauses/BGPs | | | Rewriting [ms] | | |
|---|---|---|---|---|---|---|---|
| Query | # subclasses | Full | Greedy | Naive | Full | Greedy | Naive |
| 1 | 0 | 123 | 123 | 150 | 76 | 64 | 50 |
| | 10 | 11 453 | 11 453 | 11 700 | 45 988 | 33 050 | 5185 |
| 2 | 0 | 320 | 320 | 2560 | 2591 | 1889 | 871 |
| | 10 | – | – | 45 760 | – | – | 75 178 |
| 3 | 0 | 1760 | 1760 | 3472 | 4719 | 3533 | 1191 |
| | 10 | 6720 | 6720 | 11 712 | 55 480 | 42 554 | 5317 |
| 4 | 0 | 480 | 480 | 480 | 295 | 267 | 122 |
| | 10 | – | – | – | – | – | – |
| 5 | 0 | 90 | 90 | 6076 | 5809 | 2674 | 1847 |
| | 10 | – | 720 | 73 556 | – | 179 405 | 119 057 |
| 6 | 0 | 102 | 102 | 792 | 272 | 108 | 101 |
| | 10 | 702 | 702 | 10 062 | 15 711 | 1375 | 3709 |

*REQUIEM rewriting* Table 3.7 shows the results of our comparison system, REQUIEM, when rewriting queries of the EUGEN benchmark. Independent of the strategy used (greedy or naive) the rewriting takes much longer, and produces a large number of clauses. The EUGEN benchmark is practically infeasible for REQUIEM because the resulting SPARQL queries are not parsable by some SPARQL engines anymore due to the high number of BGPs.

### *Materialization*

We consider the materialization queries for OMA and OMS as given in the previous section. The materializations were computed by two SPARQL Update queries, one for the OMA case and one for OMS. See Table 3.8 for the number of triples generated by the materialization queries and Table 3.9 for the time in milliseconds needed to compute the materialization on Blazegraph (Blazegraph could compute most of the materializations compared to other engines).

The number of cache triples is similar to the number of ontology triples (compare Table 3.8 with Table 3.5). Although the number of cache triples is significant compared to the ontology, it is negligible compared to the size of the whole graph including the ABox.

A majority of the cache triples result from the SUBCLASSOF macro which is also the longest path and the path which takes the longest to compute.

*OMA versus OMS* The materialization for OMA produces on average approx. 50% more triples than the materialization of OMS. Thus our assumption of the OMS materialization producing a smaller number of triples than the OMA materialization was confirmed by the evaluated ontologies.

**Table 3.8**: Number of triples generated by the OMA and OMS materializations

| | OMA | | | shared | | OMS |
|---|---|---|---|---|---|---|
| Benchmark | c:dom | c:rng | c:spi | c:sc | c:sp | c:spo |
| DBPedia+ | 669 | 584 | 2 | 1 922 | 2 | 0 |
| EUGEN 1 | 69 | 24 | 34 | 453 | 35 | 6 |
| EUGEN 10 | 69 | 24 | 34 | 753 | 35 | 6 |
| EUGEN 100 | 69 | 24 | 34 | 3 453 | 35 | 6 |
| FLY | 8 143 | 7 | 8 | *timeout* | 41 | 35 |
| IMDB-MO | 1 290 | 965 | 32 | 1 797 | 32 | 5 |
| LUBM | 28 | 18 | 6 | 60 | 10 | 6 |
| UOBM | 36 | 24 | 17 | 151 | 24 | 16 |

**Table 3.9**: Materialization times of the OMA and OMS optimization in milliseconds

| | OMA | | | shared | | OMS |
|---|---|---|---|---|---|---|
| Benchmark | c:dom | c:rng | c:spi | c:sc | c:sp | c:spo |
| DBPedia+ | 356 | 74 | 36 | 4 714 | 62 | 23 |
| EUGEN 1 | 178 | 98 | 45 | 5 810 | 181 | 47 |
| EUGEN 10 | 171 | 98 | 53 | 8 611 | 182 | 49 |
| EUGEN 100 | 165 | 75 | 48 | 14 724 | 189 | 54 |
| FLY | 145 | 83 | 32 | *timeout* | 121 | 50 |
| IMDB-MO | 302 | 177 | 34 | 31 904 | 183 | 33 |
| LUBM | 60 | 60 | 44 | 388 | 88 | 43 |
| UOBM | 104 | 99 | 50 | 468 | 130 | 45 |

In case of the FLY ontology, we could not materialize the SUBCLASSOF macro because Blazegraph ran out of heap space even with 12 GB of memory.

### *Query Answering*

Figure 3.1 shows how the answering performance of the different queries in LUBM change with an increasing size of the RDF graph. We evaluated all 14 LUBM queries on 10 different RDF graphs with sizes from 1 to 20 universities where the data of one university amounts to around 10k triples.

The hardest queries seem to be 2, 8 and 9. With the exception of query 1, OMA is never slower than OMS. For some queries, like query 8 OI was performing even better than OMS.

The rather linear behavior of some of the optimization variants can be explained by the LUBM queries delivering always the same results (from university 0) regardless of the number of universities in the current RDF graph because the data generator creates every university disconnected from all other universities query result numbers are not affected by the number of universities.

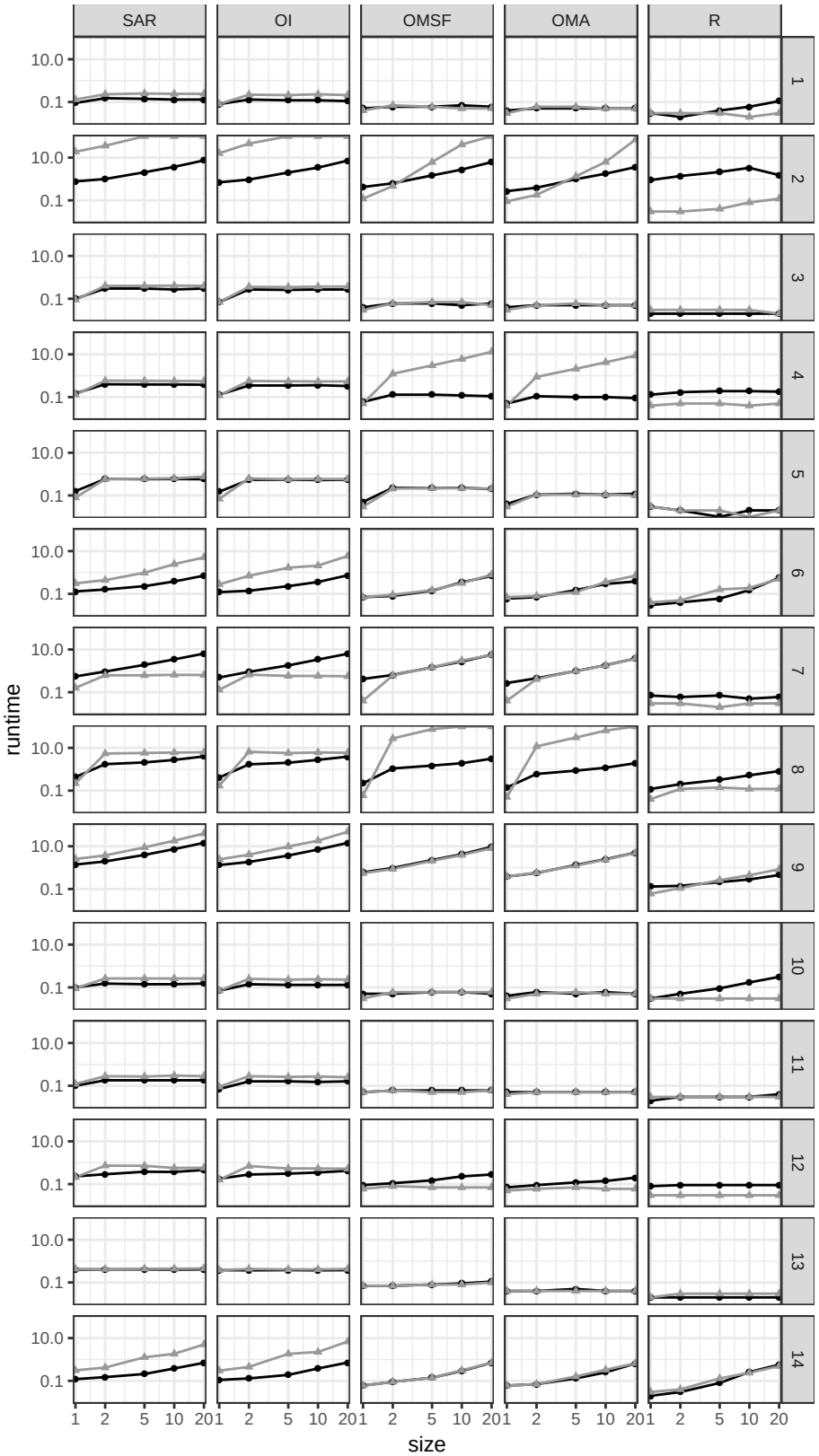



**Figure 3.1:** LUBM query evaluation times in seconds for the 14 different queries (rows) and optimization variants (columns) from 1 to 20 universities by Blazegraph; the black disks: standard settings; grey triangles: runtime optimizer enabled

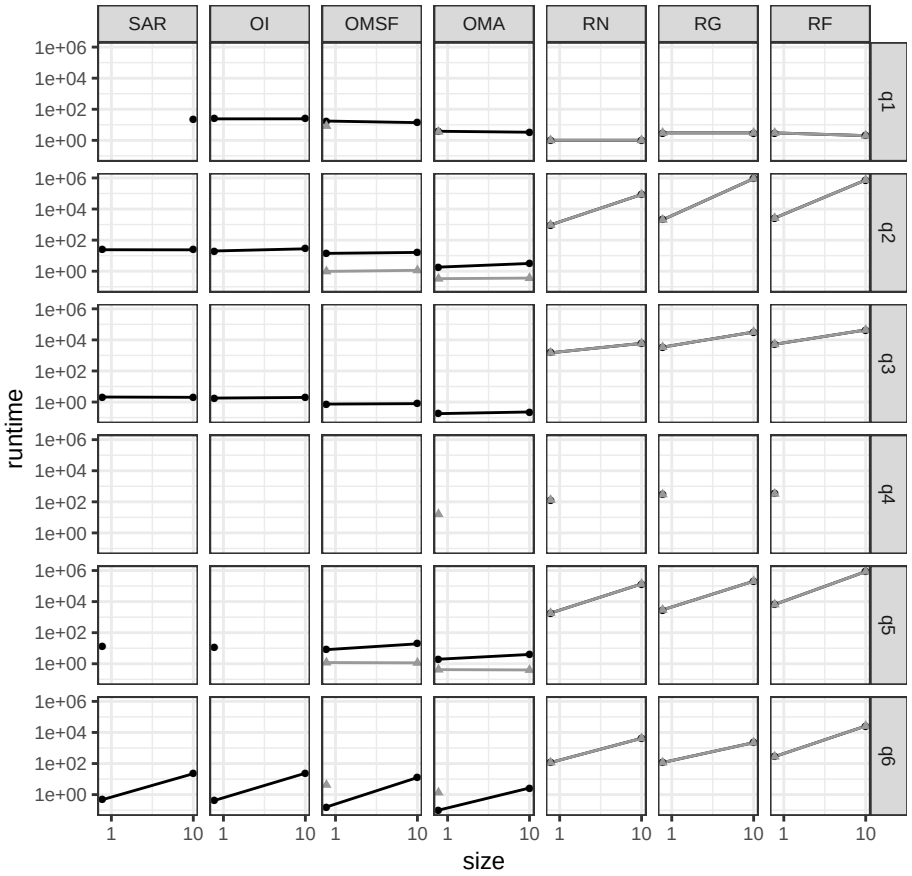



**Figure 3.2:** EUGEN query evaluation times including rewriting times in seconds for the 6 different queries (rows) and optimization variants (columns) for 1 and 10 subclasses for Department, Course, Student and Professor by Blazegraph; the black disks: standard settings; grey triangles: runtime optimizer enabled

### *Discussion*

Query rewriting times are very fast due to the fact that the rewriting algorithm is rather simple and independent of the ontology.

With the exception of the FLY ontology the path materialization was always feasible. The number of materialized triples showed to be comparable to the total number of ontology triples. OMS showed to materialize significantly fewer triples than OMA.

Unoptimized query evaluation times are always slower than evaluating the REQUIEM queries. For the optimized query rewritings in some cases a comparable performance can be achieved. Generally SAR is better suited for simple queries than to more complex ones.

The different optimizations, especially materialization, showed much better performance in many cases. Ignoring unused ontology triples OI is usually better than unoptimized SAR. the materialization optimizations are usually better than OI. OMA, which needs more materialization triples, is usually better than the, triple wise, more economical OMS.

In summary schema-agnostic rewriting offers a novel reasoning technique for the OWL QL profile. The rewriting caters for TBox and ABox being stored in a single RDF graph and essentially implements an OWL QL reasoner in a SPARQL 1.1 query. The queries generated by the schema-agnostic rewriter are more general than OBDA queries in the sense that no new query rewriting is necessary when the ontology is changed. The evaluation showed that unoptimized schema-agnostic rewriting has to pay a price fo this property when comparing it to OBDA approaches which generate queries exactly tailored to the ontology. However, with appropriate optimizations, schema-agnostic rewriting could deliver query evaluation times comparable to OBDA implementations.

# RDF Attribute Equations

4

Section 4.1 gives a detailed motivation for this chapter. Section 4.2 defines our ontology language $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$ which extends the RDFS fragment of DL-Lite by simple equations. Section 4.3 defines SPARQL queries over $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$ and present our query rewriting algorithm along with a discussion of considerations on soundness and completeness. Section 4.4 discusses alternative implementation approaches with DL reasoners and rules. Section 4.5 describes a use case experiment.

## 4.1 Introduction

A wide range of literature has discussed completion of data represented in RDF with implicit information through ontologies, mainly through taxonomic reasoning within a hierarchy of concepts (classes) and roles (properties) using RDFS and OWL. However, a lot of implicit knowledge within real world RDF data does not fall into this category: a large amount of emerging RDF data is composed of numerical attribute-value pairs assigned to resources which likewise contains a lot of implicit information, such as functional dependencies between numerical attributes expressible in the form of simple mathematical equations. These dependencies include unit conversions (e.g. between Fahrenheit and Celsius), or functional dependencies, such as the population density that can be computed from total population and area. Such numerical dependencies between datatype properties are not expressible in standard ontology languages such as RDFS or OWL. Rule based approaches also fail to encode such dependencies in the general case.

*Example* 4.1. Sample RDF data about cities, aggregated from sources such as DBPedia or Eurostat,[1] may contain data of various levels of completeness and using numerical attributes based on different units like

1 cf. http://dbpedia.org/, http://eurostat.linked-statistics.org/

```
:Jakarta  :tempHighC 33 .          :New_York :tempHighF 84 .
:New_York :population 8244910 .    :New_York :area_mile2 468.5 .
:Vienna   :population 1714142 .    :Vienna   :area_km2 414.6 .
:Vienna   :populationDensity 4134 .    ...
```

Users familiar with SPARQL might expect to be able to ask for the population density, or for places with temperatures over 90°F with queries like

```
SELECT ?C ?P WHERE { ?C :populationDensity ?P } or
```

```
SELECT ?C WHERE { ?C :tempHighF ?TempF FILTER(?TempF > 90) }
```

However, implicit answers from mathematical knowledge such as the following equations would not be returned by those queries:

$$tempHighC = (tempHighF - 32) \cdot \frac{5}{9} \tag{4.1}$$

$$populationDensity = \frac{population}{area_{km2}} \tag{4.2}$$

One might ask why such equations cannot be directly added to the terminological knowledge modeled in ontologies? We aim to show that it actually can; further, we present an approach how to extend the inference machinery for SPARQL query answering under ontologies to cater for such equations. Inspired by query rewriting algorithms for query answering over DL-Lite [Calvanese, Giacomo et al. 2007], we show how similar ideas can be deployed to extend a DL-Lite fragment covering the core of RDFS with so-called equation axioms.

We focus on query rewriting techniques rather than e.g. rule-based approaches such as SWRL [Horrocks, Patel-Schneider et al. 2004], where the equations from Example 4.1 could encoded as

$$tempHighC(X, C) \Leftarrow tempHighF(X, F), C = (F - 32) \cdot \frac{5}{9} \tag{4.3}$$

$$popDensity(X, PD) \Leftarrow pop(X, P), area_{km2}(X, A), PD = \frac{P}{A} \tag{4.4}$$

given respective arithmetic built-in support in a SWRL reasoner. However, note that these rules are not sufficient: (i) rule Equation (4.3) is in the "wrong direction" for the query in Example 4.1, that is, we would need different variants of the rule for converting from *tempHighC* to *tempHighF* and vice versa; (ii) the above rules are not *DL safe* (i.e., we do not suffice to bind values only to explicitly named individuals, as we want to compute *new* values) which potentially leads to termination problems in rule-based approaches (and as we will see it actually does in existing systems). Our approach addresses both these points in that (i) equations are added as first class citizens to the ontology language, where variants are considered directly in the semantics, (ii) the presented query rewriting algorithm always terminates and returns finite answers; we also discuss reasonable completeness criteria.

## 4.2 Extending Description Logics by Equations

We herein define a simple, restricted form of arithmetic equations and extend a lightweight fragment of DL-Lite by such equations.

**Definition 4.1.** Let $\{x_1, \ldots, x_n\}$ be a set of variables. A *simple equation* $E$ is an algebraic equation of the form $x_1 = f(x_2, \ldots, x_n)$ such that $f(x_2, \ldots, x_n)$ is an arithmetic expression over numerical constants and variables $x_2, \ldots, x_n$

where $f$ uses the elementary algebraic operators $+, -, \cdot, \div$ and contains each $x_i$ exactly once. vars($E$) is the set of variables $\{x_1, \ldots, x_n\}$ appearing in $E$.

That is, we allow non-polynomials for $f$—since divisions are permitted—but do not allow exponents (different from $\pm 1$) for any variable; such equations can be solved uniquely for each $x_i$ by only applying elementary transformations, assuming that all $x_j$ for $j \neq i$ are known: i.e., for each $x_i$, such that $2 \leq i \leq n$, an equivalent equation $E'$ of the form $x_i = f'(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ is uniquely determined. Note that since each variable occurs only once, the standard procedure for solving single variable equations can be used, we write solve$(x_1 = f(x_2, \ldots, x_n), x_i)$ to denote $E'$.[2]

2 in analogy to notation used by computer algebra systems (such as Mathematica or Maxima)

### *The Description Logic $DL^{E}_{RDFS}$*

When we talk about Description Logics (DL), we consider a fragment of DL-Lite$_{\mathcal{A}}$ [Poggi et al. 2008] with basic concepts, existential quantification, attributes over concrete value domains, role/attribute inclusions, and inverse roles which we extend by simple attribute equations. We we call this fragment $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$, i.e., it is just expressive enough to capture (the DL fragment of) the RDFS semantics [Hayes 2004] extended with equations. In contrast to DL-Lite$_{\mathcal{A}}$, $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$ leaves out role functionality, as well as concept and role negation, and we restrict ourselves to a single value domain for attributes, the set of rational numbers $\mathbb{Q}$.[3]

3 Note that since we only consider this single type of attributes, we also do not introduce value-domain expressions from [Poggi et al. 2008]. Further, instead of $\delta(U)$ by [Poggi et al. 2008] we just write $\exists U$.

**Definition 4.2.** Let $A$ be an atomic concept name, $P$ be an atomic role name, and $U$ be an atomic attribute name. As usual, we assume the sets of atomic concept names, role name, and attribute names to be disjoint. Then DL *concept expressions* are defined as $C ::= A \mid \exists P \mid \exists P^- \mid \exists U$.

In the following, let $\Gamma$ be an infinite set of constant symbols (which, in the context of RDF(S) essentially equates to the set $I$ of IRIs).

**Definition 4.3.** A $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$ knowledge base (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consists of a finite set of terminological axioms $\mathcal{T}$ (TBox) and assertions $\mathcal{A}$ (ABox). For $A, P_i, U_i$ and $C$ denoting atomic concepts, roles, attributes, and concept expressions, resp., $\mathcal{T}$ can contain:

| | |
|---|---|
| $C \sqsubseteq A$ | (concept inclusion axiom) |
| $P_1 \sqsubseteq P_2$ | (role inclusion axiom) |
| $U_1 \sqsubseteq U_2$ | (attribute inclusion axiom) |
| $U_0 = f(U_1, \ldots, U_n)$ | (equation axiom) |

A set of role (attribute, resp.) inclusion axioms is called a *role hierarchy* (*attribute hierarchy*, resp.). For $a, b \in \Gamma$, and $q \in \mathbb{Q}$, an *ABox* is a set of *concept assertions* $C(a)$, *role assertions* $R(a, b)$, and *attribute assertions* $U(a, q)$. Finally, by $\Gamma_{\mathcal{K}}$ (and $\Gamma_A, \Gamma_P, \Gamma_U$, resp.), we denote the (finite) sets of constants from $\Gamma$ appearing in $\mathcal{K}$ (as concepts, roles, and attributes, resp.).

**Table 4.1**: $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$ axioms in RDFS

| | $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$ | RDFS |
|---|---|---|
| 1 | $A_1 \sqsubseteq A_2$ | $\mathsf{A_1}$ rdfs:subClassOf $\mathsf{A_2}$ |
| 2 | $\exists P \sqsubseteq A$ | P rdfs:domain A |
| 3 | $\exists P^- \sqsubseteq A$ | P rdfs:range A |
| 4 | $\exists U \sqsubseteq A$ | U rdfs:domain A |
| 5 | $P_1 \sqsubseteq P_2$ | $\mathsf{P_1}$ rdfs:subPropertyOf $\mathsf{P_2}$ |
| 6 | $U_1 \sqsubseteq U_2$ | $\mathsf{U_1}$ rdfs:subPropertyOf $\mathsf{U_2}$ |
| 7 | $U_0 = f(U_1, \ldots, U_n)$ | $\mathsf{U_0}$ definedByEquation "$\mathsf{f(U_1, \ldots, U_n)}$" |
| 8 | $A(x)$ | x rdf:type A |
| 9 | $R(x, y)$ | x R y |
| 10 | $U(x, q)$ | x U "$q$"^^owl:rational |

Rows 1–6 of Table 4.1 show the obvious correspondence between $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$ syntax and the essential RDFS terminological vocabulary. As for line 7, we can encode equation axioms in RDF by means of a new property definedByEquation and write the respective arithmetic expressions $f(U_1, \ldots, U_n)$ as plain literals (instead of e.g. breaking down the arithmetic expressions into RDF triples). ABox assertions are covered in rows 8–10, where we note that we use datatype literals of the type owl:rational from OWL 2 for rational numbers (which however subsumes datatypes such as xsd:integer, xsd:decimal more commonly used in real world RDF data).

As mentioned before in the context of Definition 4.1, we consider equations that result from just applying elementary transformations as equivalent. In order to define the semantics of equation axioms accordingly, we will make use of the following definition.

**Definition 4.4.** Let $E\colon U_0 = f(U_1, \ldots, U_n)$ be an equation axiom then, for any $U_i$ with $0 \leq i \leq n$ we call the equation axiom solve$(E, U_i)$ the $U_i$*−variant* of $E$.

Note that the DL defined herein encompasses the basic expressivity of RDFS (subproperty, subclassOf, domain, range)[4] and in fact, rather than talking about a restriction of DL-Lite$_{\mathcal{A}}$, we could also talk about an extension of DL-Lite$_{RDFS}$ [Arenas et al. 2012].[5]

4 leaving out subtleties such as e.g. those arising from non-standard use [Bruijn and Heymans 2007] of the RDF vocabulary

5 DL-Lite$_{RDFS}$ actually also allows to write axioms of the form $P_1 \sqsubseteq P_2^-$ which we do not allow since these in fact are beyond the basic expressivity of RDFS.

**Definition 4.5** (Interpretation). An *interpretation* $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ consists of a non-empty set $\Delta^{\mathcal{I}}$ called the object domain, and an interpretation function $\cdot^{\mathcal{I}}$ which maps

- each atomic concept $A$ to a subset of the domain $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$,
- each atomic role $P$ to a binary relation over the domain $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$,
- each attribute $U$ to a binary relation over the domain and the rational numbers $U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \mathbb{Q}$, and
- each element of $\Gamma$ to an element of $\Delta^{\mathcal{I}}$.

For concept descriptions the interpretation function is defined as follows:

- $(\exists R)^{\mathcal{I}} = \left\{x \in \Delta^{\mathcal{I}} \,\middle|\, \exists y.(x,y) \in R^{\mathcal{I}}\right\}$
- $(\exists R^{-})^{\mathcal{I}} = \left\{y \in \Delta^{\mathcal{I}} \,\middle|\, \exists x.(x,y) \in R^{\mathcal{I}}\right\}$
- $(\exists U)^{\mathcal{I}} = \left\{x \in \Delta^{\mathcal{I}} \,\middle|\, \exists q \in \mathbb{Q}.(x,q) \in U^{\mathcal{I}}\right\}$

**Definition 4.6** (Model). An interpretation $\mathcal{I}$ satisfies an axiom of the form

- $C \sqsubseteq A$ if $C^{\mathcal{I}} \subseteq A^{\mathcal{I}}$
- $P_1 \sqsubseteq P_2$ if $P_1^{\mathcal{I}} \subseteq P_2^{\mathcal{I}}$
- $U_1 \sqsubseteq U_2$ if $U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$
- $U_0 = f(U_1, \ldots, U_n)$ if
  $\forall x, y_1, \ldots, y_n(\bigwedge_{i=1}^{n}(x, y_i) \in U_i^{\mathcal{I}}) \wedge \text{defined}(f(U_1/y_1, \ldots, U_n/y_n)$
  $\Rightarrow (x, \text{eval}(f(U_1/y_1, \ldots, U_n/y_n)) \in U_0^{\mathcal{I}}$

where, by $\text{eval}(f(U_1/y_1, \ldots, U_n/y_n))$ we denote the actual value in $\mathbb{Q}$ from evaluating the arithmetic expression $f(U_1, \ldots, U_n)$ after substituting each $U_i$ with $y_i$, and by $\text{defined}(f(U_1/y_1, \ldots, U_n/y_n))$ we denote that this value is actually defined (i.e., does not contain a division by zero). Analogously, $\mathcal{I}$ satisfies an ABox assertion of the form

- $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$
- $P(a,b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in P^{\mathcal{I}}$
- $U(a,q)$ if $(a^{\mathcal{I}}, q) \in U^{\mathcal{I}}$

Finally, an interpretation $\mathcal{I}$ is called a *model* of a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, written $\mathcal{I} \models \mathcal{K}$, if $\mathcal{I}$ satisfies all (role, attribute and concept) inclusion axioms in $\mathcal{T}$, all *variants* of equation axioms in $\mathcal{T}$, and all assertions in $\mathcal{A}$.

Finally, we define conjunctive queries (with assignments) over $\text{DL}^{\text{E}}_{\text{RDFS}}$.

**Definition 4.7**. A *conjunctive query* (CQ) is an expression of the form

$$q(\vec{x}) \leftarrow \exists \vec{y}.\phi(\vec{x}, \vec{y})$$

where $\vec{x}$ is a sequence of variables called *distinguished variables*, $\vec{y}$ is a sequence of variables called *non-distinguished variables*, and $\phi$ is a conjunction of **class**, **role** or **attribute atoms** of the forms $C(x)$, $P(x,y)$, and $U(x,z)$, respectively, and **assignments** of the form $x_0 = f(x_1, \ldots, x_n)$ representing simple equations, where $x, y$ are constant symbols from $\Gamma$ or variables (distinguished or non-distinguished), and $z$ is either a value from $\mathbb{Q}$ or a variable, and the $x_i$ are variables such that for all $i \geq 1$, $x_i$ appears in an atom of the form $U(x, x_i)$ within $\phi$. A set of queries with the same head $q(\vec{x})$ is a *union of conjunctive queries* (UCQ).

For an interpretation $\mathcal{I}$, we denote by $q^{\mathcal{I}}$ the set of tuples $\vec{a}$ of domain elements and elements of $\mathbb{Q}$ which makes $\phi$ true[6] when $\vec{a}$ is assigned to distinguished variables $\vec{x}$ in $q$.

**Definition 4.8**. For a conjunctive query $q$ and a KB $\mathcal{K}$ the *answer to $q$ over $\mathcal{K}$* is the set $\text{ans}(q, \mathcal{K})$ consisting of tuples $\vec{a}$ of constants from $\Gamma_{\mathcal{K}} \cup \mathbb{Q}$ such that $\vec{a}^{\mathcal{M}} \in q^{\mathcal{M}}$ for every model $\mathcal{M}$ of the KB $\mathcal{K}$.

6 We mean true in the sense of first-order logic, where we assume that the interpretation of arithmetic expressions is built-in with the usual semantics for arithmetics over the rational numbers $\mathbb{Q}$, and that equality "=" is false for expressions that yield division by zero on the RHS.

Note that, as opposed to most DL-Lite variants (such as [Calvanese, Giacomo et al. 2007]), $\text{ans}(q, \mathcal{K})$ in our setting is not necessarily finite, as shown by the following example.

*Example* 4.2. Let $\mathcal{K}_1 = (\mathcal{T}_1, \mathcal{A}_1)$ with $\mathcal{A}_1 = u_1(o_1, 1), u_2(o_1, 1), u_3(o_1, 1)$, $\mathcal{T}_1 = \{e\colon u_1 = u_2 + u_3\}$ and $q(x) \leftarrow u_1(o_1, x)$ then $\text{ans}(q, \mathcal{K})$ contains any value from $\mathbb{N}$.

## 4.3 SPARQL over $\text{DL}^{\text{E}}_{\text{RDFS}}$

The semantics of SPARQL is defined as usual based on matching of basic graph patterns (BGPs), more complex patterns are defined as per the usual SPARQL algebra and evaluated on top of basic graph pattern matching, cf. for instance by [Prud'hommeaux and Seaborne 2008; J. Pérez, Arenas and Gutierrez 2009]. In order to remain compatible with the notion of CQs in $\text{DL}^{\text{E}}_{\text{RDFS}}$, we only allow restricted BGPs.[7]

7 We note though, that soundness of our query rewriting approach would not be affected if we allowed arbitrary BGPs.

**Definition 4.9** (Basic Graph Pattern). Let $V$ be an infinite set of variables, $I$ be the set of IRIs, $I_{RDF} = \{\text{rdfs:subClassOf}, \text{rdfs:subPropertyOf}, \text{rdfs:domain}, \text{rdfs:range}, \text{rdf:type}, \text{definedByEquation}\}$, and $I' = I \setminus I_{RDF}$, then *basic graph patterns (BGPs)* are sets of RDF triple patterns $(s, p, o)$ from $((I' \cup V) \times I' \times (I' \cup \mathbb{Q} \cup V)) \cup ((I' \cup V) \times \{\text{rdf} : \text{type}\} \times I')$

More complex graph patterns can be defined recursively on top of basic graph patterns, i.e., if $P_1$ and $P_2$ are graph patterns, $v \in V$, $g \in I \cup V$, $R$ is a filter expression, and *Expr* an arithmetic expression over constants and variables in $V$, then (i) $\{\{P_1\}\{P_2\}\}$ (conjunction), (ii) $\{P_1\}$ UNION $\{P_2\}$ (disjunction), (iii) $P_1$ OPTIONAL $\{P_2\}$ (left-outer join), (iv) $P_1$ FILTER($R$) (filter), and (v) $P_1$ BIND(*Expr* AS $v$) (assignment) are graph patterns; as a syntactic restriction we assume that $v \notin \text{vars}(P_1)$. The evaluation semantics of complex patterns builds up on basic graph pattern matching,[8] which we define in our setting simply in terms of conjunctive query answering over the underlying DL.

8 For simplicity we leave our GRAPH graph patterns or other new features except BIND introduced in SPARQL 1.1.

Following the correspondence of Table 4.1 and the restrictions we have imposed on BGPs, any BGP $P$ can trivially be mapped to a (non-distinguished-variable-free) conjunctive query of the form $q_P\colon q(\text{vars}(P)) \leftarrow \phi(P)$, where $\text{vars}(P)$ is the set of variables occurring in $P$.

*Example* 4.3. Within the SPARQL query

```
SELECT ?X WHERE { { :o1 :u1 ?X } FILTER ( ?X > 1 ) }
```

the BGP `{ :o1 :u1 ?X }` corresponds to the CQ from Example 4.2. FILTERs and other complex patterns are evaluated on top of BGP matching:

**Definition 4.10** (Basic graph pattern matching for $\text{DL}^{\text{E}}_{\text{RDFS}}$). Let $G$ be an RDF representation of a $\text{DL}^{\text{E}}_{\text{RDFS}}$ KB (cf. Table 4.1) $\mathcal{K}$. Then, the solutions of a BGP

**Function** PerfectRef$_E(q, \mathcal{T})$

**Input**: Conjunctive query $q$, TBox $\mathcal{T}$
**Output**: Union of conjunctive queries

$P := \{q\}$
**repeat**
  $P' := P$
  **foreach** $q \in P'$ **do**
    **foreach** *g in q* **do** // expansion
      **foreach** *inclusion axiom I in* $\mathcal{T}$ **do**
        **if** *I is applicable to g* **then**
          $P := P \cup \{q[g/\mathrm{gr}(g, I)]\}$
      **foreach** *equation axiom E in* $\mathcal{T}$ **do**
        **if** $g = U^{\mathrm{adn}(g)}(x, y)$ *is an (adorned) attribute atom and* $\mathrm{vars}(E) \cap \mathrm{adn}(g) = \emptyset$ **then**
          $P := P \cup \{q[g/\mathrm{expand}(g, E)]\}$
**until** $P' = P$
**return** $P$

**Function** grE$(g, I)$

**if** $g = A(x)$ **then**
  **if** $I = A_1 \sqsubseteq A$ **then return** $A_1(x)$
  **else if** $I = \exists P \sqsubseteq A$ **then return** $P(x, \_)$
  **else if** $I = \exists P^- \sqsubseteq A$ **then return** $P(\_, x)$
  **else if** $I = \exists U \sqsubseteq A$ **then return** $U(x, \_)$
**else if** $g = P(x, y)$ **and** $I = P_1 \sqsubseteq P$ **then return** $P_1(x, y)$
**else if** $g = U^{\mathrm{adn}(g)}(x, y)$ **and** $I = U_1 \subseteq U$ **then return** $U_1^{\mathrm{adn}(g)}(x, y)$

$P$ for $G$, denoted (analogously to [J. Pérez, Arenas and Gutierrez 2009]) as $[\![P]\!]_G = \mathrm{ans}(q_P, \mathcal{K})$.

Note that here we slightly abused notation using $\mathrm{ans}(q_P, \mathcal{K})$ synonymous for what would be more precisely "the set of SPARQL variable mappings corresponding to $\mathrm{ans}(q_P, \mathcal{K})$". As for the semantics of more complex patterns, we refer the reader to [Prud'hommeaux and Seaborne 2008; J. Pérez, Arenas and Gutierrez 2009] for details, except for the semantics of BIND which is newly introduced in SPARQL 1.1 [Harris and Seaborne 2013], which we define as:

$$[\![P\ \mathsf{BIND}(Expr\ \mathsf{AS}\ v)]\!]_G = \{\mu \cup \{v \to \mathrm{eval}(\mu(Expr))\} \mid \mu \in [\![P]\!]_G\}$$

Here, by $\mathrm{eval}(\mu(Expr))$ we denote the actual value in $\mathbb{Q}$ from evaluating the arithmetic expression $Expr$ after applying the substitutions from $\mu$.

### *Adapting* PerfectRef *to* $DL^{E}_{RDFS}$

Next, we extend the PerfectRef algorithm [Calvanese, Giacomo et al. 2007] which reformulates a conjunctive query to directly encode needed TBox as-

sertions in the query. The algorithm PerfectRef$_E$ in Algorithm - extends the original PerfectRef by equation axioms and conjunctive queries containing assignments, as defined before, following the idea of query rewriting by "expanding" a conjunctive query (CQ) $Q$ to a union of conjunctive queries (UCQ) $Q_0$ that is translated to a regular SPARQL 1.1 query which is executed over an RDF Store.

PerfectRef$_E$ first expands atoms using inclusion axioms (lines 6–8) as in the original PerfectRef algorithm. Here, an $DL^E_{RDFS}$ inclusion axiom $I$ is *applicable* to a query atom $g$ if the function `gr` (Algorithm -) is defined.[9] The only new thing compared to [Calvanese, Giacomo et al. 2007] in Algorithm - is the "adornment" adn($g$) of attribute atoms which we explain next, when turning to the expansion of equation axioms.

9 With $DL^E_{RDFS}$ we cover only a very weak DL, but we expect that our extension is applicable to more complex DLs such as the one mentioned by [Calvanese, Giacomo et al. 2007], which we leave for future work.

The actually new part of PerfectRef$_E$ that reformulates attribute atoms in terms of equation axioms is in lines 9–11. In order to avoid infinite expansion of equation axioms during the rewriting, the algorithm "adorns" attribute atoms in a conjunctive query by a set of attribute names. That is, given an attribute atom $U(x, z)$ and a set of attribute names $\{U_1, \ldots, U_k\}$ we call $g = U^{U_1,\ldots,U_k}(x, z)$ an *adorned attribute atom* and write $\text{adn}(g) = \{U_1, \ldots, U_k\}$ to denote the set of adornments. For an unadorned $g = U(x, z)$, obviously $\text{adn}(g) = \emptyset$. Accordingly, we call an *adorned conjunctive query* a CQ where adorned attribute atoms are allowed.

The function expand$(g, E)$ returns for $g = U^{\text{adn}(g)}(x, y)$ and $E' : U = f(U_1, \ldots, U_n)$ being the $U$-variant of $E$ the following conjunction:

$$U_1^{\text{adn}(g)\cup\{U\}}(x, y_1) \wedge \ldots \wedge U_n^{\text{adn}(g)\cup\{U\}}(x, y_n) \wedge y = f(y_1, \ldots, y_n) \tag{4.5}$$

where $y_1, \ldots, y_n$ are fresh variables. Here, the condition $\text{vars}(E) \cap \text{adn}(g) = \emptyset$ ensures that $U$ is not "re-used" during expansion to compute its own value recursively. The adornment thus prohibits infinite recursion.

We note that we leave out the *reduction* step of the original PerfectRef algorithm from [Calvanese, Giacomo et al. 2007][Fig.2, step (b)], since it does not lead to any additional applicability of inclusion axioms in the restricted Description Logic $DL^E_{RDFS}$. As we may extend PerfectRef$_E$ to more expressive DLs as part of future work, this step may need to be re-introduced accordingly.

Finally, just as before we have defined how to translate a SPARQL BGP $P$ to a conjunctive query, we translate the result of PerfectRef$_E(q_P, \mathcal{T})$ back to SPARQL by means of a recursive translation function tr(PerfectRef$_E(q_P, \mathcal{T})$). That is, for PerfectRef$_E(q_P, \mathcal{T}) = \{q_1, \ldots q_m\}$ and each $q_i$ being of the form $\bigwedge_{j=0}^{k_i} atom_j$, we define tr as follows:

| | |
|---|---|
| $\mathrm{tr}(\{q_1, \ldots q_m\})$ | $\{\ \mathrm{tr}(q_1)\ \}$ UNION ... UNION $\{\ \mathrm{tr}(q_m)\ \}$ |
| $\mathrm{tr}(\bigwedge_j = 0^{k_i} atom_j)$ | $\mathrm{tr}(atom_1) \ldots \mathrm{tr}(atom_{k_i})$ |
| $\mathrm{tr}(A(x))$ | $\mathrm{tr}(x)$ rdf : type $A$ |
| $\mathrm{tr}(P(x, y))$ | $\mathrm{tr}(x)$ P $\mathrm{tr}(y)$ |
| $\mathrm{tr}(U(x, y))$ | $\mathrm{tr}(x)$ U $\mathrm{tr}(y)$ |
| $\mathrm{tr}(y = f(y_1, \ldots, y_n))$ | BIND(f(tr($y_1$),...,tr($y_n$)) AS tr($y$) ) |
| $\mathrm{tr}(x)$, for $x \in V$ | $?x$ |
| $\mathrm{tr}(x)$, for $x \in \Gamma$ | $x$ |
| $\mathrm{tr}(x)$, for $x \in \mathbb{Q}$ | "$x$"^^owl:rational |

The following proposition follows from the results in [Calvanese, Giacomo et al. 2007], since (a) $\mathrm{PerfectRef}_{\mathrm{E}}$ is a restriction of the original PerfectRef algorithm as long as no equation axioms are allowed, and (b) any $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$ KB is consistent.

**Proposition 4.1.** *Let $q$ be a conjunctive query without assignments and $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a $DL^{E}_{RDFS}$ KB without equation axioms. Then* $\mathrm{PerfectRef}_{\mathrm{E}}$ *is sound and complete, i.e.*

$$\mathrm{ans}(q, \mathcal{K}) = \mathrm{ans}(\mathrm{PerfectRef}_{\mathrm{E}}(q, \mathcal{T}), \langle \emptyset, \mathcal{A} \rangle) \tag{4.6}$$

The following corollary follows similarly.

**Corollary 4.1.** *Let $q$ be a conjunctive query without assignments and without attribute axioms and let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an arbitrary $DL^{E}_{RDFS}$ KB. Then* $\mathrm{PerfectRef}_{\mathrm{E}}$ *is sound and complete.*

As for arbitrary $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$ knowledge bases, let us return to Example 4.2.

*Example* 4.4. Given the knowledge base $\mathcal{K}_1 = \langle \mathcal{T}_1, \mathcal{A}_1 \rangle$ and query $q$ from Example 4.2. The query $\mathrm{PerfectRef}_{\mathrm{E}}(q, \mathcal{T})$ is

$$\{\ q(x) \leftarrow u_1(o_1, x), q(x) \leftarrow u_2^{u_1}(o_1, x_2), u_3^{u_1}(o_1, x_3), x = x_2 + x_3\}$$

which only has the certain answers $x = 1$ and $x = 2$, showing that $\mathrm{PerfectRef}_{\mathrm{E}}$ is incomplete in general. As a variant of $\mathcal{K}_1$, lets consider $\mathcal{K}_2 = \langle \mathcal{T}_1, \mathcal{A}_2 \rangle$ with the modified ABox $\mathcal{A}_2 = \{u_1(o_1, 2), u_2(o_1, 1), u_3(o_1, 1)\}$. In this case, notably $\mathrm{PerfectRef}_{\mathrm{E}}$ delivers complete results for $\mathcal{K}_2$, i.e., $\mathrm{ans}(q, \mathcal{K}_2) = \mathrm{ans}(\mathrm{PerfectRef}_{\mathrm{E}}(q, \mathcal{T}_1), \langle \emptyset, \mathcal{A}_2 \rangle)$ with the single certain answer $x = 2$. Finally, the rewritten version of the SPARQL query in Example 4.3 is

```
SELECT ?X WHERE {
    { { :o1 :u1 ?X } UNION
      { :o1 :u2 ?X2 . :o1 :u3 ?X3 . BIND(?X2+?X3 AS ?X ) } }
    FILTER ( ?X > 1 )  }
```

In order to capture a class of $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$ KBs, where completeness can be retained, we will use the following definition.

**Definition 4.11.** An ABox $\mathcal{A}$ is *data-coherent* with $\mathcal{T}$, if there is no pair of ground atoms $U(x, d'), U(x, d)$ with $d \neq d'$ entailed by $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$

The following result is obvious.

**Lemma 4.1.** *Whenever $\mathcal{A}$ is* data-coherent *with $\mathcal{T}$, any conjunctive query has a finite number of certain answers.*

*Sketch.* Assume that the certain answers to q are infinite. From Corollary 4.1 we can conclude that infiniteness can only stem from distinguished variables that occur as attribute value $y$ in some attribute atom $U(x, y)$ in the query. However, that would in turn mean that there is at least one $x$ with an infinite set of findings for $y$, which contradicts the assumption of data-coherence. □

The following stronger result (which for our particular use case of BGP matching in SPARQL we only consider for non-distinguished-variable-free conjunctive queries) states that data-coherence in fact implies completeness.

**Theorem 4.1.** *If $\mathcal{A}$ is* data-coherent *with $\mathcal{T}$, the for any non-distinguished-variable-free conjunctive query $q$* $\text{PerfectRef}_{\text{E}}$ *is sound and complete.*

*Sketch.* The idea here is that whenever $\mathcal{A}$ is *data-coherent* with $\mathcal{T}$ for any fixed $x$ any certain value $y$ for $U(x, y)$ will be returned by $\text{PerfectRef}_{\text{E}}$: assuming the contrary, following a shortest derivation chain $U(x, y)$ can be either (i) be derived by only atoms $U_i(x, y_i)$ such that any $U_i$ is different from $U$, in which case this chain would have been "expanded" by $\text{PerfectRef}_{\text{E}}$, or (ii) by a derivation chain that involves an instance of $U(x, z)$. Assuming now that $z \neq y$ would violate the assumption of data-coherence, whereas if $z = y$ then $U(x, y)$ was already proven by a shorter derivation chain. □

In what follows, we will define a fragment of $\text{DL}^{\text{E}}_{\text{RDFS}}$ KBs where data-coherence can be checked efficiently. First, we note that a data-coherent ABox alone, such as for instance in $\mathcal{K}_2$ in Example 4.4 above, is in general not a guarantee for data-coherence. To show this, let us consider the following additional example.

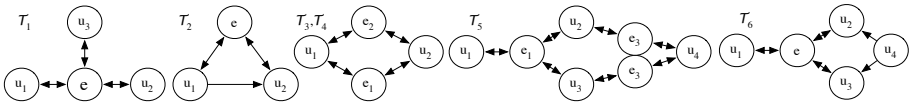
*Example* 4.5. Consider the TBox $\mathcal{T}_2 = \{e\colon u_1 = u_2 + 1, u_2 \sqsubseteq u_1\}$ As easily can be seen, any ABox containing an attribute assertion for either $u_1$ or $u_2$ is data-incoherent with $\mathcal{T}_2$.

The example also shows that considering equation axioms only is not sufficient to decide data-coherence, but we also need to consider attribute inclusion axioms. Following this intuition, we define a dependency graph over $\mathcal{T}$ as follows.

**Definition 4.12.** A TBox dependency graph is $G_{\mathcal{T}} = \langle N, E\rangle$ is constructed from nodes for each attribute and each equation axiom $N = \{e \mid e \text{ is an equation axiom in } \mathcal{T}\} \cup \Gamma_U$. There exist edges $(e, v)$ and $(v, e)$ between every equation $e$ and its variables $v \in \text{vars}(e)$. Furthermore there exists an edge $(u, v)$ for each attribute inclusion axiom $u \sqsubseteq v$. If $G$ contains no (simple) cycle with length greater than 2, then we call $\mathcal{T}$ *attribute-acyclic.*

*Example* 4.6. Given $\mathcal{T}_1, \mathcal{T}_2$ from Examples 4.2 and 4.5, let further $\mathcal{T}_3 = \{e_1\colon u_1 = u_2 + 1, e_2\colon u_2 = u_1 + 1\}$, $\mathcal{T}_4 = \{e_1\colon u_1 = u_2 + 1, e_2\colon u_2 = u_1 - 1\}$, and $\mathcal{T}_5 = \{e_1\colon u_1 =$

$u_2 - u_3, e_2 : u_4 = u_2\, e_3 : u_4 = u_3\}$ $\mathcal{T}_6 = \{e : u_1 = u_2 - u_3, u_4 \sqsubseteq u_2\, u_4 \sqsubseteq u_3\}$ then the resp. dependency graphs are as follows where the graphs for $\mathcal{T}_2$–$\mathcal{T}_5$ are cyclic.




Notably, since $e_2$ is a variant of $e_1$ in $\mathcal{T}_4$, $\mathcal{T}_4$ is actually equivalent to an acyclic TBox (removing either $e_1$ or $e_2$), whereas this is not the case for $\mathcal{T}_3$; more refined notions of acyclicity, which we leave for future work, might capture this difference. Therefore, as shown in Examples 4.2 and 4.4 for $\mathcal{T}_1, \mathcal{T}_2$. Further, let us point out the subtle difference between $\mathcal{T}_5$ and $\mathcal{T}_6$. In $\mathcal{T}_5$, when $e_1$–$e_3$ are viewed as equation system, partially solving this system would result in the new equation $u_1 = 0$, independent of the ABox. Since PerfectRef$_\mathrm{E}$ does not solve any equation systems (but only instantiates equations with values from the ABox), it would not detect this. On the contrary, in $\mathcal{T}_6$, only when a concrete "witness" for $u_4$ is available in the ABox, this constrains the value of $u_1$ to be 0, which could be correctly detected by means of PerfectRef$_\mathrm{E}$: for attribute-acyclic TBoxes, data-coherence indeed (finitely) depends on the ABox and we can define a procedure to check data-coherence (and thus completeness) by means of PerfectRef$_\mathrm{E}$ itself.

**Proposition 4.2.** *Let $\mathcal{T}$ be an attribute-acyclic TBox, and $\Gamma_U = \{u_1, \ldots, u_m\}$. The following SPARQL query $Q^{\mathcal{T}}_{check}$*

```
ASK { { tr(PerfectRef_E(q_P1, T)) FILTER( ?Y1 != ?Z1 ) }
      UNION... UNION
      { tr(PerfectRef_E(q_Pm, T)) FILTER( ?Y1 != ?Z1 ) } }
```

*where $P_i$ = { ?X $u_i$ ?$Y_1$ . ?X $u_i$ ?$Z_2$ } determines data-coherence in the following sense: an ABox $\mathcal{A}$ is data-coherent with $\mathcal{T}$ if $Q$ returns "no".*

The idea here is that since $\mathcal{T}$ is attribute-acyclic, and due to the restriction that variable occurs at most once in simple equations, finite witnesses for data-incoherences can be acyclically derived from the ABox, and thus would be revealed by PerfectRef$_\mathrm{E}$.[10]

10 As a side remark, in fact it suffices to apply the query from Proposition 4.2 to only those attributes $u_i$ appearing in a particular CQ $q$ for determining completeness of the results in this query.

## 4.4 Discussion of Alternative Implementation Approaches

Our approach relies on standard SPARQL 1.1 queries and runs on top of any off-the-shelf SPARQL 1.1 implementation by first extracting the TBox and then rewriting BGPs in each query according to the method described in the previous section. In order to compare this rewriting to alternative approaches, we have looked into DL reasoners as well as rule-based reasoners, namely, Racer, Pellet, and Jena Rules. We discuss the feasibility of using either of these for query answering under $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$ separately.

Racer [Haarslev and Möller 2001] provides no SPARQL interface but uses its own functional query language *new Racer Query Language* (nRQL). The system allows for modeling some forms of equation axioms, cf. examples modeling unit conversions in [Haarslev and Möller 2003], but Racer only uses these for satisfiability testing and not for query answering (which is orthogonal to our approach, as due to the lack of negation there is no inconsistency in $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$).

SWRL [Horrocks and Patel-Schneider 2004; Horrocks, Patel-Schneider et al. 2004] implementations like Pellet [Sirin et al. 2007] allow to handle DL-safe rules [Motik, Ulrike Sattler and Studer 2005], that is, rules where each variable appears in at least one non-DL-Atom. We discussed potential modeling of equation axioms as SWRL rules already in Example 4.1: as mentioned there, rules for each variant of each equation axiom must be added to enable query answering for $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$. Taking this approach, experiments with Pellet showed that queries over certain data-coherent ABoxes were answered correctly (despite—to our reading—rules like (4.3)+(4.4) are not DL-safe in the strict sense), but we still experienced termination problems for e.g. the data and query mentioned in Example 4.1, since strictly speaking, the data for `:Vienna` is not data-coherent (due to rounding errors). Due to the finite nature of our rewriting, our approach always terminates and is thus robust even for such—strictly speaking—incoherent data. Section 4.5 will give more details.

Jena [Jena 2017] provides rule-based inference on top of TDB in a proprietary rule language with built-ins, with SPARQL querying on top. Similar to SWRL, we can encode all variants of equation axioms. Jena allows to execute rules in backward and forward mode, where backward execution does not terminate due to its recursive nature (including empty ABoxes). Forward execution suffers from similar non-termination problems as mentioned above for incoherent data as in Example 4.1, whereas forward execution for data-coherent ABoxes terminates. Jena offers a hybrid rule based reasoning where pure RDFS inferencing is executed in a backward-chaining manner, but still can be combined with forward rules; this approach was incomplete in our experiments, because property inclusion axioms did not "trigger" the forward rules modeling equation axioms correctly.

## 4.5 A Practical Use Case and Experiments

For a prototypical application to compare and compute base indicators of cities—as its needed for studies like Siemens' Green City Index [The Economist Intelligence Unit 2012]—we collected open data about cities from several sources (DBPedia, Eurostat,…) from several years. When aggregating these sources into a joint RDF dataset, different kinds of problems such as incoherences, incomplete data, incomparable units along the lines of the extract in Example 4.1 occurred. Most indicators (such as demography, economy, or cli-

mate data) comprise numeric values, where functional dependencies modeled as equation axioms are exploitable to arrive at more complete data from the sparse raw values.

For an initial experiment to test the feasibility of the query answering approach presented in this chapter, we assembled a dataset containing ABox 254 081 triples for a total of 3 162 city contexts (i.e., when we speak of a "city" sloppily, we actually mean one particular city in a particular year) along with the following (attribute-acyclic) TBox:

$$
\begin{aligned}
&e1: \mathsf{tempHighC} = (:\mathsf{tempHighF} - 32) \cdot 5 \div 9\\
&e2: \mathsf{populationRateMale} = :\mathsf{populationMale} \div :\mathsf{population}\\
&e3: \mathsf{populationRateFemale} = :\mathsf{populationFemale} \div :\mathsf{population}\\
&e4: \mathsf{area\_km2} = :\mathsf{area\_m2} \div 1000000\\
&e5: \mathsf{area\_km2} = :\mathsf{area\_mile2} \div 2.589988110336\\
&e6: \mathsf{populationDensity} = :\mathsf{population} \div :\mathsf{area\_km2}
\end{aligned}
$$

$$
:\mathsf{City} \sqsubseteq :\mathsf{Location} \quad \mathsf{foaf}:\mathsf{name} \sqsubseteq \mathsf{rdfs}:\mathsf{label} \quad \mathsf{dbpedia}:\mathsf{name} \sqsubseteq \mathsf{rdfs}:\mathsf{label}
$$

We use the following queries for our experiments:

Q1. Return the population density of all cities:

```
SELECT ?C ?P
WHERE { ?C rdf:type :City . ?C :populationDensity ?P . }
```

Q2. Select cities with a maximum annual temperature above 90°F.

```
SELECT ?C
WHERE { ?C rdf:type :City . ?C rdfs:label ?L .
        ?C :tempHighF ?P . FILTER(?F > 90) }
```

Q3. Select locations with a label that starts with "W" and a population over 1 million:

```
SELECT ?C
WHERE { ?C rdf:type :Location . ?C rdfs:label ?L .
        ?C :population ?P .
        FILTER(?P > 1000000 && STRSTARTS(?L,"W")) }
```

Q4. Select places with a higher female than male population rate.

```
SELECT ?C
WHERE { ?C :populationRateFemale ?F .
        ?C :populationRateMale ?M . FILTER( ?F > ?M ) }
```

Experimental results are summarized in Table 4.2. For the reasons given in Section 4.4, we compare our approach only to Jena Rules. Experiments were run on the dataset using Jena and ARQ 2.9.2 (without a persistent RDF Store). For Jena Rules, first we encoded the essential RDFS rules plus all variants of equation axioms in a straightforward manner as forward rules, leading to the expected non-termination problems with incoherent data. To avoid this, we created a coherent sample of our dataset (253,114 triples) by removing triples leading to possible incoherences, however still reaching a timeout of

**Table 4.2:** Query response times in seconds

| | Coherent Sample of our Dataset | | | Full Dataset | | |
|---|---|---|---|---|---|---|
| # | Our System | Jena naive | Jena noValue | Our System | Jena naive | Jena noValue |
| Q1 | 6.5 | >600 | 30.7 | 7.3 | – | 30.1 |
| Q2 | 5.8 | >600 | 32.7 | 5.7 | – | 31.3 |
| Q3 | 7.8 | >600 | 32.5 | 8.2 | – | 29.0 |
| Q4 | 6.9 | >600 | 34.3 | 7.9 | – | 32.4 |

10min for all 4 queries. As an alternative approach, we used Jena's negation-as-failure built-in `noValue` which returns sound but incomplete results, in that it fires a rule only if no value exists for a certain attribute (on the inferences so far or in the data); similar to our approach, this returns complete results for data-coherent datasets and always terminates. As an example of encoding the variants of an axiom in Jena Rules, we show the encoding of equation e6 (which is identical to the naive encoding except the `noValue` predicates). Possible divisions by 0, which we do not need to care about in our SPARQL rewriting, since BIND just filters them out as errors, are caught by `notEqual(`*Quotient*`,0)` predicates.

```
[  (?city :area ?ar) (?city :population ?p) notEqual(?ar, 0)
   quotient(?p, ?ar, ?pd) noValue(?city, :populationDensity)
 -> (?city :populationDensity ?d)]
[  (?city :area ?ar) (?city :populationDensity ?pd)
   product(?ar, ?pd, ?p) noValue(?city, :population)
 -> (?city :population ?p)]
[  (?city :populationDensity ?pd) (?city :population ?p)
   notEqual(?pd, 0) quotient(?p, ?pd, ?ar) noValue(?city, :area)
 -> (?city :area ?ar)]
```

Overall, while this experiment was mainly meant as a feasibility study of our query-rewriting approach, the results as shown in Table 4.2 are promising: we clearly outperform the only rule-based approach we could compare to. However, looking further into alternative implementation strategies and optimizations remains on our agenda.

As a final remark, we observed during our experiments that single Web sources tend to be coherent in the values they report for a single city, thus data-incoherences, i.e. ambiguous results in our queries for one city typically stem from the combination of different sources considered for computing values through equations. As a part of future work, we aim to further investigate this, building up on our earlier results for combining inferences in SPARQL with conveying provenance information in the results, cf. [Zimmermann et al. 2012].

## 4.6 Further Related Work and Possible Future Directions

OWL ontologies for measurements and units such as QUDT [Ralph Hodgson 2011], OM [Rijgersberg, Assem and Top 2012] provide means to describe units and—to a certain extent—model conversion between these units, though without the concrete machinery to execute these conversions in terms of arbitrary SPARQL queries. Our approach is orthogonal to these efforts in that (a) it provides not only a modeling tool for unit conversions, but integrates attribute equations as axioms in the ontology language, and (b) allows for a wider range of use cases, beyond conversions between pairs of units only. It would be interesting to investigate whether ontologies like QUDT and OM can be mapped to the framework of $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$ or extensions thereof.

Moreover, in the realm of DL-Lite query rewriting, following the PerfectRef algorithm[Calvanese, Giacomo et al. 2007] which we base on, there have been a number of extensions and alternative query rewriting techniques proposed [Pérez-Urbina, Motik and Horrocks 2010; Rosati and Almatelli 2010; Rosati 2012; Kontchakov, Lutz et al. 2011; Gottlob and Schwentick 2012] which could likewise serve as a basis for extensions by attribute equations. Another obvious direction for further research is the extension to more expressive ontology languages than $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$. Whereas we have deliberately kept expressivity to a minimum in this chapter, apart from further DL-Lite fragments we are particularly also interested in lightweight extensions of RDFS such as OWL LD [Glimm, Adian Hogan et al. 2012] which we aim to consider for future work.

Apart from query answering, this work opens up research in other reasoning tasks such as query containment of SPARQL queries over $\mathrm{DL}^{\mathrm{E}}_{\mathrm{RDFS}}$. While containment and equivalence in SPARQL are a topic of active research [J. Pérez, Arenas and Gutierrez 2009; Schmidt, Meier and Lausen 2010; Chekol et al. 2012] we note that containment could in our setting depends not only on the BGPs, but also on FILTERs. E.g., intuitively query QQ4. in our setting would be equivalent (assuming `:population` $> 0$) to

```
SELECT ?C WHERE { ?C :populationFemale ?F .
                  ?C :populationMale ?M . FILTER( ?F > ?M ) }
```

While we leave closer investigation for future work, we note another possible connection to related work [Clément de Saint-Marcq et al. 2012] on efficient query answering under FILTER expression also based in constraint-based techniques.

Lastly, we would like to point out that our approach could be viewed as rather related to Constraint-handling-rules [Frühwirth 2006] than to mainstream semantic Web rules approaches such as SWRL, etc.; we aim to further look into this.

Part III

# Application

# 5 Use-Case: Open City Data

Section 5.2 gives an overview of the City Data Pipeline architecture, including a description of data sources and a description of how the resulting data set is made available in a re-usable and sustainable manner via a web interface, a Linked Data interface and a public SPARQL endpoint. Section 5.3 describes the data gathering as well as the main challenges in this context.

## 5.1 Introduction

The public sector collects large amounts of statistical data. For example, the United Nations Statistics Division[1] provides regularly updated statistics about the economy, demographics and social indicators, environment and energy, and gender on a global level. The statistical office of the European Commission, Eurostat[2], provides statistical data mainly about EU member countries. Some of the data in Eurostat has been aggregated from the statistical offices of the member countries of the EU. Even several larger cities provide data in on their own open data portals, e.g., Amsterdam, Berlin, London, or Vienna[3]. Increasingly, such data can be downloaded free of charge and used under liberal licences.

1 http://unstats.un.org/unsd/

2 http://ec.europa.eu/eurostat/

3 http://data.amsterdam.nl/, http://daten.berlin.de/, http://data.london.gov.uk/, and http://data.wien.gv.at/

Such open data can benefit public administrations, citizens and enterprises. The public administration can use the data to support decision-making and back policy decisions in a transparent manner. Citizens can be better informed about government decisions, as publicly available data can help to raise awareness and underpin public discussions. Finally, companies could develop new business models and offer tailored solutions to their customers based on such open data. As an example for making use of such data, consider Siemens' Green City Index (GCI) [The Economist Intelligence Unit 2012], which assesses and compares the environmental performance of cities. In order to compute the KPIs used to rank cities' sustainability, the GCI used qualitative and also quantitative indicators about city performance, such as for instance $CO_2$ emissions or energy consumption per capita. Although many of these quantitative indicators are openly available, the datasets had to be collected, integrated, and checked for integrity violations because of the following reasons: (i) heterogeneity: ambiguous data published by different Open Data sources in different formats, (ii) missing data, that needed to be

added or estimated by manual research, and, last but not least, (iii) outdated data: soon after the GCI had been published in 2012, its results were likely already obsolete.

Inspired by this concrete use case of the GGI, the goal of the present work is on collecting, integrating, and enriching quantitative indicator data about cities including basic statistical data about demographics, socio-economic factors, or environmental data, in a more automated and integrated fashion to alleviate these problems.

Even though there are many relevant data sources which publish such quantitative indicators as open data, it is still cumbersome to use data from multiple sources in combination and to keep this data up-to-date. The system we present in this paper, the Open City Data Pipeline, thus contributes by addressing all of the three above challenges (i)–(iii) in a holistic manner:

*(i) Heterogeneity:* All this data is published in different formats such as CSV, JSON, XML, proprietary formats such as XLS, just as plain HTML tables, or even worse within PDF files—and so far to a much lesser degree only as RDF or even as Linked Data [Neumaier, Umbrich and Polleres 2016]. Also, the specifications of the individual data fields—(a) how indicators are defined and (b) how they have been collected—are often implicit in textual descriptions only and have to be processed manually for understanding whether seemingly identical indicators published by different sources are indeed comparable.

**Our contribution**: we present a systematic approach to integrate statistical data about cities from different sources as *Statistical Linked Data* [Kämpgen, O'Riain and Harth 2012] as a standardised format to publish both the data and the metadata. We build a small ontology of core city indicators, around which we can grow a statistical Linked Data cube: we use standard Linked Data vocabularies such as the RDF Data Cube (QB) [Cyganiak, Reynolds and Tennison 2014] vocabulary to represent data of statistical data cubes, as well as the PROV [Lebo, McGuinness and Sahoo 2013] vocabulary to track the original sources of the data, and we create an extensible pipeline of crawlers and Linked Data wrappers collect this data from the sources.

*(ii) Missing values:* Data sources like Eurostat Urban Audit cover many cities and indicators. However, for reasons such as cities providing values on a voluntary basis, the published datasets show a large ratio of missing values. The impact of missing values is aggravated when combining different data sets, due to either covering different cities or using different, non-overlapping sets of indicators.

**Our contribution**: our assumption – inspired also by works that suspect the existence of quantitative models behind the working, growth, and scaling of cities [Bettencourt et al. 2007] – is that most indicators in such a scoped domain as cities have their own structure and dependencies, from

which we can build statistical prediction models and ontological background knowledge in the form of equations.[4] We have developed and combined integrated methods to compute missing values on the one hand using statistical inference, such as different standard regression methods, and on the other hand rule-based inference based on background knowledge in the form of equations that express knowledge about how certain numerical indicators can be computed from others.[5] While this new method is inspired by our own prior work on using statistical regression methods [Bischof, Martin et al. 2015a] and equational knowledge in isolation [Bischof and Polleres 2013], as we can demonstrate in our evaluation, the combination of both methods outperforms either method used alone. We re-publish the imputed/estimated values, adding respective PROV records, and including error estimates, as Linked Data.

*(iii) Updates and changes:* Studies like the GCI are typically outdated soon after publication since reusing or analysing the evolution of their underlying data is difficult. To improve this situation, we need regularly updated, integrated data stores which provide a consolidated, up-to-date view on data from relevant sources.
**Our contribution**: the extensible single data source wrappers (based on the work around rule-based linked data wrappers by Stadtmüller et al. [Stadtmüller et al. 2013]) in our pipeline architecture, are crawling each integrated source regularly (once a day) for new data, thus keeping the information as up-to-date as possible, while at the same time re-triggering the missing value enrichment methods and thereby continuously improving the quality of our estimations for missing data: indeed we can show in our evaluations that the more data we collect in our pipeline over time, the better our prediction models for missing values get.

4 We sometimes refer to "predicting" instead of "imputing" values when we mean finding suitable approximation models to estimate indicator values for cities and temporal contexts where they are not (yet) available. These predictions may (not) be confirmed, if additional data becomes available.

5 Such equational knowledge could be also understood as "mapping" between indicators, which together with manually crafted equality mappings between indicators published by different data sources can be exploited for enrichment, e.g. if one source publishes the population and area of a city, but not the population density, then this missing value, available for other cities directly from other sources, could be computed by an equation.

In summary, our work's contribution is twofold, both in terms of building a practically deployed, concrete system to integrate and enrich statistical data about cities in a uniform, coherent and re-usable manner, and contributing novel methods to enrich and assess the quality of Statistical Linked Data:

1. as for the former, we present the Open City Data Pipeline which is based on *a generic, extensible architecture* and how we integrate data from multiple data sources that publish numerical data about cities in a modular and extensible way, which we re-publish as Statistical linked data.
2. as for the latter, we describe *the combination of statistical regression methods with equational background knowledge*, which we call *QB equations*, in order to impute and estimate missing values.

We also *evaluate* our approach in terms of measuring the errors (by evaluating the estimated root mean square error rate (RMSE) per indicator) of such estimates, demonstrating that firstly, the combination of statistical inference with equations indeed pays off, and secondly, the regular update and

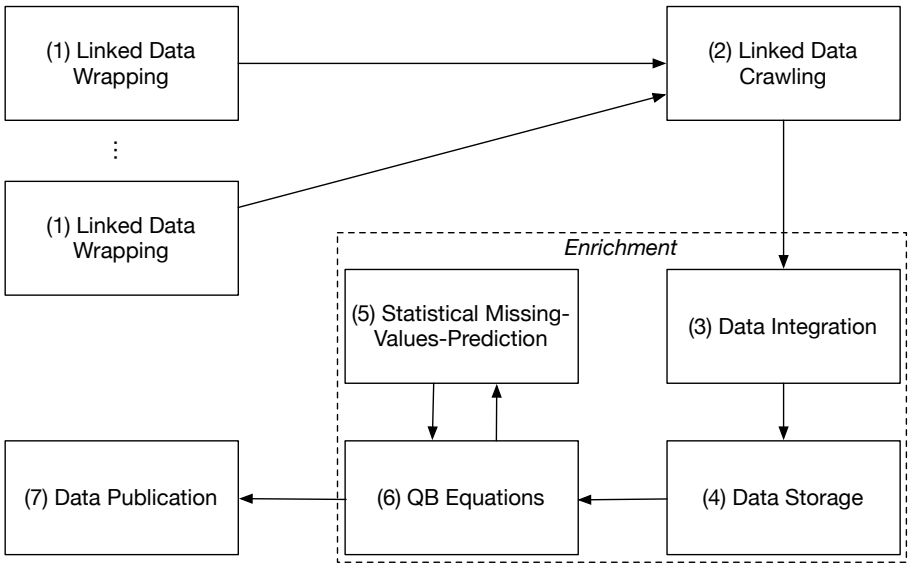



**Figure 5.1**: Open City Data Pipeline workflow

collection of additional data through our pipeline contributes to improve our estimations for missing values in terms of accuracy. Note that the method of enrichment by QB equations can not only be used for imputing missing values, but also be used to assess the quality of ambiguous values from different data sources: by "rating" different observed values for the same indicator and city from different sources against their distance to our estimation, we have means to return confidence in different sources in such an integrated system.

## 5.2 Overview and System Architecture

The workflow of the Open City Data Pipeline (OCDP) is illustrated in Figure 5.1 and consists of several steps:

1. Data is provided as Statistical Linked Data via wrappers which have to be created once per source in the Wrapping step.
2. A crawler collects data regularly (currently, weekly) from different sources in the Crawling step through the wrappers.
3. In the Data Integration step the data is integrated into the global cube, where data is enriched by links and heterogeneities resolved.
4. In the Data Storage step, the data is loaded into a SPARQL endpoint.
5. One further enrichment step exploits equational background knowledge in the form of QB equations.
6. Another further enrichment step applies statistical methods for missing values prediction.
7. Finally, in the Data publication step, the resulting enriched Linked data is made accessible.

In order to realise these steps, the architecture of the OCDP system implements several components. Figure 5.2 gives a high level overview of the architecture with a triple store being the central part. The data quality improvement workflow uses various methods to improve data quality and enrich the data.

We start with surveying data sources that serve as input to the pipeline in Section 5.2. We introduce the different components, their inputs, outputs, and interfaces in Section 5.2 and explain how we make the resulting data available in Section 5.2.

### *Data Sources*

Many interesting statistical data sources are nowadays available. Many indicators in these data sources are provided on a country level and only a subset of indicators are available on the city level. We have identified the following potential providers of statistical data concerning cities:

- DBpedia[6];
- Wikidata[7];
- Eurostat with Urban Audit;
- United Nations Statistics Division (UNSD) statistics;
- U.S. Census Bureau statistics;
- Carbon Disclosure Project[8];
- individual city data portals.

6 http://dbpedia.org/

7 http://wikidata.org/

8 https://www.cdp.net

In particular, we use statistical data from the United Nations and from Eurostat, which are integrated and enriched by the OCDP. The data sources contain data ranging from the years 1990 to 2016, but most of the data concerns the years after 2000. Further, not every indicator is covered over all years, where the highest coverage of indicators is between 2004 and 2015 (see Tables 5.1 and 5.2). Most European cities are contained in the Eurostat datasets. The UNSD contains the capital cities and cities with a population over 100 000, all listed in the United Nations Demographic Yearbook.[9]

9 http://unstats.un.org/unsd/demographic/products/dyb/dyb2012.htm

The previous OCDP of ISWC 2015 [Bischof, Martin et al. 2015a] contains data from 1990 to 2013 with 638,934 values from the Eurostat data source and 69,772 values from the U.N. data source. Due to some reorganisation in the Eurostat and U.N. datasets, Eurostat contains now 506,854 values and the U.N. provides 40,532 values. Regarding indicators, we now have 209 instead of 215 Eurostat and 64 instead of 154 U.N. indicators. The reason for the drop in indicators is due to the fact that the U.N publishes fewer datasets. The same effect can be seen for the cities, where we have 966 instead of 943 Eurostat and 3,381 instead of 4,319 U.N cities. Due to the smaller size of the datasets (see Tables 5.1 and 5.2), we now have an improved missing values ratio of 81.7% (before 86.3%) for Eurostat, resp. 94.4% (before 99.5%) for the U.N. dataset.

We now describe each of the data sources in detail.

Linked Data Sources
Linked Data Wrappers
Crawler
Enrichment
Data Integration
Statistical Missing-
Values-Prediction
Triple Store

*Eurostat* Eurostat[10] offers various datasets concerning E.U. statistics. The data collection is conducted by the national statistical institutes and Eurostat itself. Of particular interest is the Urban Audit (UA) collection, which started as an initiative to assess the quality of life in European cities. UA aims to provide an extensive look at the cities under investigation, since it is a policy tool to the European Commission: "The projects' ultimate goal is to contribute towards the improvement of the quality of urban life" [Office for Official Publications of the European Communities 2004]. Currently, data collection takes place every three years (last survey in 2015) and is published via Eurostat Urban Audit. All data is provided on a voluntary basis which leads to varying data availability and missing values in the collected datasets. At the city level, Urban Audit contains over 200 indicators divided into the categories Demography, Social Aspects, Economic Aspects, and Civic Involvement. Currently, we extract the datasets that include the following topics:

10 http://ec.europa.eu/eurostat

- Population by structure, age groups, sex, citizenship, and country of birth
- Fertility and mortality
- Living conditions and education
- Culture and tourism
- Labour market, economy, and finance
- Transport, environment, and crime.

*United Nations Statistics Division (UNSD)* The UNSD offers data on a wide range of topics such as education, environment, health, technology and tourism. The focus of the UNSD is usually on the country level, but there are some datasets on cities available as well. Our main source is the UNSD Demographic and Social Statistics, which is based on the data collected annually (since 1948) by questionnaires to national statistical offices.[11] Currently we use the datasets on the city level that include the following topics:

11 http://unstats.un.org/unsd/demographic/

- Population by age distribution, sex, and housing
- Households by different criteria (e.g., type of housing)
- Occupants of housing units / dwellings by broad types (e.g., size, lighting, etc.)
- Occupied housing units by different criteria (e.g., walls, waste, etc.)

The full UNSD Demographic and Social Statistics data has over 650 indicators, wherein we kept a set of 64 course-grained indicators and dropped the most fine-grained indicator level. For example, we keep *housing units total* but drop *housing units 1 room*. We prefer more coarse-grained indicators to avoid large groups of similar indicators which are highly correlated.

### *Pipeline Components*

We now give an overview of each of the components of the OCDP system.

**Table 5.1**: Values of the Eurostat Dataset

| Year(s) | Cities | Indicators | Available | Missing | Missing Ratio (%) |
|---|---|---|---|---|---|
| *1990* | 131 | 88 | 1 799 | 9 641 | 84.27 |
| *2000* | 433 | 163 | 6 420 | 63 996 | 90.88 |
| *2005* | 598 | 168 | 20 460 | 79 836 | 79.60 |
| *2010* | 869 | 193 | 56 528 | 110 996 | 66.26 |
| *2015* | 310 | 69 | 2 030 | 19 291 | 90.48 |
| *2004–2016* | 879 | 207 | 437 565 | 1 331 250 | 75.26 |
| *All (1990–2016)* | 966 | 209 | 506 854 | 2 257 171 | 81.66 |

**Table 5.2**: Values of the United Nations Dataset

| Year(s) | Cities | Indicators | Available | Missing | Missing Ratio (%) |
|---|---|---|---|---|---|
| *1990* | 5 | 3 | 8 | 7 | 46.67 |
| *2000* | 1 078 | 61 | 3 861 | 61 836 | 94.12 |
| *2005* | 777 | 61 | 2 110 | 45 226 | 95.54 |
| *2010* | 1 525 | 64 | 5 866 | 91 670 | 93.99 |
| *2015* | 216 | 3 | 568 | 77 | 11.94 |
| *2004–2016* | 2 095 | 64 | 28 849 | 511 759 | 94.66 |
| *All (1990–2016)* | 3 381 | 64 | 40 532 | 685 548 | 94.42 |

*Statistical Linked Data Wrappers* Currently, none of the mentioned data sources publishes statistical data as Statistical Linked Data upfront. Thus, we use a set of wrappers which publish the data from these sources according to the principles listed in Chapter 2. Section 5.3 below explains these wrappers in more detail.

*Linked Data Crawler* A Linked Data crawler starts with a seed list of URIs and crawls relevant connected Linked Data. The resulting RDF data is collected in one big RDF file and eventually loaded into the triple store. Section 5.3 explains the linked data crawler in more detail.

*Triple Store* We use a standard Virtuoso 7 triple store as a central component to store data at different processing stages. For data loading we use the Virtuoso SQL console which allows faster data loading. For all other data access we rely on Virtuoso's SPARQL 1.1 interface which allows not only to query for data but with SPARQL Update also to insert new triples.

*Enrichment Component: Data quality improvement workflow* In an iterative approach we improve data quality of the crawled raw data. This component covers steps (4)-(6) in the workflow shown in Figure 5.1: in this configurable workflow we use the several different sub-components corresponding to these steps consecutively. Each workflow component first reads input data (=observations) from the triple store via SPARQL queries, processes the data

accordingly and inserts new triples into the triple store either via SPARQL Insert queries or the Virtuoso bulk loader facility (the first option is more flexible – it allows the execution of the workflow on a different machine – the second usually allows faster data loading).

The workflow currently uses three different subcomponents corresponding to steps (4), (5) and (6), respectively:

- The *Data Integration* sub-component, corresponding to step (3), performs some linking and data integration steps and materialises the global cube in a separate named graph. This linking and materialisation effectively resolves different types of heterogeneity found in the raw data: (i) different URIs for members, (ii) different URIs for dimensions, (iii) different DSDs (although the DSDs must be compatible to some extent for the integration to make sense). Eventually the global cube provides a unified view over many datasets from several sources. This component is implemented with SPARQL Update queries and supplied background knowledge for the integration. The exact process of linking statistical data and the materialisation will be described in more detail in Section 5.3 and Section 5.3 below.
- The *Statistical Missing Values Prediction* sub-component for missing value prediction, corresponding to step (5), extracts the whole global cube generated by the materialisation as one big data matrix, which is then used for applying different standard statistical regression methods to train models for missing value prediction. This component is implemented as a set of R scripts which extract the data with SPARQL queries. We then train and evaluate the models for each of the indicators. If the selected model delivers predictions in a satisfactory quality we apply the model and get estimates for the indicators. Finally the component exports the statistical data together with error estimates to one RDF file which is then loaded into the triple store with the Virtuoso bulk load feature and added to the global cube. Section 7.1 below explains the details of this components in more detail.
- The *QB Equations* sub-component, corresponding to step (6), uses equations from different sources to infer even more data. To this end, we introduce QB equations. These QB equations provide an RDF representation format for equational knowledge and a semantics as well as a forward chaining implementation to infer new values. QB equations are implemented in a naive rule engine which directly executes SPARQL INSERT queries on the triple store. Chapter 6 introduces the concept of QB equations with syntax, semantics and implementation.

Lastly, in Section 6.3, we also explain the interplay between the Data Enrichment sub-components in more detail, that is—roughly—after cleansing and linking in component (4) we first run the QB equations component (6) once, to compute any values by equations that can be derived from the raw factual data alone, then approximate the remaining missing values by the statistical missing values prediction component (5), after which finally we run the QB

equations component (6) again to improve predictions from (5) iteratively. As we will see in a detailed evaluation inSection 7.2, this iterative combination indeed performs better than using either (5) or (6) alone.

*Data Publication*

Eventually after the data is crawled and loaded into the triple store, improved and enriched by our workflow, the resulting global cube is available for consumption.

We provide a SPARQL endpoint[12] based on Virtuoso, where the global cube is stored in a named graph.[13] The prefix names used in the examples above are already set in Virtuoso, thus no prefix declarations are necessary for SPARQL queries.

We also provide a simple user interface[14] to query values for a selected indicator and city in the global cube. Queries are directly executed on the triple store during loading of the website using a JavaScript library called Spark; thus one can have a look at the SPARQL queries in the source code. We show all predicted values for transparency reasons. We simply order by the error value, i.e., the most trustworthy value per year is always shown first.

12 http://citydata.wu.ac.at/ocdp/sparql

13 http://citydata.wu.ac.at/qb-materialised-global-cube

14 http://kalmar32.fzi.de/indicator-city-query.php

## 5.3 Data Conversion, Linking, and Integration

We now explain our approach for data conversion, linking, and integration. The approach is modular and extensible in the sense that every new data source can be prepared for consideration separately and independently from other sources. The data integration pipeline can be re-run at any time and thus allow for up-to-date data.

The approach consists of the following components:

- Linked Data wrappers that publish numerical data from various data sources as Statistical Linked Data (Section 5.3);
- the definition of a unified view over all relevant Statistical Linked Data (Section 5.3);
- semi-automatically generated links between Statistical Linked Data from different sources (Section 5.3);
- a rule-based Linked Data crawler to collect the relevant data and creates the unified view (Section 5.3).

*Wrappers*

We use Linked Data as interface to access and represent relevant data sources (e.g., Eurostat or UNSD), which are originally published in tabular form. The

uniform Linked Data interface hides the specialities and structure of the original data source. When the wrapper receives an HTTP request for a particular dataset, it retrieves the data on-the-fly from the original source, transforms the tabular representation to RDF, using the RDF Data Cube vocabulary, and returns the RDF representation of the original tabular data.

The wrappers provide a table of contents with links to all available datasets (as a collection of `qb:DataSet` triples), including the data structure definition of the datasets (as `qb:DataStructureDefinition`). The individual data points are modelled as observations (as `qb:Observation`). The data structure definition includes the available dimensions (as `qb:dimension`) and concept schemes (as `skos:ConceptScheme`). We require a list of dataset and data structure definitions to be able to crawl the data.

Each wrapper coins URIs for identifying the relevant resources, for example, indicators or locations. We use URIs as unique identifiers for datasets, dimensions, and dimension values from different data sources.

The data sources identify indicators differently. For example, UNSD provides population numbers in dataset "240", while Eurostat provides population numbers in dataset "urb_cpop1". We use the City Data Ontology to unify the various indicator identifiers. Similarly, locations have varying identifiers and sometimes varying names in the different data sources. For a relatively clear-cut example consider the city of Vienna: UNSD uses city code "001170" and label "WIEN", whereas Eurostat uses code "AT001C1" and label "Wien". The wrappers generate a Uniform Resource Identifier (URI) for every city out of the unique identifiers in the original tabular data.

We use the following wrappers that provide access to the underlying data source via a Linked Data interface:

*Eurostat Wrapper* The Eurostat wrapper[15] makes the Eurostat datasets, originally available in tabular form at the Eurostat website, available as Linked Data. Eurostat provides several dictionary files in SDMX format; these files are used to construct a list of dimension values in the data structure definition and to generate URIs for relevant entities (such as cities). All files are accessed from the original Eurostat server once the wrapper receives a HTTP request on the particular URI, ensuring that the provided RDF data is up-to-date. Population data in the Eurostat wrapper[16] uses http://estatwrap.ontologycentral.com/dic/indic_ur#DE1001V to identify "Population on the 1st of January, total". The indicator URI is mapped to indicator URIs from the City Data Ontology in a subsequent step.

15 http://estatwrap.ontologycentral.com/

16 http://estatwrap.ontologycentral.com/id/urb_cpop1

*UNSD Wrapper* The UNSD wrapper[17] makes the UNSD datasets, originally available in tabular form at the UNSD website, available as Linked Data. The UNSD wrapper provides a simple data structure definition describing the available dimensions and measure. In total, we cover 14 datasets ranging from population to housing data. Most indicators, e.g., population of the "240" data-

17 http://citydata.wu.ac.at/Linked-UNData/

set,[18] are directly mapped to an indicator URI from the City Data Ontology, namely http://citydata.wu.ac.at/ns#population.

18 http://citydata.wu.ac.at/Linked-UNData/data/240

### *Unified View over Statistical Linked Data*

As the different data sources use different identifiers (and the wrappers use different URIs), we need to link the varying URIs before we can do an integrated querying of the data. As the foundation for efficiently querying Statistical Linked Data—and in turn enriching the data as described in Section 7.1 and Chapter 6—we define a unified view of all crawled datasets about cities in a simplified version of the global cube [Kämpgen, Stadtmüller and Harth 2014]. In the following, we describe the structure of the global cube.

We define the unified view as the basis for querying as follows. The `qb:Observations` (consisting of dimensions and measures) have the following structure, starting with the dimensions:

- For the time dimension we use `dcterms:date`.
- For the time dimension values we use single years represented as String values such as `"2015"`.
- For the geospatial dimension we use `sdmx-dimension:refArea`, which is recommended by the QB standard.
- For the geospatial dimension values we use instances of `dbpedia:City`, such as `dbpedia:Vienna`.
- For the indicator dimension we use `cd:hasIndicator`.
- For the indicator dimension values we use instances of `cd:Indicator`, such as `cd:population_female`. For the indicator dimension values, we defined the CDP ontology as the main hub of indicator URIs to link to since there was no list with common indicator values.

Most data source follow the practice of using an unspecific measure `sdmx-measure:obsValue` and a dimension indicating the measured variable, e.g., `estatwrap:indic_na`. For the unified view, we thus also assume data cubes to have only one general measure, `sdmx-measure:obsValue`. Please note that there are different equivalent alternative representations of the same information. Specifically for measure properties, in QB there is a choice for the structuring of the observations. Either use a single observation value property and a dedicated indicator dimension, or encode the indicator in the measure property. To sum up: in-line with established usage, we use a single measure property, but that structure contains all the information that would also be present in the alternative representation.

If we want to pose queries over the two datasets, we have two options. Either specifically write the query to consider possibly different identifiers (i.e., need to know all identifiers) or 2) assume existing links and reasoning. Then, if we query for values for the canonical identifiers (as for any other identifier in the equivalence class), we also get the values for the respective

other identifiers. In this chapter, we assume reasoning to allow for flexible addition of new sources without the need to change the queries for each new data source.

Take as an example we want a query all values of the indicator "population" of the area "Vienna", in the year "2010" over data from both datasets. The indicator would be expressed as a dimension, with a URI representing "population" as dimension value. The area would be expressed with a dimension, with a URI representing "Vienna" as dimension value. The query looks like the following:

```
SELECT ?city ?year ?value
WHERE {
 ?obs cd:hasIndicator cd:population ;
      sdmx-dimension:refArea dbpedia:Vienna ;
      dcterms:date ?year ;
      sdmx-measure:obsValue ?value .
}
```

Our unified view uses the basic modelling features of the QB vocabulary. In particular, we model indicators in a way that include what otherwise might be encoded as separate dimensions. In the more complex modelling, we would need to use the union of all dimensions of the source datasets, which would lead to introducing an "ALL" dimension value for those observations that do not distinguish the particular dimension. The "ALL" dimension value would need to be generated for many dimensions, which complicates the representation (see [Kämpgen, Stadtmüller and Harth 2014] for details). Rather than adding a dimension "sex" to encode gender, we create separate indicator URIs, for example for population, population male and population female. A benefit of the relatively simple structure means that queries and rules operating on the unified view are also simple.

We have published the data structure definition of the global cube using the QB vocabulary. Besides the general measure (`sdmx-measure:obsValue`), the `qb:DataStructureDefinition` of the global cube uses the mentioned dimensions `dcterms:date`, `sdmx-dimension:refArea`, and `cd:hasIndicator`. Also, we have defined instances of `qb:AttributeProperty` for `cd:estimatedRMSE` (for describing the error), `cd:preferredObservation` (for linking to more reliable values), `prov:wasGeneratedBy` (for describing provenance information) and `prov:generatedAtTime` (for the time of generation) that help to interpret and evaluate the trustworthiness of values.

Please note that data sources use different identifiers for dimensions and dimension URIs. In the global cube, we use canonical URIs to represent resources from different data sources.

### *Linking and Mapping Data*

We start by explaining the required mappings for dimension URIs, followed by explaining the required mappings for dimension value URIs. In general,

the data from the UNSD wrapper, due the simpler representation in the original data source, requires less mappings than the data from the Eurostat wrapper.

The two data sources exhibit the three dimensions for `dcterms:date` (year), `sdmx-dimension: refArea` (city) and `cd:hasIndicator` (indicator). We map the following dimension URIs of the global cube using `rdfs:subPropertyOf`:

- For the time dimension our wrappers directly use `dcterms:date`. The time dimension hence does not require any further mapping.
- For the geospatial dimension the UNSD wrapper uses `sdmx-dimension:refArea`. The Eurostat wrapper uses different representations for the geospatial dimension, such as `eurostat:geo`, `eurostat:cities` and `eurostat:metroreg`, which we link to `sdmx-dimension:refArea`.
- For the indicator dimension we use `cd:hasIndicator`. Again, the UNSD wrapper directly uses that URI, while the data from the Eurostat wrapper requires links from `eurostat:indic_na` and `eurostat:indic_ur` to `cd:hasIndicator`.

The Eurostat site provides a quite elaborate modelling of dimensions, code lists and so on in SDMX files. The datasets from the Eurostat wrapper use various units such as `:THS` denoting "Thousand" and `:COUNT` denoting that the number was computed from a count operation. However, for the datasets from Eurostat which we consider in the pipeline, all observations provide in the global cube three canonical dimensions, a single observation property, and only single dimension values for the other dimensions. Hence, we can assume that additional dimensions and their values are part of the indicator.

The UNSD site has a simpler structure than Eurostat. The modelling of different dimensions and code lists is less elaborate. Thus, for the UNSD wrapper, we have ensured on the level of the published RDF that each dataset only provides the canonical dimensions.

The two wrappers use different URIs for the same dimensions, e.g., `eurostat:geo` and `sdmx-dimension:refArea`. The wrappers also use different URIs for the same dimension values, e.g.,

- For the time dimension values we use single years represented as String values such as `"2015"`.
- For the geospatial dimension values we link to DBpedia URIs from other representations such as `http://estatwrap.ontologycentral.com/dic/cities#AT001C1` and `http://citydata.wu.ac.at/resource/40/001170#000001`.
- For the indicator dimension values we link to instances of `cd:Indicator`, such as `cd:population` and `cd:population_male`. The UNSD wrapper directly uses these values. For the URIs used in the data from the Eurostat wrapper, we link to instances of `cd:Indicator`.

We now describe how we generated these links to map data from different

sources to the canonical representation, starting with the dimension and dimension value URIs. We manually created the `rdfs:subPropertyOf` triples connecting the Eurostat dimension URIs with our canonical URIs, and semi-automatically generated the indicator URIs from an Excel sheet provided by Eurostat. We then created an RDF document with links from the newly generated URIs to the URIs of the Eurostat wrapper. We manually adapted the UNSD wrapper to use the newly generate URIs as indicator URIs.

We choose to have a one-to-one (functional) mapping of every city from our namespace to the English DBpedia URI, which in our re-published data is encoded by `owl:sameAs` relations. We identify the matching DBpedia URIs for multilingual city names and apply basic entity recognition, similar to Paulheim et al. [Paulheim and Fürnkranz 2012], with three steps using the city names from UNSD data:

- Accessing the DBpedia resource directly and following possible redirects.
- Using the Geonames API[19] to identify the resource.
- For the remaining cities, we manually looked up the URI on DBpedia.

19 http://api.geonames.org/

The mappings of geospatial URIs from the Eurostat wrapper were done in a similar fashion. All the mappings are published online as RDF documents that are accessed during the crawling step.

### *Data Crawling and Integration*

The overall RDF graph can be published and partitioned in different documents. Thus, to access the relevant RDF documents, the system has to resolve the URIs of entities related to the dataset. Related entities are all instances of QB-defined concepts that can be reached from the dataset URI via QB-defined properties. For example, from the URI of a `qb:DataSet` instance, the instance of `qb:DataStructureDefinition` can be reached via `qb:structure`. Similarly, instances of `qb:ComponentProperty` (dimensions/measures) and `skos:Concept` (members) can be reached via links.

Once all numeric data is available as Linked Data, we need to make sure to collect all relevant data and metadata starting from a list of initial URIs. First, the input to the crawling is a seed list of URIs of instances of `qb:DataSets`. One example of a "registry" or "seed list" of dataset URIs is provided by the PlanetData wiki.[20] A seed list of such datasets is published as RDF and considered as input to the crawling. We use two such seed lists: one with links to the relevant instances of `qb:DataSet` from the UNSD wrapper, and another one with links to the relevant instances of `qb:DataSet` from the Eurostat wrapper.

20 http://wiki.planet-data.eu/web/Datasets

Then, Linked Data crawlers deploy crawling strategies for RDF data where they resolve the URIs in the seed list to collect further RDF and in turn resolve a specific (sub-)set of contained URIs. An example Linked Data crawler is LD-Spider [Isele et al. 2010], which uses a depth-first or breadth-first crawling

strategy for RDF data. Linked Data crawlers typically follow links without considering the type.

A more directed approach would apply a crawling strategy that starts with resolving and loading the URIs of `qb:DataSets` relevant for the task, and then in turn resolves and loads instances of QB concepts that can be reached from the dataset URIs.

To specify how to collect Linked Data, we use the Linked Data-Fu language [Stadtmüller et al. 2013] in which rule-based link traversal can be specified. For instance, to retrieve data from all `qb:DataSets`, we define the following rule:

```
{
  ?ds rdf:type qb:DataSet.
} =>
{
  [] http:mthd httpm:GET .
     http:requestURI ?ds .
} .
```

The head of a rule corresponds to an update function of an internal graph representation in that it describes an HTTP method that is to be applied to a resource. In our example, the head of a rule applies a HTTP GET method to the resource `?ds`. The body of a rule corresponds to the condition in terms of triple patterns that have to hold in the internal graph representation. In our example, `?ds` is defined as an instance of `qb:DataSet`.

Similarly, we retrieve instances of `qb:DataStructureDefinition`, `qb:ComponentSpecification`, `qb:DimensionProperty`, `qb:AttributeProperty`, `qb:MeasureProperty`, `qb:Slice`, `qb:SliceKey`, and `qb:ObservationGroup`. Also, we access the list of possible dimension values (based on `qb:codeList` in data structure definitions) as well as each single dimension value. The only instances we do not resolve are observations since these are usually either modelled as blank nodes or provided together with other relevant information with the RDF document containing `qb:DataSet` or `qb:Slice`.

Crawling may include further information, e.g., `rdfs:seeAlso` links from relevant entities or `owl:sameAs` links to equivalent URIs. Assuming that the number of related instances of QB concepts starting from a QB dataset is limited and that links such as `rdfs:seeAlso` for further information are not crawled without restriction (e.g., only from instances of QB concepts), the directed crawling strategy terminates after a finite amount of steps.

Besides all the relevant data and metadata of `qb:DataSets`, we collect the following further information:

- The City Data Ontology[21] (CDP ontology) that contains lists of common statistical indicators about cities.
- The QB Equations Ontology[22] that contains the vocabulary to describe QB equations and is further detailed in Chapter 6.
- The Eurostat QB equations[23] that contains a set of QB equations generated

21 http://citydata.wu.ac.at/ns

22 http://citydata.wu.ac.at/ocdp/qb-equations

23 http://citydata.wu.ac.at/ocdp/eurostat-equations

from formulas published by Eurostat as further detailed in Chapter 6.

- Background information[24] that links indicators of Estatwrap to the CDP ontology as further described in Section 5.3.
- Background information providing additional `owl:equivalentProperty` links[25] between common dimensions not already provided by the wrappers such as between the different indicator dimension URIs `estatwrap:indic_ur`, `cd:hasIndicator` and `eurostat:indic_na`.

24 http://kalmar32.fzi.de/triples/indicator-eurostat-links.nt

25 http://kalmar32.fzi.de/triples/dimension-property-links.nt

Besides explicit information available in the RDF sources, we also materialise implicit information to 1) make querying over the triple store easier and 2) automatically evaluate relevant QB and OWL semantics. We execute the QB normalisation algorithm[26] in case the datasets are abbreviated. Also, we execute entailment rules[27] for OWL and RDFS. However, we only enable those normalisation and entailment rules that we expect to be evaluated quickly and to provide sufficient benefit for querying.

26 https://www.w3.org/TR/vocab-data-cube/#normalize-algorithm

27 http://semanticweb.org/OWLLD/

For instance, we evaluate rules about the semantics of equality, e.g., symmetry and transitivity of `owl:sameAs`. We again describe the semantics of such axioms using Linked Data-Fu. However, because we do not need the full materialisation of the equality, but only the canonical URIs, we define custom rules that only generate the triples involving the canonical URIs. Thus, the resulting dataset contains all triples required to integrate and query the canonical representation, but not more.

The crawling and integration is specified in several Linked Data-Fu programs. The programs are executed periodically using the Linked Data-Fu interpreter[28] in version 0.9.12. The interpreter issues HTTP requests to access the seed list, follows references to linked URIs, and applies the derivation rules to materialise the inferences. The crawled and integrated data is then made available for loading into a triple store. Before loading the observations into the triple store we ensure for each observation that the correct dimension URIs and member URIs are used, filter out non-numeric observation values and mint a new observation URI if a blank node is used. Finally, the filtered and skolemised observations are loaded into an OpenLink Virtuoso triple store (v07) using the standard RDF bulk loading feature[29].

28 https://linked-data-fu.github.io/

Thus, the global cube can be queried in subsequent imputation and calculation steps.

29 See http://citydata.wu.ac.at/ocdp/import for a collection of information about the loading process.

# 6 QB Equations

OWL 2 gives only little support for reasoning with literal values. Although knowledge about the relations of different numeric literals (equational knowledge) exists even in ontologies, for example the QUDT ontology [Ralph Hodgson 2011], it can not be used to infer new literal values by an OWL reasoner since OWL 2 in general can not compute new (numeric) literals. For statistical data, especially statistical linked data, equational knowledge can be interesting to compute derived indicators or fill in the gaps of missing values. Examples for useful equational knowledge include unit conversion, indicator definitions, or linear regression models. With QB equations (QBe) we introduce a framework to exploit equational knowledge to infer numerical data using Semantic Web technologies, with fine-grained provenance tracking and error propagation for inaccurate values. After applying the ML prediction methods in the OCDP workflow, the QBes generate observations from the whole combined dataset. The resulting observations are used for evaluation, consistency checking, and are published if they are new or better than any existing observation with the same dimension members.

*Example* 6.1. The Eurostat indicator "Women per 100 men" is defined as an equation as follows:

$$\text{women\_per\_100\_men} = \frac{\text{population\_female} \cdot 100}{\text{population\_male}}$$

The approach of QBes presented in the following is a combination and extension of two earlier approaches "RDF attribute equations" and "complex correspondences". RDF attribute equations (see Chapter 4) use equational knowledge and give an RDF syntax and a Description Logic semantics to derive numerical OWL data properties from other such properties. The approach was implemented in a backward-chaining manner for SPARQL queries. QBes however, operate on QB observations instead of OWL data properties, are implemented in a forward-chaining manner and provide error propagation and fine-grained provenance tracking. Complex correspondences [Kämpgen, Stadtmüller and Harth 2014] define rules, with numerical functions, to compute QB observations from other QB observations. They transfer the concept of correspondences over relational data to the Semantic Web data model using the QB vocabulary. In contrast to complex correspondences, QBes are given in an RDF syntax and is more generic since it uses (more general)

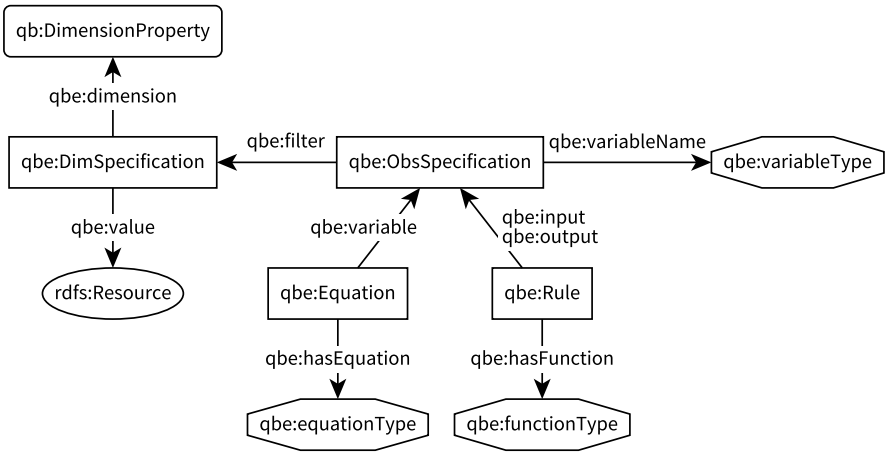



**Figure 6.1**: QB equations ontology

equations instead of functions resulting in more computed values without the need to (manually) create one rule for each variable in the equation.

## 6.1 Syntax

We express QBes in an RDF syntax. Since—to the best of our knowledge—no vocabulary exists for this purpose so far we have to introduce a new vocabulary expressing QBes and QB rules.

Each QBe is identified by a IRI and consists of two parts: (i) a representation of a mathematical equation using (arithmetic) functions and variables, and (ii) a mapping of observations to variables using observation specifications. Figure 6.1 gives an overview of the QB equations ontology showing all the introduced classes, properties, and datatypes as well as reuse of the QB ontology. When encoded in RDF we call these relationships *QB equations* or *QB rules*. QBes specify relationships between observations which can be reformulated into different "directions" while QB rules are valid only in one direction.

*Representation of the Equation or Function* One possibility to represent equations in RDF would be building an operator tree in RDF like the MathML, an XML representation of mathematical concepts, including equations. The SWRL RDF syntax also uses such a verbose syntax for example for predefined mathematical functions.

To keep the representation simple and still human-readable without a custom UI we define the core of the QBes, which is the equation itself, as a literal that directly reuses SPARQL's arithmetic expression syntax,[1] i.e., we use a datatype literal with the datatype qbe:equationType, the lexical space of which is defined by the following grammar rule (in the syntax and referring

1 cf. http://www.w3.org/TR/sparql11-query/#rNumerivExpression

to non-terminal symbols of the SPARQL grammar):

equationType ::= Var '=' NumericExpression

This choice enables standard SPARQL parsers or other standard libraries for mathematical expressions for processing these equations, and—as we will see—straightforward implementation of the application of equations by SPARQL engines. As an example we give again the equation for the Eurostat indicator definition of "Women per 100 men":

"?women_per_100_men = ?population_female * 100 / ?population_male"^^qbe:equationType

The property qbe:hasEquation relates an instance of the class qbe:Equation to such an equation literal. The lexical space of datatype qbe:variableType is—analogous to qbe:equationType—defined by the SPARQL grammar non-terminal 'Var'.

*Observation Specification* The second part maps observations to the variables used in the equation. Usually observations are specified by giving values for all of the dimensions. This approach would be too constraining and might lead to multiple representations of essentially the same equation. Instead an observation specification only needs values for some of the dimensions. In an example of unit conversions, one would only specify the value of the unit dimension because the equation should be applicable to any kind of observation given in that unit, regardless of the values of the other dimensions. Intuitively the values of all other unspecified dimensions must be the same among all specified observations.

*Example* 6.2. The following example shows the complete definition of the equation for the Eurostat indicator "Women per 100 men". The QBe defines a variable ?women_per_100_men which binds to all observations for which the member of the dimension estatwrap:unit is set to cd:women_per_100_men. The other two variables ?mile are defined analogously. Eventually the QBe gives the equation relating the variables as a qbe:equationType-typed literal.

```
ex:women-per-100-men a qbe:Equation ;
  qbe:variable [ a qbe:ObsSpecification ;
    qbe:filter [ a qbe:DimSpecification ;
      qb:dimension cd:hasIndicator ;
      qbe:value cd:women_per_100_men ] ;
    qbe:variablename "?women_per_100_men"^^qbe:variableType ] ;
  qbe:variable [ a qbe:ObsSpecification ;
    qbe:filter [ a qbe:DimSpecification ;
      qb:dimension cd:hasIndicator ;
      qbe:value cd:population_male ] ;
    qbe:variablename "?population_male"^^qbe:variableType ] ;
  qbe:variable [ a qbe:ObsSpecification ;
    qbe:filter [ a qbe:DimSpecification ;
      qb:dimension cd:hasIndicator ;
      qbe:value cd:population_female ] ;
    qbe:variablename "?population_female"^^qbe:variableType ] ;
```

```
qbe:hasEquation "?women_per_100_men␣=␣?population_female␣*␣100␣/␣?population_male"
    ↪ ^^qbe:equationType.
```

The type declarations are only given for completeness and are not necessary in practise.

QBes can be evaluated in multiple directions, effectively creating a function to compute a value for each of the variables from all the other variables. In the example above we can infer observations for each of the three indicators cd:women_per_100_men, cd:population_female, and cd:population_male from the other two. Obviously this works only for invertible functions including the usual arithmetic operators: addition, subtraction, multiplication, and division.[2]. In fact, we can reuse the definition of simple equations from Chapter 4, which guarantee this property:

2 while we have to take care of division by zero, for details cf. Chapter 4

**Definition 6.1** (from Chapter 4). Let $\{x_1, \ldots, x_n\}$ be a set of variables. A *simple equation* $E$ is an algebraic equation of the form $x_1 = f(x_2, \ldots, x_n)$ such that $f(x_2, \ldots, x_n)$ is an arithmetic expression over numerical constants and variables $x_2, \ldots, x_n$ where $f$ uses the elementary algebraic operators '+', '-', '*','/' and contains each $x_i$ exactly once.

Equations of this form can be easily transformed into an equivalent form $x_i = f'(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ for each appearing variable $x_i$, $2 \leq i \leq n$. For instance, in our example the equation can likewise be used to compute cd:population_female:

```
"?women_per_100_men␣*␣?population_male␣/␣100␣=␣?population_female"^^qbe:equationType
```

These equivalent transformations can be easily computed by standard mathematical libraries (which we will use in our implementation, cf. Section 6.3 below). A central piece of this transformation is a function *solve* with two parameters: the equation as string and the name of the target variable to solve for. The *solve* function algebraically solves an equation for a variable and returns a function. For example `solve("a=b/c",c)` would return the function `"b/a"` whereas `solve("a=b/c",b)` would return `"a*c"`. The function *solve* is implemented in every computer algebra system—for example in Maxima with roots going back to the 1960s. That is, we could write

$$f'(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = solve(x_1 = f(x_2, \ldots, x_n), x_i)$$

Analogously to qbe:equationType we define a datatype qbe:functionType describing an arithmetic function or expression whose lexical space is again defined via a SPARQL grammar non-terminal rule:

functionType ::= NumericExpression

Following the example for equationType a *function* for computing "Women per 100 men" is the following:

```
"population_female␣*␣100␣/␣?population_male"^^qbe:functionType
```

*QB rules* (or functions) are similar to equations but can be evaluated only in one direction. Thus QB rules specify not (generic) variables but one or more input variables and exactly one output variable. These variables are specified in the same way as the variables in QBes, while the output variable does not need a qbe:variableName.

*Example* 6.3. A QB rule to convert the values for "Women per 100 men" to integer, using the function round to demonstrate a non-invertible function.

```
ex:qbrule1 a qbe:Rule ;
  qbe:input [
    qbe: filter [
      qbe:dimension cd:hasIndicator ;
      qbe:value cd:women_per_100_men ] ;
    qbe:variablename "?women_per_100_men"^^qbe:variableType ] ;
  qbe:output [
    qbe: filter [
      qbe:dimension cd:hasIndicator ;
      qbe:value cd:women_per_100_men_approx ] ] ;
  qbe:function "round(?women_per_100_men)"^^qbe:functionType.
```

## 6.2 Semantics

We define the semantics of QBes by rewriting to a rule language. In fact SPARQL INSERT queries can be seen as rules over RDF triple stores where the pattern of the INSERT clause is the rule head and the graph pattern in the WHERE clause is the rule body. We note that this "idea" is not new and straightforwardly implementing the same concept as interpreting CONSTRUCT statements as rules, introduced e.g. in [Polleres, Scharffe and Schindlauer 2007], where we defined a formal semantics for such rules based on the Answer Set Semantics for non-monotonic Datalog programs (ASP) with external (built-in) predicates and aggregates [Eiter et al. 2005]; builtin-predicates are introduced in SPARQL 1.1 through expressions and assignment (BIND ... AS), and non-monotonicity in SPARQL is introduced by features such as OPTIONAL and NOT EXISTS.

We explain the semantics of QBes in three steps: (i) normalisation of QBes to QB rules ($N$ QB rules generated from a QBe in $N$ variables), (ii) conversion of QB rules (generated from QBes or as input) to SPARQL INSERT queries, (iii) a procedure to evaluate a program (a set of SPARQL INSERT queries) until a fixpoint is reached. See Figure 6.2 for an overview of the semantics with the example of the Eurostat indicator "Women per 100 men" in three variables used below. Note here, that in the general case rules with the expressive power of ASP with external predicates do not have a unique, finite fixpoint, but we will define/discuss how we can guarantee termination in our case.

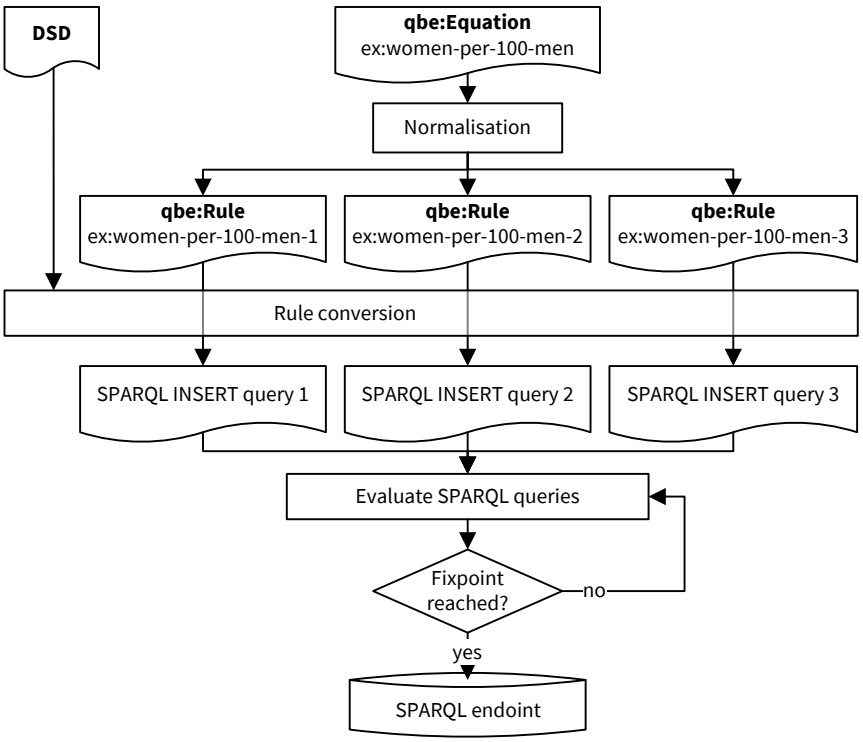



**Figure 6.2:** QB equation workflow with the example of a Eurostat indicator definition/equation

### *Normalisation*

A QBe in $n$ variables can be viewed as a meta rule representing $n$ rules: as discussed before, for each variable $x_i$ a rule to compute $x_i$ from all the other variables in the equation $e$ can be generated by resolving the equation to $x_i = solve(e, x_i)$ . To simplify the semantics specification we thus first normalise each QBe to $n$ QB rules and then in the next step give the semantics for QB rules.

That is, the QB rules generated in the normalisation, have $x_i$ as the output variable, and the other $(n-1)$ variables as input variables with $f'(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ being the function to compute the output.

Listing 6.1 shows the main algorithm of the conversion. The function r in Listing 6.2 takes three parameters: the original QBe IRI, the name of the output variable, and the function. The function sk(...), with a variable number of parameters is a Skolem function deterministically returning a unique IRI for each unique parameter combination.

Eventually, after applying Listing 6.1 we could replace all QBes in an RDF graph with the QB rules in $R$, i.e. we see these representations equivalent. The remainder of our semantics only deals with rules.

*Example* 6.4. After normalisation the QBe of Example 6.2 results in three QB rules, one for each of the two variables. The QB rule to compute the Eurostat

**Listing 6.1**: Algorithm to convert QB equations to QB rules

```
R := {}
for each (?e, ?eq) where { ?e rdf:type qbe:Equation.
                           ?e qbe:hasEquation ?eq }:
    for each (?v, ?vname) where { ?e qbe:variable ?v .
                                  ?v qbe:variableName ?vname }
        ?f := solve(?eq, ?vname)
        add r(?e, ?vname, ?f) to R
return R
```

**Listing 6.2**: Algorithm to create a QB rule

```
def r(?e, ?outvar, ?f):
rule := empty graph
?rulename := sk(?e, ?vname)
rule := { ?rulename a qbe:Rule .
          ?rulename qbe:hasFunction ?f .
          ?rulename prov:wasDerivedFrom ?e .
          ?rulename qbe:output ?outvar . }
for each (?v, ?vname) where { ?e qbe:variable ?v .
                              ?v qbe:variableName ?vname }
    if ?vname != ?outvar:
        add { ?rulename qbe:input ?v } to rule
return rule
```

"Women per 100 men" indicator. Instead of variables we now have `input` and `output` and the equation was replaced by a function in the input variables.

```
ex:women-per-100-men-w a qbe:Rule ;
  prov:wasDerivedFrom ex:women-per-100-men ;
  qbe:input [
    qbe: filter [
      qb:dimension cd:hasIndicator ;
      qbe:value cd:population_male ] ;
    qbe:variablename "?population_male"^^qbe:variableType ] ;
  qbe:input [
    qbe: filter [
      qb:dimension cd:hasIndicator ;
      qbe:value cd:population_female ] ;
    qbe:variablename "?population_female"^^qbe:variableType ] ;
  qbe:output [
    qbe: filter [
      qb:dimension cd:hasIndicator ;
      qbe:value cd:women_per_100_men ] ] ;
  qbe:hasFunction "?population_female * 100 / ?population_male"^^qbe:functionType.
```

For each of the other two indicators cd:population_female and cd:population_male one QB rule is created analogously.

### *Rule Conversion*

In this step QB rules are converted to SPARQL INSERT queries. The query has to implement several tasks: retrieve the input observations, compute the output

observations, generate several IRIs, perform error propagation and provenance tracking and ensure termination when evaluated repeatedly.

Compared to other rule languages SPARQL queries provide very complex features, but, as shown earlier [Polleres and Wallner 2013; Angles and Gutierrez 2008], can be compiled to—essentially—non-recursive Datalog with negation, wherefore INSERT queries, read as rules, have the same expressivity.

Without loss of generality, we make the following assumptions (which could be easily checked in a pre-processing step, e.g., with a SPARQL ASK query assuring that there is a single measure value per observation):

- there is always only a single measure per observation
- the measure predicate that holds the measure value is fixed to sdmx-measure:obsValue

On a high level, the INSERT queries corresponding to QB rules have the following structure:

```
INSERT {
  output observation template
    – with PROV annotations to describe the generation by \abbrev{qb}{}e rules
    – error estimation   }
WHERE {
  one pattern for each input observation
    – with all dimensions as specified in the DSD and error estimate

  BIND patterns for IRI creation for
    – ID for the newly generated observation
    – prov: Activity

  further BIND patterns to
    – assign current time to variable for PROV annotation
    – compute measure value of target observation
    – estimate error of target observation

  Termination condition   }
```

*Output Observation* The output observation is set in the head with the fixed dimensions from the DSD and fixed dimension values if specified in the observation specification of the QB rule. The other dimension values are taken from the input variables. The rule head for Example 6.4 would look like the following query fragment, incorporating the PROV annotations:

```
?obs qb:dataSet globalcube:global–cube–ds ;
  cd:hasIndicator cd:women_per_100_men ;
  dcterms:date ?year ;
  sdmx–dimension:refArea ?city ;
  sdmx–measure:obsValue ?value ;
  prov:wasDerivedFrom ?population_male_obs, ?population_female_obs ;
  prov:wasGeneratedBy ?activity ;
  prov:generatedAtTime ?now ;
  cd:estimatedRMSE ?error .
```

It is important to note that the SPARQL INSERT application is *idempotent*,

i.e., repeated applications of a generated SPARQL INSERT query will not add any more triples after the first application. Idempotence would be lost if blank nodes are used in the head, because they would create a fresh blank node for every application of a SPARQL query, even if the SPARQL query returns only a single result. Furthermore, we have to ensure that all values generated by the query are completely determined by the variable bindings of the WHERE clause.

*Provenance Propagation* For every new derived observation we record the provenance, i.e., each derived observation has a link to each input observation ?obsin1, …, ?obsinN and to the rule or equation used for the computation ?equation. Firstly this provenance information provides transparency: We know precisely how a derived observation was computed. Secondly we use the provenance information during the derivation process to ensure termination. Furthermore, we record the time of the rule application and the agent, which could be the script or person responsible for the query creation.

```
? activity  a prov: activity  ;
  prov: qualifiedAssociation  [
    a prov:Association  ;
    prov:agent    cd:import.sh  ;
    prov:hadPlan <http :// citydata .wu.ac.at/ocdp/eurostat-rules#
        ↪ e4c56a2955372924bde20c2944b2b28f3> ] .
```

The preliminaries in Chapter 2 give an example of a part of the derivation tree generated by this rule head fragment.

*Input Observations* For each input observation one set of triple patterns which asks for one observation is generated for the SPARQL WHERE clause. For each qbe:DimSpecification a dimension value is fixed. For all the other dimensions a fixed variable is used in all input observations. In the example below, again generated from Example 6.4 the query contains for all dimension values variables, except for cd:hasIndicator which is fixed to cd:population_male and cd:population_female as specified by the QB rule input dimension specification. Furthermore the observation value and the error estimated are retrieved.

```
?population_male_obs qb:dataSet globalcube:global-cube-ds;
  cd: hasIndicator  cd:population_male ;
  dcterms:date  ?year;
  sdmx-dimension:refArea ?city  ;
  sdmx-measure:obsValue ?population_male ;
  cd:estimatedRMSE ?population_male_error .

?population_female_obs qb:dataSet globalcube:global-cube-ds;
  cd: hasIndicator  cd:population_female ;
  dcterms:date  ?year;
  sdmx-dimension:refArea ?city  ;
  sdmx-measure:obsValue ?population_female ;
  cd:estimatedRMSE ?population_female_error .
```

*Value Creation with* BIND Several SPARQL variables used for the output observation need to be computed using variables from the input observations. Most importantly the output measure value has to be created using the function of the QB rule.

```
BIND(100.0*?population_female/?population_male AS ?value)
```

Several URIs have to be generated for the rule head. We use a Skolem function to generate these URIs. The inputs of this Skolem function are the IRI of the QB rule rule, the input variables var1, ... varN and a string "_static_" to differentiate the different variables in the head. We implement this Skolem function with string concatenation and a hash function.

```
BIND(IRI(CONCAT(STR(rule), MDA(CONCAT(STR(?var1), ..., STR(?varN))))) AS ?targetvar)
```

We have to generate two URIs: observation, and PROV activity.

```
BIND(CONCAT("http://citydata.wu.ac.at/ocdp/eurostat-rules#", MD5(CONCAT("http://citydata.
    ↪ wu.ac.at/ocdp/eurostat-rules#28f3",STR(?population_male_obs), STR(?
    ↪ population_female_obs))))) AS ?skolem)
BIND(IRI(CONCAT(?skolem, "_obs")) AS ?obs)
BIND(IRI(CONCAT(?skolem, "_activity")) AS ?activity)
```

Furthermore we bind the current time to a variable to use in the provenance part of the head.

```
BIND(NOW() as ?now)
```

*Error Propagation* Values computed based on values with an associated error also need an error estimate. The procedure to estimate an error of the new value is called *error propagation* [Bevington and Robinson 2003; Ku 1966]. In our use case we do not promise precise statistical error quantifications, but just want to propagate an upper bound of the error estimations of the inputs to the computed output value. We chose a error propagation function which is simple to implement in standard SPARQL. To this end, we incorporate a relatively naive error propagation function which however can be adapted to more accurate estimations if necessary in the future [Bevington and Robinson 2003; Ku 1966].

We proceed herein as follows. The error values we have from our predictions are given as RMSE, i.e., the root-mean-square-error, which intuitively characterises how far off in absolute numbers the actual value is on average from our prediction. To compute a conservative estimate of how these errors "add up" when used in computations, we proceed as follows. Depending on the function $f$ used for computing the computed output value, the $n$ variables $x_1, \ldots x_n$ and their associated indicators $ind_1, \ldots ind_n$, we denote by $r_1, \ldots r_n$ the estimated RMSEs for these indicators, i.e. $r_i = RMSE(ind_i)$.

In Table 6.1 we define the propagated estimated RMSE ($per$) of a computed observation recursively over the operator tree of the function term $expr = f(x_1, \ldots x_n)$. Intuitively, we assume here the following: if the real val-

**Table 6.1**: Computing the propagated error (*per*) for a given expression (*expr*)

| *expr* | *per*(*expr*) |
|---|---|
| *const* | 0 |
| $x_i$ | $r_i$ |
| $a + b$ | $per(a) + per(b)$ |
| $a - b$ | $per(a) + per(b)$ |
| $a/b$ | $(\|a\| + per(a))/(\|b\| - per(b)) - a/b$ |
| $a * b$ | $(\|a\| + per(a)) * (\|b\| + per(b)) - a * b$ |

ues $x_i'$ for indicators $ind_i$ lie exactly $r_i$ away—i.e., exactly the estimated RMSE above ($x_i' = x_i + r_i$) or below ($x_i' = x_i - r_i$)— from the predicted value $x_i$, we intend to estimate how much off would a value computed from these predicted values *maximally* be; here, *const* denotes a constant value, and $a, b$ are sub-expressions. Furthermore we assume that the RMSE $r_i$ is always less than the observed value $x_i$.

If now, for an equation $x_f = f(x_1, \ldots x_n)$, the propagated estimated RMSE $per(f(x_1, \ldots x_n))$ is smaller than the so far estimated RMSE $r_f$ for indicator $ind_f$ then we assume it potentially pays off to replace the predicted value so far with the newly computed value by the rule corresponding to the equation.

To cater for rounding errors during the computation we add a small $\epsilon$ of 0.0001 to the error estimate. In some sense this $\epsilon$ punishes each rule application and thus enables quicker termination later. Eventually the following BIND expression will be generated to compute the propagated error as defined by *per* and assign it to the corresponding variable used in the head of the rule.

```
BIND((ABS(100.0)+0.0) *(ABS(?population_female)+?population_female_error)*1.0/ (ABS(?
    ↪ population_male)−?population_male_error)−100.0*?population_female*1.0/?
    ↪ population_male + 0.0001 as ?error)
```

*Termination* So far we introduced triple patterns and BIND expressions into the rule body. As remarked above the BIND expressions implement Skolem functions and thus avoid duplicating the same output observations over and over again (our SPARQL INSERT queries are idempotent). We now give two different termination conditions which can be used separately or together to ensure termination of the QB rules program.

To ensure termination of the whole SPARQL INSERT rule program we use a similar termination condition as in earlier work Chapter 4, we block the repeated application of the same rule to derive a particular observation. With the PROV annotations in fact we create an equation dependency graph. Given an observation $o$, a SPARQL path expression $o$ prov:wasDerivedFrom+ $o'$ returns all the observations $o'$ transitively used in the computation of $o$. Furthermore the SPARQL path expression $o$ prov:wasDerivedFrom*/prov:wasGeneratedBy/prov:qualifiedAssociation/prov:hadPlan $r$ gives all the rules $r$ transitively used during the computation of $o$. So, in order to ensure termination, we define that a QB rule $r$ is only *applicable* to

materialise an observation $o$ if $r$ does not occur in the result of that path expression.

In the SPARQL INSERT query can we implement this condition by adding one of the following patterns for each input observation ?i where r is the IRI of the rule (or equation) itself.

```
FILTER NOT EXISTS {
?i prov:wasDerivedFrom*/prov:wasGeneratedBy/prov:qualifiedAssociation/prov:hadPlan/prov:
    ↪ wasDerivedFrom? r }
```

Thus as a worst case the evaluation will be terminated by this condition after applying each rule $n$ times, where $n$ is the number of QB rules in the system, because after applying each rule once for the derivation of a single observation no rule can be applicable anymore. An example of such a worst case would be a chain of QB rules where $r_i = r_{i+1}$ and $0 < i < n$ and a single given observation for $r_0$.

Another termination condition is based on the error propagation described above. Intuitively the condition ensures that an observation $o$ from a computation is only materialised if no observation $o'$ exists that (i) shares the same dimension values and (ii) has a lower or comparably low error estimate.[3]

3 That is, we add a *confidence threshold* that can be adapted, based on the confidence in the respective error propagation function, in order to only materialise new computed observations if we expect a *significant* improvement

```
?obsa qb:dataSet globalcube:global−cube−ds ;
  dcterms:date ?year;
  sdmx−dimension:refArea ?city ;
  cd:hasIndicator cd:women_per_100_men ;
  sdmx−dimension:sex ?sex ;
  estatwrap:unit ?unit ;
  sdmx−dimension:age ?age ;
  cd:estimatedRMSE ?errora .

  FILTER(?errora <= ?error * CT) }
```

Here, the constant factor CT is a value greater than or equal to 1, that determines a *confidence threshold* of how much improvement with respect to the estimated error is required to confidently "fire" the computation of a new observation. Thus, we materialise only observations that we expect to be significantly (i.e.,by a factor of CT) better with respect to error estimates. Since for any reasonable error propagation function the error estimates tend to increase with each rule application, consequently, together the two termination conditions can lead to faster termination.

For our naive error propagation function, CT = 30.0 turned out to be a reasonable choice, cf. the evaluation results in Section 7.2. Choosing CT = 1 would require an error propagation function with very high confidence, i.e., that never estimates a too low error, which we cannot guarantee for our naive estimation function.[4]

4 Note that this has also been the reason why we introduced the factor CT, as the earlier simpler condition FILTER((?errora <= ?error) produced too many over-optimistic—and in fact worse—observations.

We note here that we really need *both* termination conditions, since relying on error estimates alone would need to ensure that the error propagation "converges" in the sense that application of rules does not decrease error rates. Our simple method for error propagation – in connection with *cyclic*

**Table 6.2**: Example data for Bolzano in the year 2010

| Source | Indicator | Value | Error |
|---|---|---|---|
| Crawl (UN) | Population | 103 582.0 | 0.0 |
| Prediction | No. of available beds per 100 residents | 23.5 | 0.55 |
| QBe | No. of bed-places in tourist accomm. est. | 2 434.5 | 56.6 |
| Prediction | No. of bed-places in tourist accomm. est. | 1 490.5 | 3 228.8 |
| Crawl (UN) | Population male | 49 570.0 | 0.0 |
| Prediction | Population female | 54 836.2 | 7 044.0 |
| QBe | Women per 100 men | 110.6 | 14.3 |
| Crawl | Women per 100 men | 109.0 | 0.0 |

rule application—does not guarantee this as demonstrated by the following, simple example:

*Example* 6.5. For two indicators $i$ and $j$ let two equations be $i = j/2$ and $j = i/2$. Essentially, this boils down to the (cyclic) equation $i = i/4$ where—in each application—we would derive smaller error estimate.

While this example is quite obviously incoherent (in the sense of rules being cyclic in terms of the rule dependency graph defined in Chapter 4[Definition 12]), we still see that with cyclic application of rules the convergence of error rates cannot be ensured in general. In practice such incoherent systems of equations are hardly useful, however the first termination condition would still serve its purpose.

*Example* 6.6. Taking the observations in lines 1 and 2 of Table 6.2 we can compute the "No. of bed-places in tourist accommodation establishments" for Bolzano 2010 as 2434.5 with an RMSE (computed with the propagated error function *per*) of 56.6 (line 3). The QBe observation of line 3 is classified as "better" than the best predicted observation (line 4) because of the RMSE comparison with respect to the confidence threshold: $56.6 \cdot 30 < 3228.8$.

Similarly, the observations from line 5 and 6 are used by the QB equation of the running example to compute the observation in line 7. Since there exists already an observation from the crawl with a better RMSE (line 8), the computed QBe observation will be ignored in this case.

## 6.3 Implementation

As described in Section 6.2 we compile QBes into a semantically equivalent set of rules. Usually there are two strategies for query answering in rule based knowledge bases: forward or backward chaining. For the OCDP we decided to implement a forward chaining approach to enrich the global data cube with the newly inferred observations. Forward chaining approaches materialise as much as possible thus allowing faster query evaluation times. On the other hand forward chaining approaches require more memory or disk space and

updates lead to re-materialisation. In our case the space requirements are manageable and updates are not frequent.

Our forward chaining implementation approach relies on the iterative application of SPARQL INSERT queries (which implement the rules). Overall, QBes infer and persist new observations in three steps: (Normalisation) convert all QBes to QB rules, (Rule conversion) for each QB rule we create a SPARQL query, and (Query evaluation) iteratively evaluate the constructed SPARQL INSERT queries until a fixpoint is reached (that is, no better observations can be derived).

*Normalisation* As described in the semantics above in this first step we convert QBes to QB rules. The algorithm in Listing 6.1 already outlines our implementation. We implemented the algorithm using Python 2.7 and the libraries `rdflib` for RDF/SPARQL processing and `sympy` providing the *solve* function to algebraically solve an equation for a variable. The Python script reads the QBes from an RDF file containing the QBes,[5] converts them to QB rules and publishes them again as Linked Data[6] and in the triple store.

5 http://citydata.wu.ac.at/ocdp/eurostat-equations.rdf

6 http://citydata.wu.ac.at/ocdp/eurostat-rules.rdf

*Creating SPARQL Queries* We create a SPARQL INSERT query for each QB rule. The listing in B gives as a complete example of the SPARQL INSERT query resulting from converting one of the QB rules. Due to a serious performance problem of SPARQL INSERT queries applied on the Virtuoso triple store in a preliminary evaluation, we used SPARQL CONSTRUCT queries instead, and load the resulting triples into the triple store afterwards. The conversion from QB rules to SPARQL CONSTRUCT queries is analogous to the algorithm described in Section 6.2 above to convert a QB rule to a SPARQL INSERT query.

*Evaluating Queries in Rule Engines* In principle the rule engine naively evaluates SPARQL INSERT queries, or respectively, CONSTRUCT queries + re-loads, over and over again until a fixpoint is reached.

Apart from the termination conditions described in Section 6.2 we ensure that the repeated application of a rule on the same observations does not create any new triples by using Skolem constants instead of blank nodes (see also discussion on idempotency above). Thus, in order to check whether a fixpoint has been reached, it is enough to check in each iteration simply if the overall number of triples has changed or not. So, for a naive implementation we could simply use a SPARQL query to count the number of triples in the named graph.

However, unfortunately, in our experiments, such a simple implementation solely based on "onboard" means of the SPARQL engines turned out to be infeasible due to performance reasons. Thus, for the time being, we resorted to just evaluating one iteration of all generated rules, in order to evaluate our conjecture that rules improve our prediction results.

Eventually, we may need to resort to (offline) using a native rule engine.

Indeed, in practical applications such rule/datalog engines have shown to perform better than recursive views implemented directly on top of databases in the past for instance for computing RDFS closure, cf. [Ianni et al. 2009]. For the moment, we leave this to future work and resort, as mentioned, to a fixed number of iterations of rule applications.

## 6.4 Related Work

Modelling the actual numerical data and the structure of that data captures only a part of the knowledge around statistical data that can be represented in a machine-interpretable manner. Equations in particular are a rich source of knowledge in statistical data. Lange [Lange 2013] gives an extensive overview of representations of mathematical knowledge for the Semantic Web. We first cover representation of equations for layout purposes, and then cover representations that permit the interpretation of the formulas by machines.

Non-RDF based representations of mathematical knowledge include MathML [Carlisle and Miner 2014] and OpenMath [Buswell et al. 2004] and use XML for serialisation and focus more on a semantic representation of mathematical entities and are not directly useful for reasoning. Although GeoSPARQL [Perry and Herring 2012] uses MathML in XMLLiterals and OpenMath provides preliminary integration into RDF,[7] these representations are hard to reuse for RDF tools and still are not suitable for an RDF QB setup.

7 http://www.openmath.org/cd/contrib/cd/rdf.xhtml

The OWL ontologies QUDT [Ralph Hodgson 2011], OM [Rijgersberg, Assem and Top 2012], and SWEET [Raskin and M. J. Pan 2005] provide means to describe units and to some extent model conversion between these units, but do not specify a concrete machinery to perform these conversions. Our approach is orthogonal to these efforts in that it provides not only a modelling tool for unit conversions but more general equations and also gives a semantics to automatically infer new values.

Semantic Web rule languages and systems often implement numerical functions—for example RIF uses numerical functions from XPath [Kay et al. 2010]. Other examples for rule languages and systems include SWRL and Apache Jena rules. Converting equations to rules naively can lead to a set of recursive rules which often lead to non-termination even for one equation alone (cf. Chapter 4).

To add reasoning over numbers Description Logics were extended with *concrete domains* (cf. [Baader et al. 2007]). A concrete domain is a domain separate from the usual instance domain of the model based semantics. Examples for concrete domains include different sets of numbers or strings. A specific concrete domain extension defines predicates over the concrete domain, e.g., *greater than* for numbers, or *substring* for strings. Often also a limited set of functions (for computation) can be supplied. Racer [Haarslev and Möller 2001] implements concrete domains with numbers. But computed values are

only used during reasoning and are not available to the user afterwards. OWL Equations [Parsia and Uli Sattler 2012], a concrete domain extension carried over to OWL, allows comparing numerical values—even computed values; still the same limitations apply.

## 6.5 Summary

# 7 Combined Approach

Section 7.1 explains the missing data prediction process in more detail. Both the basic value imputation mechanism and the refinement by QB equations are evaluated in Section 7.2. Section 7.3 puts our approach in the context of related work.

## 7.1 Imputation: Predicting Missing Values

As discussed in Sections 5.1 and 5.2, the filling in of missing values by reasonable predictions is a central requirement for the OCDP, since we discovered a large number of missing values in our datasets (see Tables 5.1 and 5.2).

The prediction workflow is given in Figure 7.1. The initial step regards the loading, transposing, and cleansing of the observations taken from the global cube. Then, for each indicator, we impute all the missing values with neutral values for the principal components analysis (PCA), and perform the PCA on the new matrix, which creates the principal components (PC) that are used as predictors. Next, the predictors are used for the model building step using a basket of statistical Machine learning methods such as multiple linear regression. Finally, we use the best model from the basket to fill in all missing value in the original matrix and publish them using the Missing Values wrapper.

In our earlier work [Bischof, Martin et al. 2015a], we have evaluated two approaches to choose the predictors, one based on applying the base methods to complete subsets in the data and the other based on PCA. In the present paper, we only use the PCA-based approach, since, although it delivers slightly lower prediction accuracy, it allows us to cope more robustly with the partially very sparse data, such that we can also predict values for indicators that do not provide sufficiently large subsets of complete, reliable predictors.

*Base Methods* Our assumption is that every indicator has its own statistical distribution (e.g., normal, exponential, or Poisson distribution), sparsity, and relationship to other indicators. Hence, we aim to evaluate different regression methods and choose the best fitting method/model to predict the missing values per indicator. In order to find this best fitting method, we measure the prediction accuracy by comparing the *normalised root mean squared er-*

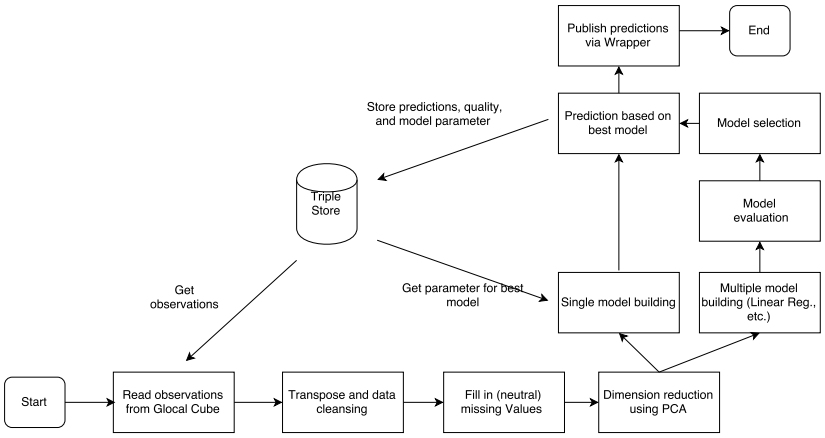



**Figure 7.1**: Prediction Workflow

*ror* in % (NRMSE) [Witten and Frank 2011] of every tested regression method. While in the field of Data Mining [Witten and Frank 2011; Han 2012] (DM) numerous regression methods for missing value prediction were developed, we chose the following three "standard" methods for our evaluation due to their robustness and general performance:

*K-Nearest-Neighbour Regression* (KNN), models denoted as $M_{KNN}$, is a wide-spread DM technique based on using a distance function to a vector of predictors to determine the target values from the training instance space. As stated in [Han 2012], the algorithm is simple, easily understandable and reasonably scalable. KNN can be used in variants for clustering as well as regression.

*Multiple Linear Regression* (MLR), models denoted as $M_{MLR}$, has the goal to find a linear relationship between a target and several predictor variables. The linear relationship can be expressed as a regression line through the data points. The most common approach is *ordinary least squares* to measure and minimise the cumulated distances [Han 2012].

*Random Forest Decision Trees* (RFD), models denoted as $M_{RFD}$, involve the top-down segmentation of the data into multiple smaller regions represented by a tree with decision and leaf nodes. Each segmentation is based on splitting rules, which are tested on a predictor. Decision nodes have branches for each value of the tested attribute and leaf nodes represent decision on the numerical target. A random forest is generated by a large number of trees, which are built according to a random selection of attributes at each node. We use the algorithm introduced by Breiman [Breiman 2001].

*Principal Component Analysis* All three of base methods need a complete data matrix as a basis for calculating predictions for the respective target indicator column. Hence, we need for each target indicator (to be predicted) a complete training data subset of predictor indicators. However, as discussed in [Bischof, Martin et al. 2015a], when dealing with very sparse data, such

complete subsets are very small and would allow us to predict missing values only for a few indicators and cities. Instead, we omit the direct use of indicators as predictors. Instead, we first perform a PCA to reduce the number of dimensions of the data set and use the new compressed dimensions, called *principal components* (PCs) as predictors for the three base methods: as stated in [Han 2012], the PCA is a common technique for finding patterns in data of high dimensions (in our case, many different indicators for many different cities and years). We use PCA to +compress the large number of indicators to a smaller set of +principal components which can later be used as predictors. The second main advantage of PCA is in terms of dealing with sparse data: as described in [Roweis 1997], all the missing values in the raw data matrix can be replaced by *neutral* values for the PCA created according to the so-called *regularised iterative PCA algorithm*. This step allows to perform PCA on the entire data matrix, even if only a few complete subsets exist.

### *Preprocessing*

Before we can apply the PCA and subsequently the base regression methods we need to pre-process and prepare the data from the global cube to bring it into the form of a two-dimensional data matrix. This preprocessing starts with the extraction of the observations from the global cube. Since the described standard DM methods can not deal with the hierarchical, multidimensional data of the global cube, we need to "temporary flatten" the data back to tuples. For this, we pose the following SPARQL query, with an increasing year range that is currently *2004–2017*.

```
SELECT DISTINCT ?city ?indicator ?year ?value
FROM <http://citydata.wu.ac.at/qb-materialised-global-cube>
WHERE {

?obs dcterms:date ?year.
?obs sdmx-dimension:refArea ?city.
?obs cd:hasIndicator ?indicator.
?obs sdmx-measure:obsValue ?value.
{ ?obs a cd:CrawledObservation } UNION { ?obs a cd:factualQBeObservation }.

FILTER(xsd:integer(?year) >= 2004)

} ORDER BY ?indicator ?city ?year
```

The SPARQL query flattens the multidimensional data to an input data table with tuples of the form:

$$\langle City,\ Indicator,\ Year,\ Value \rangle.$$

Based on the initial table, we perform a simple preprocessing as follows:

- Removing nominal columns and encode boolean values;
- Merging the dimensions year and city to one, resulting in: $\langle City\ Year,\ Indicator,\ Value \rangle$;

that is, we further flatten the consideration of city per year to city/year "pairs"
- Finally, we transpose the initial table to a two-dimensional data matrix with one row per city/year-pair one column per indicator, resulting in tuples of the form:
  $\langle City\ Year,\ Indicator_1 Value_1, \ldots, Indicator_n Value_n \rangle$;
- From this large matrix, we delete columns and rows which have a missing values ratio larger than 99%, that is, we remove city/year pairs or indicators that have too many missing values to make reasonable predictions, even when using PCA.

Our initial data set from merging Eurostat and UNSD contains 1 961 cities with 875 indicators. By merging city and year and transposing the matrix we create 12 008 city/year rows. After deleting the cities/year-pairs and indicators with a missing values ratio larger than 99%, we have the final matrix of 6 298 rows (city/year) with 212 columns (indicators).

Note that the flattening approach and deletion of too sparse rows/columns are generic and could obviously still be applied if we added more data sources, but our experiments herein focus on the Eurostat and UNSD data.

### *Prediction using PCA and the Base Regression Methods*

Next, we are ready to perform PCA on the data matrix created in the previous subsection. That is, we *impute* all the missing values with *neutral* values for the PCA, according to the *regularised iterative PCA algorithm* described in [Roweis 1997]. In more detail, the following steps are evaluated having an initial data set $A_1$ as a matrix and a predefined number of predictors $n$ (we test this approach also on different $n's$):

1. Select the target indicator $I_T$;
2. Impute the missing values in $A_1$ using the regularised iterative PCA algorithm resulting in matrix $A_2$ and remove the column with $I_T$;
3. Perform the PCA on the $A_2$ resulting in a matrix $A_3$ of a maximum of 80 PCs;
4. Append the column of $I_T$ to $A_3$ creating $A_4$ and calculate the correlation matrix $A_C$ of $A_4$ between $I_T$ and the PCs;
5. Create the submatrix $A_5$ of $A_4$ on the selection of the PCs with the highest absolute correlation coefficients and limit them by $n$;
6. Create submatrix $A_6$ of $A_5$ for validation by deleting rows with miss. values for $I_T$;
7. Apply stratified tenfold cross-validation on $A_6$. which results in the best performing model $M_{Best}$;
8. Use the method for $M_{Best}$ to build a new model on $A_5$ (not $A_6$) for predicting the missing values of $I_T$.

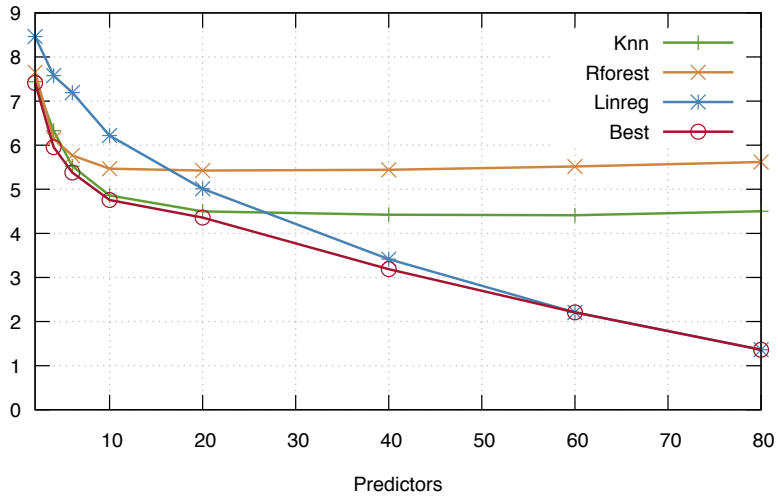



**Figure 7.2:** Prediction results using PCA

### *Evaluation and Publishing*

Figure 7.2 shows the results for the median NRMSE with an increasing number of predictors (selected from the 80 PCs) and compares the performance of KNN, RFD, MLR, and the selection of best method. Clearly, for 80 predictors MLR performs best with a median NRMSE of 0.56%, where KNN (resp. RFD) has a median NRMSE of 4.36% (resp. 5.27%). MLR is the only method that improves steady up to 80 predictors. KNN provides good results for a lower number of predictors, but starts flattening with 20 predictors. Contrary to MLR, the parameter of KNN and MLR have to be adjusted according to number of predictors, hence optimising the number of clusters for KNN could improve the result. The red line in Figure 7.2 shows the median NRMSE with the best regression method chosen. Up to 60 predictors, the overall results improves by selecting the best performing method (for each indicator). The best median NRMSE of 0.55% is reached with 80 predictors, where MLR is predominant and only 5 out of 232 indicators are predicted by KNN. We emphasise that, compared to the result of our earlier experiments in [Bischof, Martin et al. 2015a], the median NRMSE improved from 1.36% to 0.55%, which is mainly related to the lower sparsity of the datasets.

Finally, we note again why we added PCA, as opposed to attempting predictions based on complete subsets: in our preliminary evaluations, based on the comparison of the two approaches in [Bischof, Martin et al. 2015a], by picking the best performing regression method per indicator with ten predictors from the raw data based on complete subsets the median NRMSE was 0.25%. However, due to the low occurrence of complete subsets of reasonable size for ten predictors, only one third of the missing values could be imputed compared to using PCA. We acknowledge that this comes at a cost, as the a median NRMSE when using PCA goes up to 0.55% with 80 predictors. How-

ever, due to the sparsity in the data we decided to trade better completeness for accuracy of the prediction.

We publish the predicted values created by the combination PCA and selecting the best regression method per indicator where we apply a threshold of NRMSE of 20% as a cut off. This leads with the current evaluation to no removal of any indicator. Following our strategy of using statistical linked data wrappers, we publish the predicted values using the *Missing Values wrapper*,[1] which provides a table of content, a structure definition, and datasets that are created for each prediction execution.

1 http://citydata.ai.wu.ac.at/MV-Predictions/

### *Workflow and Provenance*

The full prediction workflow of our statistical prediction for missing values is shown in Figure 7.1 and is based on all observations but the old predicted values in the global cube. The *data preprocessing and transposing* for the input data matrix is written in Python, but all other steps such as *PCA*, *model building*, and *model evaluation* are developed in R [R Development Core Team 2009] using its readily available "standard" packages (another advantage of relying on standard regression methods). All the scripts and their description are available on the website of the *Missing Values wrapper*. We conducted an evaluation of the execution time on our Ubuntu Linux server with 2 cores, 2.6 GHz, and 16 GB of RAM. A single prediction run requires approx. 10min for each indicator (approx. 3 min for each method) resulting in a total time of about 35 hours for all indicators, which still is reasonably doable for rerunning wrappers, recomputing models and predictions in a weekly batch job.

Looking back to Figure 7.1, one can see that the workflow branches after four steps, where we distinguish two cases. In the case of no previous executions, we perform the full prediction steps as described in the previous section. In the case of previous executions, we already have provenance information available in our triple store, which describes the last execution and the related model provenance information (for each indicator). The model provenance includes for each indicator the number of PCs, the number of predictors used from these PCs, the chosen prediction base method, method parameters (i.e., the number of clusters in the KNN), and the NRMSE.

To sum up, we keep provenance for our predictions on three levels:

- For each execution, we publish the median NRMSE over all indicators, number of predictors, creation date, and the creation agent;
- For each indicator, we publish the model provenance data;
- For each predicted value published as a qb:Observation, we provide the overall absolute estimated RMSE (using the predicate cd:estimatedRMSE) and the estimated NRMSE (using the predicate cd:estimatedNormalizedRMSE). Further, we point to better observations (published with an lower NRMSE) us-

ing the predicate cd:preferredObservation which might occur if another approach such as a different base method or QB Equations (discussed in Chapter 6) improve the predicted values.

For describing the model provenance, we use the *MEX vocabulary*, which is compared to other vocabularies (i.e., DMOP [Keet et al. 2015]) lightweight and designed for exchanging machine learning metadata [Esteves et al. 2015]. We use the *MEX Algorithm* layer to describe our prediction method and its parameter and the *MEX Performance* layer to describe the NRMSE. Further, we describe each execution using attributes of *MEX Execution*.

*Example* 7.1. The following example gives an intuition into reading the +data about missing value predictions.

```
@prefix prov: <http :// www.w3.org/ns/prov#> .
@prefix cdmv: <http :// citydata . wu.ac.at/MV-Predictions/> .
@prefix mexp: <http :// mex.aksw.org/mex-perf/> .
@prefix mexc: <http :// mex.aksw.org/mex-core#> .
@prefix mexa: <http :// mex.aksw.org/mex-algo#> .

cvmv:predDS1 rdf:type qb:DataSet .
cvmv:predDS1 prov:wasGeneratedBy cvmv:runP1 .
cvmv:predDS1 dc: title "A3_2004-2016_ncp80_seed_100_pred_80" .

cvmv:runP1 rdf:type mexc:Execution ; prov: Activity .
cvmv:runP1 cdmv:predictionPCs 80 .
cvmv:runP1 mexp:rootMeanSquaredError 1.0705 .
cvmv:runP1 mexc:endsAt "2017-07-31T10:52:02Z"^^xsd:dateTime .

cvmv:runP1 cvmv:hasPredicted cvmv:runP1_1 .
cvmv:runP1_1 mexc:datasetColumn cd:no_bed-
    ↪ places_in_tourist__accommodation_establishments .
cvmv:runP1_1 mexc:hasAlgorithmConfig mexa:Regression .
cvmv:runP1_1 cd:estimatedAbsoluteRMSE 3228.8726 .
cvmv:runP1_1 cd:estimatedNormalizedRMSE 1.78259 .
cvmv:runP1_1 cdmv:size 2737 .

cdmv:obs1 rdf:type cd:PredictedObservation.
cdmv:obs1 cd:hasIndicator no_bed-places_in_tourist__accommodation_establishments .
cdmv:obs1 sdmx-dimension:refArea dbpedia:Bolzano .
cdmv:obs1 dcterms:date "2010" .
cdmv:obs1 sdmx-measure:obsValue 1490.4485 .
cdmv:obs1 cd:estimatedAbsoluteRMSE 3228.8726 .
cdmv:obs1 cd:estimatedNormalizedRMSE 1.78259 .
cdmv:obs1 cd:preferredObservation cdmv:obs1 .
cdmv:obs1 qb:dataSet cvmv:predDS1 .
```

The example shows a qb:DataSet of predicted values generated by a run on the 2017-07-31 using our PCA-based approach. We show one predicted value and its RMSEs for the indicator no_bed-places_in_tourist__accommodation_establishments of the city of Bolzano in the year 2010. The best method for this indicator was MLR which is indicated by the triple: cvmv:runP1_1 mexc:hasAlgorithmConfig mexa:Regression.

The triple cdmv:obs1 cd:preferredObservation cdmv:obs1 states that currently

there is no better prediction available, i.e., that this observation is itself the most preferred (i.e., best) for the respective indicator for this city/year.

In summary, while through the availability of more and new raw data we could improve the prediction quality compared to [Bischof, Martin et al. 2015a], this is—essentially, apart from the more structured workflow and publication using provenance information for all predictions—where we stopped missing value prediction in our earlier work in [Bischof, Martin et al. 2015a]. What we will show next in Chapter 6 is that prediction quality can be further improved by combining the statistical regression methods from this section with ontological background knowledge in the form of equations.

### *Implementation*

There are various possible combinations of QB equations and statistical inferences conceivable. Based on our experiments (and we will further argue these choices the details evaluation of the performance of our approach in Section 7.2 below, we have decided for the following implementation. Here, we follow the workflow described in Figure 5.1 and Section 5.2 above. That is, we proceed in three steps as follows:

1. *Materialisation of observations by application of* QB equations *on the raw data*: in a first step we load the integrated and linked observations from the data integration step (4). Note here that each observation, in order to be considered in our rules needs an cd:estimatedRMSE, which per default is set to 0 for factual observations. However, note that due to the linking of different data sources, we could potentially have several ambiguous observations for the same city, year and indicator in this raw data already, e.g. two or more different population values from different data wrappers materialised in the global cube. Let us say, we have for city C1 in the year 2017 three different population values in three different factual observations from UNdata, Eurostat and dbpedia, i.e. $obs_{UN}$: 1 000 000, $obs_{EuroStat}$: 980 000, and $obs_{Dbpedia}$: 1 020 000. In principle, we take no preference among sources, so we proceed as follows in a preprocessing step: for such case we set the cd:estimatedRMSE to the difference from the average of all available ambiguous values (for the same indicator and city/year pair). That is, we set:

   ```
   :obsUN estimatedRMSE 0.
   :obsEurostat  estimatedRMSE 20000.
   :obsDbpedia estimatedRMSE 20000.
   ```

   After this preprocessing, we apply a first application of QB equations, in which case—obviously—the value from $obs_{UN}$ would be preferred for any equation having cd:population as input indicator.
2. *Materialisation of observations by* statistical missing-value prediction: as a second step, we apply enrichment by *statistical regression methods* and computation of estimated RMSEs per indicator as described in Section 7.1;

these statistical regression methods can potentially benefit already from derived additional factual knowledge from the prior step.

3. *Materialisation of further observations by re-application of* QB equations: finally, using these predictions, we iteratively re-apply *QB equations* wherever we expect (through error propagation) an improvement over the current RMSE by using computed values rather than statistical predictions only.

We remark that one could imagine alternatively to re-iterate steps 3. and 2. as well, i.e., by re-computing statistical models from step 2. based on the application of equations in step 3. again, and so on. However, for instance due to impreciseness of error estimations alone, this would be extremely prone to overfitting and—as expected—rather showed a quality decrease than improvement in some preliminary experiments we ran.

## 7.2 Evaluation

In this section, we summarise experiments we conducted to evaluate both the performance of our crawls as well as the quality of enrichment of our pipeline.

The OCDP runs distributed over several components. The UNData wrapper is a server component that runs at WU Vienna and the Eurostat wrapper is also a server component that runs in a cloud environment. The crawler and rule engine is a client component that dereferences the seed URIs, follows links, and performs the reasoning that lead to the unified global cube. The resulting files are then inserted into the SPARQL endpoint. From that point on, all further enrichment is carried out over the combined global cube via the SPARQL endpoint.

In Section 7.2 we first describe in more detail the process that leads to the global cube accessible via the SPARQL endpoint. We then cover in Section 7.2 the enrichment process, consisting of statistical missing value prediction and of calculating the values based on QB equations. The missing value prediction is performed asynchronously regularly on a workstation, following the regular crawls of the crawler.

### *Crawling and Global Cube Materialisation*

The machine that runs the crawling and rule engine Linked Data-Fu is equipped with two eight-core Intel Xeon E5-2670 CPUs @ 2.60GHz and 256 GB of main memory. Crawling and integration runs separately for Eurostat and UNData. The result is one N-Quads file containing the Eurostat portion of the global cube and one N-Quads file containing the UNData portion of the global cube. We separately run the "rapper" RDF parser over the files to ensure that subsequent SPARQL loading steps do not fail with syntax errors.

The crawling and rule application requires around 6 minutes for Eurostat and around 24 minutes for UNData. For the 1,666,379 observations of Eurostat, 473 RDF documents containing 10,751,759 triples are dereferenced (567 MB). The rule application derives 1,666,619 triples. For the 128,693 observations of UNData, 4,505 RDF documents containing 1,690,231 triples are dereferenced (152 MB). The rule application derives 1,002,135 triples. While accessing UNData yields less triples than accessing Eurostat, the UNData data is distributed over many more files and requires more HTTP requests.[2] Thus, the UNData access takes much longer than the Eurostat access.

2 We wait 500 ms between requests to not overload the server providing data.

Loading the global cube into the Virtuoso SPARQL endpoint requires around 190 seconds. Filtering and skolemisation takes 20 minutes.

### *Combining Statistical Missing-Values-Prediction and QB Equations*

In Section 7.1 we have reported on evaluation of the missing values prediction in detail. Recall that we perform PCA to reduce the number of dimensions, which allows us to impute all the missing values with neutral values for the PCA and then evaluate the quality of the predictions using different ( the respectively best one per indicator) base statistical regression methods.

As for runtime performance, our current statistical prediction runs need on a Ubuntu Linux server with 2 cores and 2.6 GHz approximately 35 hours for all indicators and testing all base methods. The run time might slightly grow with the number of indicators, hence we aim to optimise the predictions runs by using our model provenance information, and evaluate only the best base method, which should reduce the runtime by factor three.

As mentioned in Section 7.1, we have identified two main goals for filling in missing value:

1. It is important to build models which are able to predict many (preferably all) missing values.
2. Second, the prediction accuracy of the models is essential, so that the Open City Data Pipeline can fulfil its purpose of publishing high-quality, accurate data and predictions.

Prediction accuracy in our approach is a median 0.55%RMSE over all indicators for the years 2004–2017, which allows us to predict new 608,848 values on top of the existing 693 684. Recall that despite the use of PCA, this difference occurs, since we drop too sparse rows/columns in the data matrix before PCA, in order to accept at an acceptably low overall median %RMSE, so we cannot predict anything for very sparse areas of the data matrix. Still, while we already discussed in Section 7.1 that the accuracy has improved considerably since our prior work in [Bischof, Martin et al. 2015a], however, as mentioned beforehand, our main goal was to improve these predictions further by the combination with QB equations, i.e. to both improve the quality of predictions and enable to predict more missing values overall.

So, we will next focus on evaluation presenting some numbers on the considered QB equations themselves and their evaluation performance, and then report on the correctness of and improvements by the combination of QB equations with statistical regression methods.

Section 6.3 described the implementation of QB equations in the OCDP. In this section we give some results about the behaviour of the QB equations part of the OCDP system[3] and some evaluation of the QB evaluation themselves and how they improve the results of the whole OCDP.

3 for more details see http://citydata.wu.ac.at/ocdp/import

*Normalisation* The normalisation to generate QB rules from QB equations took 25 seconds to normalise 61 QB equations from Eurostat into 267 QB rules.[4]

Appendix B contains a complete example QB equation from Eurostat and one of the normalised QB rules.

4 the Eurostat indicator definition for the population change over 1 year is the only indicator not expressible in QB equations

*Creating SPARQL Queries* First we filter out 76 QB rules for which at least one input variable matches no existing observation. Such rules can never deliver any results and evaluating them is thus needless. Virtuoso could generally not evaluate 44 QB rules which contain seven or more input variables. Eventually we created 147 SPARQL CONSTRUCT queries in five seconds.

Appendix B shows a complete example of a SPARQL CONSTRUCT query together with the corresponding QB rule.

*Evaluating Queries in Rule Engines* This one iteration of evaluating all generated 147 SPARQL CONSTRUCT queries took 28 minutes (time-outs for 12 queries) and inserted 1.8M observations (46M triples) into the global cube.

### *Combining QB Equations with Statistical Methods*

From the different data sources the OCDP crawler collects 991k observations. The statistical missing-values prediction return 522k observations better than any QB equation observation (if existing). The QB equations return 230k observations better than any other prediction or QB equation observation (if existing); additionally, 232k new observations computed by QB equations were actually not predictable at all (due to bad quality) with the statistical regression methods. Eventually the whole OCDP dataset contains 1975k observations.

Apart from these overall numbers, we provide in the following subsections more details on particular aspects on the correctness of and improvements by the combination of statistical regression methods and QB equations for predicting missing values.

### *Correctness of Generated QB Observations*

Firstly, to show the correctness of the QB equation approach on raw data (first step of the workflow described in Section 7.1, we compared the observations derivable by QB equations only from crawled observations with the corresponding crawled observations. We made the following observations: in the sources we currently consider, equations had already been applied in the raw data before import, and thus applying them beforehand in Step 1 of the workflow in Section 7.1 did not have a notable effect. This can mainly be explained by the fact that our considered equations stem from Eurostat's indicator definitions, and therefore from within *one* dataset, where they are already pre-computed. That is, in the cases of consistent input observations (no more than one observation per city-year-indicator combination) the QBes computed consistent results with respect to the crawl.

Notably, however, for the cases of inconsistent/ambiguous input observations in the raw data, the QB equations also possibly compute ambiguous resulting observations. In fact, we discovered 48 643 such cases of inconsistent/ambiguous observations, that is, multiple observations per city-year-indicator combination. While again, as described in Section 7.1, Step 1, we do not resolve these inconsistencies in the raw data, we "punish" them in the computation by assigning inconsistent input observations with an estimated RMSE corresponding to the deviation from the average above all the inconsistent observations for a single city-year-indicator combination.

We note that using QB equations could also be used to aid consistency checking, for instance our experiments unveiled also a missing factor in one of the Eurostat indicator definitions[5] as well as wrongly mapped cities and indicators during the development process.

5 the indicator "Women per 100 men—aged 75 years and over" is missing a factor 100

In general, it makes sense to evaluate the QB equations based on the crawled data and thus enrich the crawled dataset to achieve better results in the following application of statistical regression methods. However, in our experiments which focused in the UN and Eurostat datasets, the prior application had only marginal effects: in our case almost half of the new observations (10178 of 26452) that could be generated in this way were for the indicator "Women per 100 men", because this is the only Eurostat indicator for which the UN dataset contained both necessary base indicators (population male and population female); the other cases could again be traced back to inconsistencies in the source data.

### *Quality Increase*

To test the quality increase of the combined method we tested which ones were the best observations, comparing statistically predicted observations, with QB-equation-generated observation depending on the estimated RMSE associated with each observation, with real factual observations. As described

in Chapter 6 a QB equation observation is only computed if the estimated RMSE multiplied by the confidence threshold (CT) is smaller than the estimated RMSE of any other corresponding observation. Through experimentation during the development of the OCDP we found a confidence threshold of 30 being a good compromise between data quality without sacrificing too many good observations. We got the same or a better RMSEs for 80 of the 82 tested indicators: these 82 indicators are those for which overlapping indicators from the statistical predictions and QB equations were available together with actual observed values from the crawl.

We have summarised the results of this analysis in Table 7.1 for detailed RMSEs for the predicted and the QB equation observations of all 82 tested indicators. For each indicator, the tables lists in the first three columns the numbers of crawled, predicted and QBe computed predictions. The next three columns list the accuracy in terms of actual RMSE (i.e. not estimated, but comparing real existing observations with values generated through predictions or through QBequations): we see here that, in combination the two methods performed better or equal than statistical predictions alone in most cases (indicated in bold face values in the "Combined" column), i.e., we got as mentioned above the same or better RMSE for 80 out of the tested 82 indicators.

Finally, an important property of the combined method is how precise/accurate the propagated error computation is, since this propagated estimated error (RMSE in our case) is used to decide which observation is classified as the best observation for a given city-year-indicator combination. We thus model this as a binary classification task: for a fixed city-year-indicator combination, given the corresponding prediction observation and the best observation generated by a QB equation, both with error estimates, is the QB equation observation value nearer to the actual value than the predication observation value? In this case we are more interested in a minimal number of false positives (QB equation observation wrongly selected as better) even at the cost of a higher number of false negatives (better QB equation observation wrongly not selected as better). Of the usual measures to quantify the quality of classification tasks we thus are mainly interested in precision. We get an average precision of 90.8% for a confidence threshold of 30, while we miss quite some true positives (significantly lower accuracy). See Table 7.1 for detailed results ("precision" and "accuracy") of all 82 indicators.

As we can demonstrate in even such an incomplete materialisation of equations allows us to predict a significant amount of new or improved observations, that could not be predicted with equal accuracy solely with the methods described in Section 7.1.

**Table 7.1**: Evaluation results 82 indicators (for which crawled, predicted, and QBe observations existed). The "Observation source" lists how many chosen best observations which of the three sources contributed. The "RMSE" columns give the RMSEs of Predictions and QBes as well as the combined system. The quality measures, especially the precision, give an indication how well the error propagation classified better observations. In the cases marked with a * no improvements were observed by the QBes, i.e., the statistical predictions were better than any possible QBe.

| | Observation source | | | RMSE | | | Quality measures | |
|---|---|---|---|---|---|---|---|---|
| Indicator | Crawled | Prediction | QBe | Prediction | QBe | Combined | Precision | Accuracy |
| average size of households | 3 463 | 3 194 | 0 | 0.04 | 0.08 | **0.04** | * | 0.63 |
| crude birth rate per 1 000 inhabitants | 6 739 | 1 079 | 194 | 10.19 | 0.38 | **10.19** | * | 0.36 |
| crude death rate per 1 000 inhabitants | 6 417 | 1 273 | 59 | 9.30 | 0.28 | **9.30** | * | 0.31 |
| economically active population female | 4 517 | 3 025 | 104 | 3 083.12 | 9 636.50 | **3 083.12** | * | 0.72 |
| economically active population male | 4 520 | 3 025 | 104 | 2 887.80 | 12 171.77 | **2 887.80** | * | 0.78 |
| economically active population total | 4 750 | 2 765 | 108 | 4 581.38 | 6 596.52 | **4 581.38** | * | 0.49 |
| employment jobs in agriculture fishery nace rev 2 a | 3 244 | 3 003 | 729 | 231.26 | 1 044.50 | **228.71** | 1.00 | 0.45 |
| employment jobs in construction nace rev 2 f | 3 447 | 2 294 | 1 317 | 775.68 | 1 378.20 | **775.62** | 1.00 | 0.24 |
| employment jobs in mining manufacturing energy nace rev 2 b-e | 3 442 | 2 511 | 1 105 | 2 042.92 | 6 692.59 | **2 042.89** | 1.00 | 0.40 |
| eu foreigners | 4 268 | 774 | 2 341 | 2 777.38 | 597.50 | **2 746.81** | 0.91 | 0.29 |
| eu foreigners as a proportion of population | 4 209 | 2 840 | 306 | 0.65 | 0.42 | **0.65** | * | 0.25 |
| foreign-born | 2 149 | 246 | 3 961 | 18 408.69 | 3 095.48 | **18 305.70** | 0.98 | 0.19 |
| foreign-born as a proportion of population | 2 136 | 4 074 | 134 | 2.94 | 1.52 | **2.94** | * | 0.43 |
| foreigners | 3 290 | 389 | 2 748 | 13 187.75 | 1 394.98 | **13 176.48** | 0.98 | 0.22 |
| foreigners as a proportion of population | 3 261 | 2 914 | 238 | 1.65 | 0.89 | **1.65** | * | 0.38 |
| households owning their own dwelling | 2 624 | 1 449 | 3 036 | 3 998.24 | 50 431.29 | **3 998.24** | * | 0.26 |
| households with children aged 0 to under 18 | 4 023 | 967 | 2 341 | 1 952.82 | 6 703.39 | **1 952.77** | 1.00 | 0.14 |
| infant mortality per year | 5 214 | 1 383 | 1 062 | 1.61 | 7.98 | **1.61** | * | 0.28 |
| infant mortality rate per 1 000 live births | 5 083 | 2 046 | 418 | 0.58 | 1.01 | **0.58** | * | 0.42 |
| lone parent households per 100 househ. with children aged 0–17 | 3 037 | 4 042 | 107 | 0.73 | 3.33 | **0.73** | * | 0.78 |
| lone parent private households with children aged 0 to under 18 | 3 180 | 282 | 3 774 | 674.46 | 341.52 | **674.46** | 1.00 | 0.13 |
| lone pensioner above retirement age households | 3 609 | 2 849 | 766 | 88.58 | 4 590.84 | **88.58** | * | 0.26 |
| nationals | 5 569 | 1 122 | 1 056 | 73 639.26 | 12 052.11 | **73 355.72** | 1.00 | 0.22 |
| nationals as a proportion of population | 5 531 | 1 733 | 457 | 116.24 | 2.92 | **116.24** | 0.94 | 0.33 |
| native-born | 2 159 | 350 | 3 845 | 99 075.25 | 8 725.38 | **99 075.25** | * | 0.18 |

| | Observation source | | | RMSE | | | Quality measures | |
|---|---|---|---|---|---|---|---|---|
| Indicator | Crawled | Prediction | QBe | Prediction | QBe | Combined | Precision | Accuracy |
| native-born as a proportion of population | 2 146 | 3 999 | 197 | 23.54 | 2.84 | **23.54** | * | 0.58 |
| no available beds per 1 000 residents | 3 836 | 3 437 | 30 | 57.56 | 397.42 | **57.56** | * | 0.63 |
| no bed-places in tourist accomm. establishments | 4 226 | 3 | 3 443 | 166 966.67 | 3 349.37 | **166 948.07** | 0.98 | 0.67 |
| no children 0–4 in day care or school | 4 051 | 3 113 | 503 | 579.63 | 522.03 | **579.62** | 0.91 | 0.47 |
| no children 0–4 in day care publ and priv per 1 000 children 0–4 | 3 323 | 3 556 | 93 | 20.92 | 610.03 | **20.91** | 1.00 | 0.74 |
| no cinema seats per 1 000 residents | 2 283 | 3 027 | 1 336 | 33.37 | 1.82 | **33.37** | 1.00 | 0.15 |
| no cinema seats total capacity | 2 660 | 2 035 | 2 278 | 822.75 | 633.30 | **798.6** | 0.99 | 0.18 |
| no deaths in road accidents | 5 574 | 668 | 1 044 | 2.42 | 1.86 | **2.42** | * | 0.20 |
| no households living in apartments | 2 115 | 1 261 | 3 369 | 6 103.92 | 32 588.96 | **6 103.92** | * | 0.27 |
| no households living in houses | 2 153 | 2 027 | 2 542 | 10 502.07 | 24 758.61 | **10 498.78** | 0.88 | 0.26 |
| no live births per year | 6 974 | 231 | 987 | 476.76 | 156.75 | 489.98 | 0.08 | 0.23 |
| no private cars registered | 4 693 | 856 | 1 814 | 43 201.98 | 4 490.18 | **43 047.54** | 0.83 | 0.25 |
| no registered cars per 1 000 population | 4 549 | 2 264 | 464 | 562.66 | 18.06 | **562.59** | 0.90 | 0.31 |
| no tourist overnight stays in reg accomm. per year per resident | 4 821 | 2 674 | 54 | 13.10 | 0.79 | **13.10** | * | 0.14 |
| non-eu foreigners | 4 250 | 377 | 2 730 | 7 884.07 | 1 085.92 | **7 858** | 0.97 | 0.23 |
| non-eu foreigners as a proportion of population | 4 191 | 2 902 | 236 | 1.40 | 0.35 | **1.40** | * | 0.31 |
| one person households | 4 234 | 231 | 2 982 | 3 901.77 | 14 048.54 | **3 900.09** | 1.00 | 0.14 |
| people killed in road accidents per 10 000 pop | 5 172 | 1 765 | 37 | 0.33 | 0.02 | **0.33** | * | 0.19 |
| persons unemployed female | 5 741 | 2 105 | 7 | 734.71 | 387.00 | **734.71** | * | 0.28 |
| persons unemployed male | 5 798 | 2 054 | 7 | 808.36 | 342.44 | **807.88** | 1.00 | 0.22 |
| persons unemployed total | 5 262 | 2 402 | 14 | 450.51 | 523.96 | **450.51** | * | 0.51 |
| population | 17 058 | 50 | 99 081 | 746 087.75 | 746 119.20 | 746 256.30 | 0.19 | 0.34 |
| population female | 14 181 | 580 | 4 380 | 76 682.77 | 75 971.54 | **76 682.77** | * | 0.50 |
| population living in private househ. excl. institutional househ. | 3 602 | 2 498 | 614 | 55 048.78 | 25 328.24 | **55 048.78** | * | 0.25 |
| population male | 14 183 | 4 838 | 122 | 71 411.92 | 71 769.93 | **71 411.92** | * | 0.48 |
| population on the 1st of january 10–14 years total | 4 914 | 272 | 1 546 | 3 831.38 | 656.61 | **3 831.38** | * | 0.23 |
| population on the 1st of january 25–34 years total | 6 416 | 276 | 666 | 13 231.20 | 1 064.67 | **13 214.86** | 1.00 | 0.21 |
| population on the 1st of january 35–44 years total | 6 461 | 194 | 724 | 7 044.37 | 1 020.55 | **7 018.79** | 1.00 | 0.21 |
| population on the 1st of january 45–54 years total | 6 435 | 366 | 551 | 6 395.69 | 838.57 | **6 352.97** | 1.00 | 0.20 |

| | Observation source | | | RMSE | | | Quality measures | |
|---|---|---|---|---|---|---|---|---|
| Indicator | Crawled | Prediction | QBe | Prediction | QBe | Combined | Precision | Accuracy |
| population on the 1st of january 5–9 years total | 4 866 | 211 | 1 629 | 4 084.84 | 717.81 | **4 084.84** | * | 0.22 |
| population on the 1st of january 55–64 years total | 8 379 | 134 | 140 | 5 172.41 | 808.54 | **5 091.16** | 1.00 | 0.27 |
| population on the 1st of january 65–74 years total | 8 401 | 124 | 123 | 4 870.38 | 590.66 | **4 837.58** | 1.00 | 0.25 |
| population on the 1st of january 75 years and over female | 6 860 | 1 140 | * | 471 071.80 | 22 798.19 | **114 272.33** | 0.72 | 0.71 |
| population on the 1st of january 75 years and over male | 6 889 | 1 129 | * | 12 645.78 | 279 859.81 | **12 645.78** | * | 0.72 |
| population on the 1st of january 75 years and over total | 8 413 | 55 | 195 | 5 594.77 | 539.03 | **5 577.07** | 1.00 | 0.20 |
| private households excl. institutional households | 5 130 | 3 | 2 468 | 6 759.09 | 51 025.85 | **6 672.43** | 0.96 | 0.37 |
| proportion households that are lone-pensioner households | 3 508 | 3 700 | * | 0.08 | 0.12 | **0.08** | * | 0.48 |
| proportion of employment in agriculture fishery | 3 172 | 3 525 | 253 | 0.26 | 25.71 | **0.26** | * | 0.55 |
| proportion of employment in construction nace rev11 f | 3 386 | 3 550 | 98 | 0.35 | 7.71 | **0.35** | * | 0.75 |
| proportion of employment in industries nace rev11 c-e | 3 384 | 3 325 | 325 | 1.66 | 9.79 | **1.66** | * | 0.60 |
| proportion of households living in apartments | 1 867 | 4 494 | 265 | 1.97 | 4.67 | **1.97** | * | 0.67 |
| proportion of households living in houses | 1 929 | 4 214 | 482 | 1.80 | 3.08 | **1.80** | * | 0.55 |
| proportion of households living in owned dwellings | 2 299 | 4 308 | 333 | 1.75 | 21.97 | **1.75** | * | 0.67 |
| proportion of households that are 1-person households | 4 128 | 3 254 | 51 | 0.87 | 3.06 | **0.87** | * | 0.77 |
| proportion of households that are lone-parent households | 3 065 | 4 088 | 66 | 0.18 | 0.57 | **0.18** | * | 0.68 |
| proportion of households with children aged 0–17 | 3 917 | 3 345 | 54 | 0.52 | 2.20 | **0.52** | * | 0.79 |
| proportion of population aged 0–4 years | 8 328 | 313 | 0 | 0.08 | 1.00 | **0.08** | * | 0.91 |
| proportion of population aged 10–14 years | 4 893 | 1 817 | 8 | 1.16 | 0.13 | **1.16** | * | 0.62 |
| proportion of population aged 15–19 years | 8 341 | 272 | 0 | 0.10 | 6.72 | **0.10** | * | 0.94 |
| proportion of population aged 20–24 years | 8 326 | 259 | 26 | 0.15 | 9.05 | **0.15** | * | 0.95 |
| proportion of population aged 25–34 years | 6 337 | 771 | 190 | 10.25 | 0.47 | **10.25** | * | 0.35 |
| proportion of population aged 35–44 years | 6 382 | 781 | 156 | 9.27 | 0.50 | **9.27** | * | 0.35 |
| proportion of population aged 45–54 years | 6 356 | 672 | 264 | 10.43 | 0.48 | **10.43** | * | 0.24 |
| proportion of population aged 5–9 years | 4 845 | 1 847 | 0 | 1.15 | 0.14 | **1.15** | * | 0.65 |
| proportion of population aged 65–74 years | 8 384 | 183 | 72 | 8.86 | 0.29 | **8.86** | * | 0.32 |
| proportion of population aged 75 years and over | 8 396 | 198 | 60 | 6.71 | 0.25 | **6.71** | * | 0.36 |
| proportion of total population aged 55–64 | 8 362 | 208 | 74 | 13.61 | 0.40 | **13.61** | * | 0.28 |

## 7.3 Related Work

In the following, we explain how our work distinguishes itself from the related work in the areas of modelling, integrating and querying of numerical data using web technologies, of predicting/imputing of missing values, and of using declarative knowledge for inferencing of numeric information.

The work of Santos et al. [2017] describes a methodology to describe city data for automatic visualisation of indicators in a dashboard. The work is different from our work in several regards. Our work directly reuses W3C standardised vocabularies PROV and QB as well as common Linked Data wrappers and crawlers and as such is more widely applicable. Our work makes use of a pre-existing carefully handcrafted list of common statistical indicators (city data ontology) that was mainly inspired from the Eurostat urban audit but also takes into account other city data sources; the ISO 37120:2014 used by Santos et al. [2017] only lists 100 roughly defined indicators whereas urban audit uses over 200 indicators with available numbers and for some of these indicators also provides computation formulas. The work of Santos et al. [2017] focusses on automatic selection of suitable data for indicators using Prolog inferences and automatically selecting the right visualisation; this was demonstrated with only one indicator bicycle "trips per station". Our work instead focusses on the combination of declarative knowledge and machine learning for deriving/predicting new values from integrated datasets and for that presents a widely-applicable data integration, enrichment and publication pipeline evaluated on a set of more than 200 indicators.

### *Numerical Data in Databases*

Siegel, Sciore and Rosenthal [1994] introduce the notion of semantic values—numeric values accompanied by metadata for interpreting the value, e.g., the unit—and propose conversion functions to facilitate the exchange of distributed datasets by heterogeneous information systems.

Diamantini, Potena and Storti [2013] suggest to uniquely define indicators (measures) as formulas, aggregation functions, semantics (mathematical meaning) of the formula, and recursive references to other indicators. They use mathematical standards for describing the semantics of operations (MathML, OpenMath) and use Prolog to reason about indicators, e.g., for equality or consistency of indicators. In contrast, we focus on heterogeneities occurring in terms of dimensions and members, and allow conversions and combinations.

As a basis for sharing and integration of numerical data, XML is often used [J. M. Pérez et al. 2008]. XML standards such as XCube fulfil requirements for sharing of data cubes [Hümmer, Bauer and Harde 2003] such as the conceptual model of data cubes, the distinction of data (observations) and metadata (dimensions, measures), a network-transportable data format, sup-

port for linking and inclusion concepts, extensibility, conversion capability and OLAP query functionality. Other advantages include that XML allows to define a schema (XML Schema), there are data modification and query languages for XML such as XSLT and XQuery, and there are widely-used XML schemas for representing specific information, e.g., XBRL for financial reports, SDMX for statistics, DDI[6] for research studies. Another XML-based exchange standard for ETL transformations and data warehouse metadata is the Common Warehouse Metamodel (CWM)[7] by the Object Management Group (OMG).

6 http://www.ddialliance.org/

7 http://www.omg.org/spec/CWM/

However, the integration of data across different standards is still an open issue. CWM—but also other interfaces and protocols to share multidimensional datasets such as XML for Analysis and OLE DB—lack a formal definition making it more difficult to use such formalism as a basis for integration [Vassiliadis and Sellis 1999]. XML schemas are concerned with defining a syntactically valid XML document representing some specific type of information. Yet, XML schemas do not describe domain models; without formal domain models, it is difficult to derive semantic relationships between elements from different XML schemas [Klein et al. 2001]. Often, the domain model for an XML schema is represented in a semi-formal way using UML documents and free text. In contrast, schemas described as an OWL or RDFS ontology such as QB have a formal domain model based on logics.

Conceptually, we distinguish the global-as-view (GAV, also known as source-based integration) approach of data integration where the global schema is represented in terms of the data sources and the local-as-view (LAV) approach that requires sources to be defined as views over the global schema [Calvanese, De Giacomo et al. 2001; Cal et al. 2002; Genesereth 2010]. We use the GAV approach and define the global cube in terms of single data cubes using the drill-across operation. With GAV, queries over the global schema can easily be translated to queries over the data sources [Cal et al. 2002]. The advantage of LAV is that the global schema does not need to change with the addition of new data sources. The advantage of GAV is that queries over the global schema can easily be translated to queries over the data sources.

### *RDF Data Pipelines*

Within the Semantic Web community there is extensive work around triplification and building data pipelines and Linked Data wrappers for publicly available data sources on the web, where for instance the LOD2 project has created and promoted a whole stack of tools to support the life cycle of Linked Data, i.e. creating maintainable and sustainable mappings/wrappers of existing data sources to RDF and Linked Data, a good overview is provided in the book chapter by Ngomo et al. [2014] and Auer et al. [2012]. All this work could likewise be viewed as an application of the classical ETL (Extract-Transform-Load) [Golfarelli and Rizzi 2009] methodology extended to work on the web,

based on open standards and Linked Data principles [Berners-Lee 2006]. Our work is not much different in this respect, with the difference that we apply a tailored architecture for a set of selected sources around a focused topic (city data), where we believe that a bespoke combination of rule-based reasoning methods in combination with statistical machine learning can provide added value in terms of data enrichment. This is a key difference to the above-mentioned methods that rather focus on entity linkage and object consolidation in terms of semantic enrichment. However, this focused approach is also different from generic methods for reasoning over Linked Data on the web (cf. e.g. [Aidan Hogan 2011] and references therein for an overview), solely based on OWL and RDFs which (except very basic application of owl:sameAs (for consolidating different city identifiers across sources) and rdfs:subPropertyOf reasoning (for combining overlapping base indicators occurring within different sources).

Other work tries to automatically derive new from existing data. Ambite and Kapoor [2007] present Shim Services providing operations for accessing remote data, integrating heterogeneous data, and deriving new data. Workflows of operations are automatically created based on semantic descriptions of operators. Subsumption reasoning is included to match inputs of services to outputs of other services. To avoid the infinite execution of operations, a limit is defined to the depth of nested operations of the same type. In contrast to the automatically constructed workflows, our pipeline consists of a fixed set of processing steps. Instead of "shim services" that act as stand-alone components accessible via the network, we base the computation on local formulas and use a vocabulary to represent the formulas.

### *Data Modelling and Representation*

Besides the RDF Data Cube Vocabulary (QB) that we are using in this work there are other vocabularies available to publish raw or aggregated multi-dimensional datasets. For instance, there are various OWL ontologies available for representing multidimensional datasets [Niinimäki and Niemi 2009]. Also, several light-weight ontologies have been proposed, such as SCOVO [Hausenblas et al. 2009] and SCOVOLink [Vrandečić et al. 2010]. Other vocabularies for statistical data are the DDI RDF Vocabularies[8], several vocabularies inspired by the Data Documentation Initiative, and the StatDCAT application profile[9] (StatDCAT-AP) to express in a structured way the metadata of statistical datasets which are currently published by the different agencies in the European Union. In comparison to these approaches, we see the following reasons for choosing QB:

8 http://www.ddialliance.org/Specification/RDF

9 https://www.europeandataportal.eu/de/content/statdcat-ap-wg-virtual-meeting

QB, as a W3C recommendation, is an established standard for aggregating and (re-)publishing statistical observations on the web, with off-the-shelf tools to process and visualise QB data. QB's wide adoption is an important factor for data integration use cases, as sources already represented in QB can

be integrated more easily than sources in other representations. Further, QB has shown applicability in various use cases [Kämpgen and Cyganiak 2013], and exhibits the necessary formality to allow efficient and flexible integration and analysis of statistical datasets. The multidimensional data model of QB allows to make explicit different dimension value combinations, e.g., _:obs cd:unit "km2". and _:obs dcterms:date "2010". which is important for interpreting the semantics of values and for integration purposes [Vrandečić et al. 2010].

### *Missing Value Imputation*

Several reference works provide information on handling missing values from the perspective of statistics as well as from social sciences, e.g. cf. [Buhi, Goodson and Neilands 2008; Switzer and Roth 2008]. Within the Semantic Web community, a main focus on value completion has been in the prediction of generic relations, and mainly object relations (i.e. link-prediction) on the object level rather than on numerical values, cf. [Paulheim 2017] for an excellent survey on such methods. The usage of numerical values is a a rather recent topic in this respect. Along these lines, but complementary to the present work, Neumaier, Umbrich, Parreira et al. [2016] (as well as similar works referenced therein) have discussed methods to assign bags of numerical values to property-class pairs in knowledge graphs like DBpedia (tailored to finding out relations such that for instance a certain set of numbers could possibly be "population numbers of cities in France"), but not specifically to complete/impute missing values. Our method rather uses fairly standard, robust, and well-known methods (KNN, linear regression, and random forest) for numerical missing value imputation based on principle components [Roweis 1997]. This could be certainly refined to more tailored methods in the future, for instance using time-series analysis; indeed our predicted values, while reasonably realistic in the value ranges often show some non-realistic "jumps" when raw data for a certain indicator is available over a certain sequence of years, but missing for only a few years in between. Since the missing value imputation component in our architecture is modularly extensible with new/refined methods, such refinements could be added as future work.

Part IV

# Conclusions

# Conclusions and Recommendations

8

Part V

# Appendices

# RDF Prefixes

In the following table we list the used prefixes and corresponding IRIs.

| Prefix | IRI |
|---|---|
| cd: | http://citydata.wu.ac.at/ns# |
| dbr: | http://dbpedia.org/resource/ |
| dbo: | http://dbpedia.org/ontology/ |
| dcterms: | http://purl.org/dc/terms/ |
| ee: | http://citydata.wu.ac.at/ocdp/eurostat-rules |
| estatwrap: | http://ontologycentral.com/2009/01/eurostat/ns# |
| eurostat: | http://estatwrap.ontologycentral.com/ |
| foaf: | http://xmlns.com/foaf/0.1/ |
| owl: | http://www.w3.org/2002/07/owl# |
| prov: | http://www.w3.org/ns/prov# |
| qb: | http://purl.org/linked-data/cube# |
| qbe: | http://citydata.wu.ac.at/qb-equations# |
| rdf: | http://www.w3.org/1999/02/22-rdf-syntax-ns# |
| rdfs: | http://www.w3.org/2000/01/rdf-schema# |
| sdmx-dimension: | http://purl.org/linked-data/sdmx/2009/dimension# |
| sdmx-measure: | http://purl.org/linked-data/sdmx/2009/measure# |

# Complete Example of QB Equation Steps

An excerpt of the Eurostat indicator QB equations[1] in Turtle syntax as example for a QB equation: the Eurostat indicator definition for "Women per 100 men".

1 http://citydata.wu.ac.at/ocdp/eurostat-equations

```
<http :// citydata . wu.ac.at/ocdp/eurostat-equations#women_per_100_men> a qbe:Equation ;
  qbe:variable  [  a qbe:ObsSpecification  ;
    qbe: filter  [  a qbe:DimSpecification  ;
    qb:dimension cd:hasIndicator  ;
    qbe:value  cd:women_per_100_men ] ;
    qbe:variablename "?women_per_100_men"^^qbe:variableType ] ;
  qbe:variable  [  a qbe:ObsSpecification  ;
    qbe: filter  [  a qbe:DimSpecification  ;
    qb:dimension cd:hasIndicator  ;
    qbe:value  cd:population_male ]  ;
    qbe:variablename "?population_male"^^qbe:variableType ] ;
  qbe:variable  [  a qbe:ObsSpecification  ;
    qbe: filter  [  a qbe:DimSpecification  ;
    qb:dimension cd:hasIndicator  ;
    qbe:value  cd:population_female ]  ;
    qbe:variablename "?population_female"^^qbe:variableType ]  ;
  qbe:hasEquation "?women_per_100_men = ?population_female * 100 / ?population_male"
        ↪ ^^qbe:equationType.
```

In the first step this equation is normalised to the following three rules[2].

2 http://citydata.wu.ac.at/ocdp/eurostat-rules

```
ee:e4bb866b19a383a5c7ce88e853ff8bdad a qbe:Rule ;
  prov:wasDerivedFrom <http://citydata . wu.ac.at/ocdp/eurostat-equations#
        ↪ women_per_100_men> ;
  qbe:structure  globalcube:global-cube-dsd ;
  qbe:output [
    qbe: filter  _:b4bb866b19a383a5c7ce88e853ff8bdad ];
  qbe:input  [  qbe:variableName "?women_per_100_men"^^qbe:variableType ;
    qbe: filter  _:b4c56a2955372924bde20c2944b2b28f3 ];
  qbe:input  [  qbe:variableName "?population_male"^^qbe:variableType ;
    qbe: filter  _:b7177d26053419667c2b7deb4569a82b9 ];
  qbe:hasFunction "?population_male*?women_per_100_men/100"^^qbe:functionType .

_:b4bb866b19a383a5c7ce88e853ff8bdad qbe:dimension cd:hasIndicator ; qbe:value cd:
      ↪ population_female .

ee:e4c56a2955372924bde20c2944b2b28f3 a qbe:Rule ;
  prov:wasDerivedFrom <http://citydata . wu.ac.at/ocdp/eurostat-equations#
        ↪ women_per_100_men> ;
  qbe:structure  globalcube:global-cube-dsd ;
  qbe:input  [  qbe:variableName "?population_female"^^qbe:variableType ;
   qbe: filter  _:b4bb866b19a383a5c7ce88e853ff8bdad ];
```

```
  qbe:output [
    qbe: filter   _:b4c56a2955372924bde20c2944b2b28f3 ];
  qbe:input  [  qbe:variableName "?population_male"^^qbe:variableType ;
    qbe: filter   _:b7177d26053419667c2b7deb4569a82b9 ];
  qbe:hasFunction "100*?population_female/?population_male"^^qbe:functionType .

_:b4c56a2955372924bde20c2944b2b28f3 qbe:dimension cd:hasIndicator ; qbe:value cd:
    ↪ women_per_100_men .

ee:e7177d26053419667c2b7deb4569a82b9 a qbe:Rule ;
  prov:wasDerivedFrom <http://citydata.wu.ac.at/ocdp/eurostat-equations#
      ↪ women_per_100_men> ;
  qbe:structure  globalcube:global-cube-dsd ;
  qbe:input  [  qbe:variableName "?population_female"^^qbe:variableType ;
    qbe: filter   _:b4bb866b19a383a5c7ce88e853ff8bdad ];
  qbe:input  [  qbe:variableName "?women_per_100_men"^^qbe:variableType ;
    qbe: filter   _:b4c56a2955372924bde20c2944b2b28f3 ];
  qbe:output [
    qbe: filter   _:b7177d26053419667c2b7deb4569a82b9 ];
  qbe:hasFunction "100*?population_female/?women_per_100_men"^^qbe:functionType .

_:b7177d26053419667c2b7deb4569a82b9 qbe:dimension cd:hasIndicator ; qbe:value cd:
    ↪ population_male .
```

Next the QB rules are converted to SPARQL INSERT/CONSTRUCT queries. We give here the SPARQL query for the second rule above (ee:e4c56a2955372924bde20c2944b2
which computes women_per_100_men.

```
INSERT {
  ?obs qb:dataSet globalcube:global-cube-ds ;
    cd:hasIndicator  cd:women_per_100_men ;
    dcterms:publisher  ?source  ;
    dcterms:date ?year  ;
    sdmx-dimension:refArea ?city  ;
    sdmx-measure:obsValue ?value ;
    prov:wasDerivedFrom ?population_male_obs, ?population_female_obs ;
    prov:wasGeneratedBy ?activity  ;
    prov:generatedAtTime ?now ;
    cd:estimatedRMSE ?error .

  ? activity  a prov: activity  ;
  prov: qualifiedAssociation  [
    a prov:Association  ;
    prov:agent    cd:import.sh  ;
    prov:hadPlan <http :// citydata .wu.ac.at/ocdp/eurostat-rules#
        ↪ e4c56a2955372924bde20c2944b2b28f3> ] .
}
WHERE { { SELECT DISTINCT * WHERE {
  ?population_male_obs qb:dataSet globalcube:global-cube-ds;
    cd:hasIndicator  cd:population_male ;
    dcterms:date ?year;
    sdmx-dimension:refArea ?city  ;
    sdmx-measure:obsValue ?population_male ;
    cd:estimatedRMSE ?population_male_error .

  ?population_female_obs qb:dataSet globalcube:global-cube-ds;
    cd:hasIndicator  cd:population_female ;
    dcterms:date ?year;
```

```
    sdmx-dimension:refArea ?city ;
    sdmx-measure:obsValue ?population_female ;
    cd:estimatedRMSE ?population_female_error .

BIND(CONCAT(REPLACE("http://citydata.wu.ac.at/ocdp/eurostat-rules#
      ↪ e4c56a2955372924bde20c2944b2b28f3", "e4c56a2955372924bde20c2944b2b28f3",
      ↪ MD5(CONCAT("http://citydata.wu.ac.at/ocdp/eurostat-rules#
      ↪ e4c56a2955372924bde20c2944b2b28f3",STR(?population_male_obs), STR(?
      ↪ population_female_obs))))) AS ?skolem)
BIND(IRI(CONCAT(?skolem, "_source")) AS ?source)
BIND(IRI(CONCAT(?skolem, "_obs")) AS ?obs)
BIND(IRI(CONCAT(?skolem, "_activity")) AS ?activity)
BIND(NOW() as ?now)

## computation and variable assignment
BIND(100.0*?population_female*1.0/IF(?population_male != 0, ?population_male, "err") AS
      ↪ ?value)

## error propagation
BIND((ABS(100.0)+0.0)*(ABS(?population_female)+?population_female_error)*1.0/IF((ABS(?
      ↪ population_male)-?population_male_error) != 0.0, (ABS(?population_male)-?
      ↪ population_male_error), "err")-100.0*?population_female*1.0/IF(?
      ↪ population_male != 0, ?population_male, "err") + 0.1 as ?error)
FILTER(?error > 0.0)

## 1st termination condition :
## there exists no better observation with the same dimension values
FILTER NOT EXISTS {
    ?obsa qb:dataSet globalcube:global-cube-ds ;
      dcterms:date ?year;
      sdmx-dimension:refArea ?city ;
      cd:hasIndicator cd:women_per_100_men ;
      sdmx-dimension:sex ?sex ;
      estatwrap:unit ?unit ;
      sdmx-dimension:age ?age ;
      cd:estimatedRMSE ?errora .
    FILTER(?errora < ?error) }

## 2nd termination condition :
## the same equation was not used for the computation of any source observation
FILTER NOT EXISTS { ?population_male_obs prov:wasDerivedFrom*/prov:wasGeneratedBy/
      ↪ prov:qualifiedAssociation/prov:hadPlan/prov:wasDerivedFrom? <http://citydata.wu.
      ↪ ac.at/ocdp/eurostat-rules#e4c56a2955372924bde20c2944b2b28f3> . }
FILTER NOT EXISTS { ?population_female_obs prov:wasDerivedFrom*/prov:wasGeneratedBy/
      ↪ prov:qualifiedAssociation/prov:hadPlan/prov:wasDerivedFrom? <http://citydata.wu.
      ↪ ac.at/ocdp/eurostat-rules#e4c56a2955372924bde20c2944b2b28f3> . }
}}}
```

# Bibliography


Abdel Kader, Riham, Peter Boncz, Stefan Manegold and Maurice van Keulen (2009). "ROX: Run-time Optimization of XQueries". In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. SIGMOD'09, pages 615–626. DOI: 10.1145/1559845.1559910 (cited on page 50).

Abele, Andrejs, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak (2017). *Linking Open Data cloud diagram*. URL: http://lod-cloud.net/ (visited on 12/10/2017) (cited on page 19).

Abiteboul, Serge, Richard Hull and Victor Vianu (1994). *Foundations of Databases*. Addison Wesley (cited on page 40).

Ambite, José Luis and Dipsy Kapoor (2007). "Automatically Composing Data Workflows with Relational Descriptions and Shim Services". In: *Proceedings of the 6th International Semantic Web Conference (ISWC'07)*. Volume 4825. Lecture Notes in Computer Science, pages 15–29 (cited on page 127).

Angles, Renzo and Claudio Gutierrez (2008). "The Expressive Power of SPARQL ". In: *Proceedings of the 7th International Semantic Web Conference (ISWC'08)*. Volume 5318. Lecture Notes in Computer Science, pages 114–129 (cited on page 100).

Arenas, Marcelo, Elena Botoeva, Diego Calvanese, Vladislav Ryzhikov and Evgeny Sherkhonov (2012). "Representability in DL-Lite$_R$ Knowledge Base Exchange". In: *Proceedings of the 25th International Workshop on Description Logics (DL'12)*. CEUR Workshop Proceedings 846 (cited on page 60).

Auer, Sören, Lorenz Bühmann, Christian Dirschl, Orri Erling, Michael Hausenblas, Robert Isele, Jens Lehmann, Michael Martin, Pablo N. Mendes, Bert Van Nuffelen, Claus Stadler, Sebastian Tramp and Hugh Williams (2012). "Managing the Life-Cycle of Linked Data with the LOD2 Stack". In: *Proceedings of the 11th International Semantic Web Conference (ISWC'12)*. Volume 7649. Lecture Notes in Computer Science, pages 1–16 (cited on page 126).

Baader, Franz, Diego Calvanese, Deborah McGuinness, Daniele Nardi and Peter Patel-Schneider, editors (2007). *The Description Logic Handbook: Theory, Implementation, and Applications*. 2nd edition. Cambridge University Press (cited on page 107).

Beckett, Dave, editor (2004). *RDF/XML Syntax Specification (Revised)*. Available at http://www.w3.org/TR/rdf-syntax-grammar/. W3C Recommendation (cited on page 10).

Beckett, David, Tim Berners-Lee, Eric Prud'hommeaux and Gavin Carothers, editors (2014). *RDF 1.1 Turtle*. W3C Recommendation. URL: https://www.w3.org/TR/2014/REC-turtle-20140225/ (cited on page 10).

Berners-Lee, Tim (1989). *Information Management: A Proposal*. URL: https://www.w3.org/History/1989/proposal.html (visited on 12/10/2017) (cited on page 3).

Berners-Lee, Tim (2006). *Linked Data*.
URL: https://www.w3.org/DesignIssues/LinkedData.html (visited on 12/10/2017) (cited on pages 3, 18, 127).
Berners-Lee, Tim, James Hendler and Ora Lassila (2001). "The Semantic Web".
In: *Scientific American*, pages 96–101 (cited on page 3).
Bernstein, Abraham, James Hendler and Natalya Noy (2016). "A New Look at the Semantic Web".
In: *Communications of the ACM* 59.9, pages 35–37. DOI: 10.1145/2890489 (cited on page 3).
Bettencourt, Luís M. A., José Lobo, Dirk Helbing, Christian Kühnert and Geoffrey B. West (2007).
"Growth, innovation, scaling, and the pace of life in cities". In: *Proceedings of the National Academy of Sciences of the United States of America* 104.17, pages 7301–7306 (cited on page 76).
Bevington, Philip R. and D. Keith Robinson (2003).
*Data Reduction and Error Analysis for the Physical Sciences*. 3rd edition. Boston: McGraw-Hill
(cited on page 102).
Bienvenu, Meghyn, Diego Calvanese, Magdalena Ortiz and Mantas Šimkus (2014).
"Nested Regular Path Queries in Description Logics". In: arXiv: 1402.7122 [cs.LO] (cited on page 40).
Bienvenu, Meghyn and Magdalena Ortiz (2015).
"Ontology-Mediated Query Answering with Data-Tractable Description Logics".
In: *Reasoning Web. Web Logic Rules: 11th International Summer School 2015, Tutorial Lectures*.
Volume 9203. Lecture Notes in Computer Science, pages 218–307. DOI: 10.1007/978-3-319-21768-0_9
(cited on pages 5, 17).
Birbeck, Mark and Shane McCarron, editors (2010). *CURIE Syntax 1.0*. W3C Working Group Note.
URL: http://www.w3.org/TR/2010/NOTE-curie-20101216 (cited on page 10).
Bischof, Stefan, Andreas Harth, Benedikt Kämpgen, Axel Polleres and Patrik Schneider (2017).
"Imputation and Calculation on Integrated Statistical Open City Data".
In: *Journal of Web Semantics: Special Issue on Semantic Statistics*. Forthcoming (cited on pages 7, 8).
Bischof, Stefan, Markus Krötzsch, Axel Polleres and Sebastian Rudolph (2014a).
"Schema-Agnostic Query Rewriting in SPARQL 1.1".
In: *Proceedings of the 13th International Semantic Web Conference (ISWC'14)*. Volume 8796.
Lecture Notes in Computer Science, pages 584–600. DOI: 10.1007/978-3-319-11964-9_37
(cited on pages 7, 25, 28, 40).
— (2014b). *Schema-Agnostic Query Rewriting in SPARQL 1.1: Technical report*.
http://stefanbischof.at/publications/iswc14/ (cited on page 25).
— (2015). "Schema-Agnostic Query Rewriting for OWL QL ".
In: *Proceedings of the 28th International Workshop on Description Logics (DL'15)*.
CEUR Workshop Proceedings 1350. Athens, Greece. URL: http://ceur-ws.org/Vol-1350/paper-12.pdf
(cited on pages 7, 25).
— (2017). "Schema-Agnostic Query Rewriting in SPARQL 1.1". In preparation (cited on page 7).
Bischof, Stefan, Christoph Martin, Axel Polleres and Patrik Schneider (2015a).
"Collecting, Integrating, Enriching and Republishing Open City Data as Linked Data".
In: *Proceedings of the 14th International Semantic Web Conference Part II (ISWC'15)*. Volume 9367.
Lecture Notes in Computer Science, pages 57–75. DOI: 10.1007/978-3-319-25010-6_4
(cited on pages 8, 77, 79, 109, 110, 113, 116, 118).
— (2015b). "Open City Data Pipeline: Collecting, Integrating, and Predicting Open City Data".
In: *Proceedings of the 4th Workshop on Knowledge Discovery and Data Mining Meets Linked Open Data*

*co-located with 12th Extended Semantic Web Conference (ESWC'15)*. CEUR Workshop Proceedings 1365. Portoroz, Slovenia. URL: http://ceur-ws.org/Vol-1365/paper3.pdf (cited on page 8).

Bischof, Stefan and Axel Polleres (2013). "RDFS with Attribute Equations via SPARQL Rewriting". In: *The Semantic Web: Semantics and Big Data – Proceedings of the 10th ESWC (ESWC'13)*. Volume 7882. Lecture Notes in Computer Science. Montpellier, France, pages 335–350. DOI: 10.1007/978-3-642-38288-8_23 (cited on pages 7, 77).

Breiman, Leo (2001). "Random forests". In: *Machine learning* 45.1, pages 5–32 (cited on page 110).

Brickley, Dan and R.V. Guha (2014). *RDF Schema 1.1*. URL: https://www.w3.org/TR/rdf-schema/ (cited on pages 15, 19).

Bruijn, Jos de and Stijn Heymans (2007). "Logical Foundations of (e)RDF(S): Complexity and Reasoning". In: *Proceedings of the 6th International Semantic Web Conference (ISWC'07)*. Volume 4825. Lecture Notes in Computer Science, pages 86–99. DOI: 10.1007/978-3-540-76298-0_7 (cited on pages 27, 60).

Buhi, Eric R, Patricia Goodson and Torsten B Neilands (2008). "Out of sight, not out of mind: strategies for handling missing data". In: *American journal of health behavior* 32.1, pages 83–92 (cited on page 128).

*The OpenMath Standard 2.0* (2004). Technical report. The OpenMath Society. URL: http://www.openmath.org/standard/om20 (cited on page 107).

Cabena, Peter, Pablo Hadjinian, Rolf Stadler, Jaap Verhees and Alessandro Zanasi (1998). *Discovering Data Mining: From Concept to Implementation*. Prentice-Hall (cited on page 3).

Cal, Andrea, Diego Calvanese, Giuseppe De Giacomo and Maurizio Lenzerini (2002). "Data Integration Under Integrity Constraints". In: *Information Systems* 29.2 (cited on page 126).

Calvanese, Diego, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi and Riccardo Rosati (2001). "Data Integration in Data Warehousing". In: *International Journal of Cooperative Information Systems* 10.3 (cited on page 126).

Calvanese, Diego, Guiseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini and Riccardo Rosati (2007). "Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family". In: *Journal of Automated Reasoning* 39.3, pages 385–429. DOI: 10.1007/s10817-007-9078-x (cited on pages 5, 16, 25, 58, 62–65, 71).

Carlisle, David and Patrick Ionand Robert Miner, editors (2014). *Mathematical Markup Language (MathML) Version 3.0*. 2nd edition. Available at http://www.w3.org/TR/MathML3/. W3C Recommendation (cited on page 107).

Chekol, Melisachew Wudage, Jérôme Euzenat, Pierre Genevès and Nabil Layaïda (2012). "SPARQL Query Containment Under $\mathcal{SHI}$ Axioms". In: *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI'12)*, pages 10–16 (cited on page 71).

Clément de Saint-Marcq, Vianney le, Yves Deville, Christine Solnon and Pierre-Antoine Champin (2012). "Castor: A Constraint-Based SPARQL Engine with Active Filter Processing". In: *The Semantic Web: Research and Applications – Proceedings of the 9th ESWC (ESWC'12)*. Volume 7295. Lecture Notes in Computer Science, pages 391–405 (cited on page 71).

CloudFlower (2016). *Data Science Report 2016*. URL: http://visit.crowdflower.com/rs/416-ZBE-142/images/CrowdFlower_DataScienceReport_2016.pdf (cited on page 3).

Cyganiak, Richard, Dave Reynolds and Jeni Tennison, editors (2014). *The RDF Data Cube Vocabulary*. URL: https://www.w3.org/TR/vocab-data-cube/ (cited on pages 7, 19, 22, 76).

Di Pinto, Floriana, Domenico Lembo, Maurizio Lenzerini, Riccardo Mancini, Antonella Poggi, Riccardo Rosati, Marco Ruzzi and Domenico Fabio Savo (2013). "Optimizing query rewriting in ontology-based data access". In: *Proceedings of the 16th International Conference on Extending Database Technology (EDBT'13)*. ACM, pages 561–572. DOI: 10.1145/2452376.2452441 (cited on page 25).

Diamantini, Claudia, Domenico Potena and Emanuele Storti (2013). "A Logic-Based Formalization of KPIs for Virtual Enterprises". In: *Advanced Information Systems Engineering Workshops* (cited on page 125).

DuCharme, Bob (2013). *Learning SPARQL: Querying and Updating with SPARQL 1.1*. 2nd edition. O'Reilly Media (cited on page 15).

Eiter, Thomas, Giovambattista Ianni, Roman Schindlauer and Hans Tompits (2005). "A uniform integration of higher-order reasoning and external evaluations in answer-set programming". In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 90–96 (cited on page 97).

Ermilov, Ivan, Jens Lehmann, Michael Martin and Sören Auer (2016). "LODStats: The Data Web Census Dataset". In: *Proceedings of the 15th International Semantic Web Conference Part II (ISWC'16)*. Volume 9982. Lecture Notes in Computer Science, pages 38–46. DOI: 10.1007/978-3-319-46547-0_5 (cited on page 19).

Esteves, Diego, Diego Moussallem, Ciro Baron Neto, Tommaso Soru, Ricardo Usbeck, Markus Ackermann and Jens Lehmann (2015). "MEX vocabulary: a lightweight interchange format for machine learning experiments". In: *Proceedings of the 11th International Conference on Semantic Systems, SEMANTICS 2015, Vienna, Austria, September 15-17, 2015*, pages 169–176 (cited on page 115).

Franconi, Enrico, Jos de Bruijn and Sergio Tessaris (2005). "Logical Reconstruction of Normative RDF". In: *Proceedings of the OWLED 2005 Workshop on OWL: Experiences and Directions*. CEUR Workshop Proceedings 188 (cited on page 41).

Frühwirth, Thom W. (2006). "Constraint handling rules: the story so far". In: *Proceedings of the 8th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'06)*, pages 13–14 (cited on page 71).

Genesereth, Michael R. (2010). *Data Integration: The Relational Logic Approach*. San Rafael, California (USA): Morgan & Claypool Publishers. DOI: 10.2200/S00226ED1V01Y200911AIM008 (cited on page 126).

Glimm, Birte, Adian Hogan, Markus Krötzsch and Axel Polleres (2012). "OWL: Yet to arrive on the Web of Data?" In: *WWW 2012 Workshop on Linked Data on the Web*. CEUR Workshop Proceedings 937. URL: http://ceur-ws.org/Vol-937/ldow2012-paper-16.pdf (cited on pages 19, 71).

Glimm, Birte and Chimezie Ogbuji, editors (2013). *SPARQL 1.1 Entailment Regimes*. Available at http://www.w3.org/TR/sparql11-entailment/. W3C Recommendation (cited on pages 25, 39).

Golfarelli, Matteo and Stefano Rizzi (2009). *Data Warehouse Design: Modern Principles and Methodologies*. McGraw-Hill (cited on page 126).

Gottlob, Georg and Thomas Schwentick (2012). "Rewriting Ontological Queries into Small Nonrecursive Datalog Programs". In: *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*, pages 254–263 (cited on page 71).

Gray, Jim, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow and Hamid Pirahesh (1997). "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals". In: *Data Mining and Knowledge Discovery* 1.1, pages 29–53. DOI: 10.1023/A:1009726021843. URL: https://doi.org/10.1023/A:1009726021843 (cited on page 19).

Guo, Yuanbo, Zhengxiang Pan and Jeff Heflin (2005). "LUBM: A benchmark for OWL knowledge base systems". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 3.2. Selected Papers from the International Semantic Web Conference 2004, pages 158–182. DOI: 10.1016/j.websem.2005.06.005 (cited on page 49).

Gutierrez, Claudio, Carlos Hurtado and Alberto O. Mendelzon (2004). "Foundations of Semantic Web Databases". In: *Proceedings of the 23rd Symposium on Principles of Database Systems (PODS'04)*. Paris, France, pages 95–106 (cited on page 41).

Haarslev, Volker and Ralf Möller (2001). "Racer system description". In: *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR'01)*. Volume 2083. Lecture Notes in Computer Science, pages 701–705 (cited on pages 68, 107).

— (2003). "Description Logic Systems with Concrete Domains: Applications for the Semantic Web". In: *Proceedings of the 10th International Workshop on Knowledge Representation meets Databases (KRDB'03)* (cited on page 68).

Han, Jiawei (2012). *Data Mining: Concepts and Techniques*. 3rd edition. Morgan Kaufmann Publishers Inc. (cited on pages 3, 4, 110, 111).

Harris, Steve and Andy Seaborne, editors (2013). *SPARQL 1.1 Query Language*. Available at http://www.w3.org/TR/sparql11-query/. W3C Recommendation (cited on pages 11, 12, 19, 25, 41, 42, 63).

Hausenblas, Michael, Wolfgang Halb, Yves Raimond, Lee Feigenbaum and Danny Ayers (2009). "SCOVO: Using Statistics on the Web of Data". In: *6th European Semantic Web Conference (ESWC)* (cited on page 127).

Hayes, Patrick, editor (2004). *RDF Semantics*. Available at http://www.w3.org/TR/rdf-mt/. W3C Recommendation (cited on page 59).

Heath, Tom and Christian Bizer (2011). *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers. DOI: 10.2200/S00334ED1V01Y201102WBE001 (cited on pages 3, 21).

Hitzler, Pascal, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider and Sebastian Rudolph, editors (2009). *OWL 2 Web Ontology Language: Primer*. Available at http://www.w3.org/TR/owl2-primer/. W3C Recommendation (cited on pages 15, 27).

Hitzler, Pascal, Markus Krötzsch and Sebastian Rudolph (2009). *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC (cited on pages 11, 15).

Hogan, Aidan (2011). "Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora". Available from http://aidanhogan.com/docs/thesis/. PhD thesis. Digital Enterprise Research Institute, National University of Ireland, Galway (cited on page 127).

Horridge, Matthew and Sean Bechhofer (2011). "The OWL API: A Java API for OWL Ontologies". In: *Semantic Web Journal* 2.1. Special Issue on Semantic Web Tools and Systems, pages 11–21. DOI: 10.3233/SW-2011-0025 (cited on page 49).

Horrocks, Ian and Peter F. Patel-Schneider (2004). "A Proposal for an OWL Rules Language". In: *Proceedings of the 13th International Conference on World Wide Web (WWW'04)*, pages 723–731 (cited on page 68).

Horrocks, Ian, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin N. Grosof and Mike Dean, editors (2004). *SWRL: A Semantic Web Rule Language*. Available at http://www.w3.org/Submission/SWRL/. W3C Member Submission (cited on pages 58, 68).

Hümmer, Wolfgang, Andreas Bauer and Gunnar Harde (2003). "XCube – XML for Data Warehouses". In: *6th ACM International Workshop on Data Warehousing and OLAP (DOLAP)* (cited on page 125).

Ianni, Giovambattista, Alessandra Martello, Claudio Panetta and Giorgio Terracina (2009). "Efficiently Querying RDF(S) Ontologies with Answer Set Programming". In: *Journal of Logic and Computation* 19.4, pages 671–695. DOI: 10.1093/logcom/exn043 (cited on page 107).

Isele, Robert, Jürgen Umbrich, Christian Bizer and Andreas Harth (2010). "LDspider: An open-source crawling framework for the Web of Linked Data". In: *Proceedings of the 9th International Semantic Web Conference (ISWC'10) Posters and Demos*. CEUR Workshop Proceedings 658, pages 29–32. URL: http://ceur-ws.org/Vol-658/paper495.pdf (cited on page 89).

Janowicz, Krzysztof, Frank van Harmelen, James A Hendler and Pascal Hitzler (2015). "Why the Data Train Needs Semantic Rails". In: *AI Magazine* 36.1, pages 5–14. DOI: 10.1609/aimag.v36i1.2560 (cited on page 3).

Jena, Apache, editor (2017). *A free and open source Java framework for building Semantic Web and Linked Data applications*. URL: http://jena.apache.org/index.html (visited on 12/10/2017) (cited on pages 48, 68).

Kämpgen, Benedikt and Richard Cyganiak (2013). *Use Cases and Lessons for the Data Cube Vocabulary*. URL: http://www.w3.org/TR/2013/NOTE-vocab-data-cube-use-cases-20130801/ (cited on page 128).

Kämpgen, Benedikt, Seán O'Riain and Andreas Harth (2012). "Interacting with Statistical Linked Data via OLAP Operations". In: *9th Extended Semantic Web Conference (ESWC) Satellite Events* (cited on page 76).

Kämpgen, Benedikt, Steffen Stadtmüller and Andreas Harth (2014). "Querying the Global Cube: Integration of Multidimensional Datasets from the Web". In: *Proceedings of the 19th International Conference on Knowledge Engineering and Knowledge Management (EKAW'14)*, pages 250–265. DOI: 10.1007/978-3-319-13704-9_20 (cited on pages 86, 87, 93).

Kay, Michael, Norman Walsh, Ashok Malhotra and Jim Melton (2010).
*XQuery 1.0 and XPath 2.0 Functions and Operators*. 2nd edition. W3C Recommendation.
URL: http://www.w3.org/TR/2010/REC-xpath-functions-20101214/ (cited on page 107).

Keet, C. Maria, Agnieszka Ławrynowicz, Claudia d'Amato, Alexandros Kalousis, Phong Nguyen, Raul Palma, Robert Stevens and Melanie Hilario (2015). "The Data Mining OPtimization Ontology".
In: *Web Semantics: Science, Services and Agents on the World Wide Web* 32, pages 43–53 (cited on page 115).

Klein, Michel, Dieter Fensel, Frank van Harmelen and Ian Horrocks (2001).
"The relation between ontologies and schema-languages".
In: *Linköping Electronic Articles in Computer and Information Science* 6.4 (cited on page 126).

Klyne, Graham, Jeremy J. Carroll and Brian McBride (2014). *RDF 1.1 Concepts and Abstract Syntax*.
URL: https://www.w3.org/TR/rdf11-concepts/ (cited on pages 9, 19).

Kontchakov, Roman, Carsten Lutz, David Toman, Frank Wolter and Michael Zakharyaschev (2011).
"The Combined Approach to Ontology-Based Data Access".
In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 2656–2661 (cited on page 71).

Kontchakov, Roman, Mariano Rodriguez-Muro and Michael Zakharyaschev (2013).
"Ontology-Based Data Access with Databases: A Short Course". In: *Reasoning Web*. Volume 8067. Lecture Notes in Computer Science. Mannheim, Germany, pages 194–229.
DOI: 10.1007/978-3-642-39784-4_5 (cited on pages 17, 25).

Krötzsch, Markus (2012). "OWL 2 Profiles: An Introduction to Lightweight Ontology Languages".
In: *Reasoning Web. Semantic Technologies for Advanced Query Answering: 8th International Summer School 2012*, pages 112–183. DOI: 10.1007/978-3-642-33158-9_4 (cited on page 17).

Ku, Harry H. (1966). "Notes on the use of propagation of error formulas".
In: *Journal of Research of the National Bureau of Standards* 70.4 (cited on page 102).

Lange, Christoph (2013).
"Ontologies and languages for representing mathematical knowledge on the Semantic Web".
In: *Semantic Web* 4.2, pages 119–158 (cited on page 107).

Lebo, Timothy, Deborah McGuinness and Satya Sahoo, editors (2013). *PROV-O: The PROV Ontology*.
URL: http://www.w3.org/TR/2013/REC-prov-o-20130430/ (cited on pages 21, 76).

Lutz, Carsten, Inanç Seylan, David Toman and Frank Wolter (2013).
"The combined approach to OBDA: Taming role hierarchies using filters".
In: *Proceedings of the 12th International Semantic Web Conference (ISWC'13)*. Volume 8218. Lecture Notes in Computer Science, pages 314–330. DOI: 10.1007/978-3-642-41335-3_20 (cited on page 49).

Ma, Li, Yang Yang, Zhaoming Qiu, Guotong Xie, Yue Pan and Shengping Liu (2006).
"Towards a Complete OWL Ontology Benchmark".
In: *Proceedings of the 3rd European Semantic Web Conference (ESWC'06)*.
Lecture Notes in Computer Science, pages 125–139. DOI: 10.1007/11762256_12 (cited on page 49).

Mercer (2017). *Quality of Living City Rankings*. URL: http://mercer.com/qol (visited on 14/02/2017) (cited on page 4).

Moreau, Luc and Paul Groth (2013). *Provenance: an Introduction to PROV*. Volume 3. Synthesis Lectures on the Semantic Web: Theory and Technology 4. Morgan & Claypool Publishers. DOI: 10.2200/S00528ED1V01Y201308WBE007 (cited on page 21).

Motik, Boris, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue and Carsten Lutz, editors (2009). *OWL 2 Web Ontology Language: Profiles*. Available at http://www.w3.org/TR/owl2-profiles/. W3C Recommendation (cited on pages 17, 25, 27, 28).

Motik, Boris, Ulrike Sattler and Rudi Studer (2005). "Query Answering for OWL DL with Rules". In: *Journal of Web Semantics* 3.1, pages 41–60 (cited on page 68).

Muñoz, Sergio, Jorge Pérez and Claudio Gutiérrez (2007). "Minimal Deductive Systems for RDF". In: *Proceedings of the 4th European Semantic Web Conference (ESWC'07)*. Volume 4519. Lecture Notes in Computer Science, pages 53–67. DOI: 10.1007/978-3-540-72667-8_6 (cited on page 27).

Naisbitt, John (1982). *Megatrends: Ten New Directions Transforming Our Lives*. 1st edition. Warner Books (cited on page 3).

Neumaier, Sebastian, Jürgen Umbrich, Josiane Xavier Parreira and Axel Polleres (2016). "Multi-level Semantic Labelling of Numerical Values". In: *Proceedings of the 15th International Semantic Web Conference (ISWC 2016) - Part I*. Volume 9981. Lecture Notes in Computer Science. Kobe, Japan, pages 428–445 (cited on page 128).

Neumaier, Sebastian, Jürgen Umbrich and Axel Polleres (2016). "Automated Quality Assessment of Metadata across Open Data Portals". In: *ACM Journal of Data and Information Quality (JDIQ)* 8.1, 2:1–2:29. DOI: 10.1145/2964909 (cited on page 76).

Ngomo, Axel-Cyrille Ngonga, Sören Auer, Jens Lehmann and Amrapali Zaveri (2014). "Introduction to Linked Data and Its Lifecycle on the Web". In: *Reasoning Web. Reasoning on the Web in the Big Data Era - 10th International Summer School 2014, Athens, Greece, September 8-13, 2014. Proceedings*, pages 1–99 (cited on page 126).

Niinimäki, Marko and Tapio Niemi (2009). "An ETL Process for OLAP Using RDF/OWL Ontologies". In: *Journal on Data Semantics* 13, pages 97–119. DOI: 10.1007/978-3-642-03098-7_4 (cited on page 127).

Office for Official Publications of the European Communities (2004). *Urban Audit. Methodological Handbook* (cited on page 81).

Parsia, Bijan and Uli Sattler (2012). *OWL 2 Web Ontology Language Data Range Extension: Linear Equations*. Technical report. https://www.w3.org/TR/owl2-dr-linear/ (cited on page 108).

Patel-Schneider, Peter F. and Boris Motik, editors (2009). *OWL 2 Web Ontology Language: Mapping to RDF Graphs*. Available at http://www.w3.org/TR/owl2-mapping-to-rdf/. W3C Recommendation (cited on pages 16, 27).

Paulheim, Heiko (2017). "Knowledge graph refinement: A survey of approaches and evaluation methods". In: *Semantic Web* 8.3, pages 489–508 (cited on page 128).

Paulheim, Heiko and Johannes Fürnkranz (2012). "Unsupervised generation of data mining features from linked open data". In: *Proceedings of the WIMS 2012*, page 31 (cited on page 89).

Pérez, Jorge, Marcelo Arenas and Claudio Gutierrez (2009). "Semantics and Complexity of SPARQL ". In: *ACM Transactions on Database Systems* 34.3, pages 1–45. DOI: 10.1145/1567274.1567278 (cited on pages 13, 14, 42, 62, 63, 71).

— (2010). "nSPARQL: A Navigational Language for RDF". In: *Journal of Web Semantics* 8 (4), pages 255–270. DOI: 10.1016/j.websem.2010.01.002 (cited on page 40).

Pérez, Juan Manuel, Rafael Berlanga Llavori, María José Aramburu and Torben Bach Pedersen (2008). "Integrating Data Warehouses with Web Data: A Survey". In: *IEEE Trans. Knowl. Data Eng.* 20.7, pages 940–955. DOI: 10.1109/TKDE.2007.190746 (cited on page 125).

Pérez-Urbina, Héctor, Boris Motik and Ian Horrocks (2010). "Tractable Query Answering and Rewriting Under Description Logic Constraints". In: *Journal of Applied Logic* 8.2, pages 186–209. DOI: 10.1016/j.jal.2009.09.004 (cited on pages 25, 48, 71).

Perry, Matthew and John Herring (2012). *OGC GeoSPARQL-A geographic query language for RDF data.* Technical report (cited on page 107).

Poggi, Antonella, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini and Riccardo Rosati (2008). "Linking Data to Ontologies". English. In: *Journal on Data Semantics X.* Volume 4900. Lecture Notes in Computer Science, pages 133–173. DOI: 10.1007/978-3-540-77688-8_5 (cited on page 59).

Polleres, Axel, François Scharffe and Roman Schindlauer (2007). "SPARQL++ for Mapping between RDF Vocabularies". In: *OTM 2007, Part I: Proceedings of the 6th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2007).* Volume 4803. Lecture Notes in Computer Science. Vilamoura, Algarve, Portugal, pages 878–896 (cited on page 97).

Polleres, Axel and Johannes Wallner (2013). "On the Relation B etween SPARQL 1.1 and Answer Set Programming". In: *Journal of Applied Non-Classical Logics (JANCL)* 23 (1-2). Special Issue on Equilibrium Logic and Answer Set Programming, pages 159–212. DOI: 10.1080/11663081.2013.798992 (cited on pages 14, 100).

Prud'hommeaux, Eric and Andy Seaborne, editors (2008). *SPARQL Query Language for RDF.* Available at http://www.w3.org/TR/rdf-sparql-query/. W3C Recommendation (cited on pages 15, 62, 63).

Pyle, Dorian (1999). *Data Preparation for Data Mining.* Morgan Kaufmann (cited on page 3).

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing (cited on page 114).

Ralph Hodgson, Paul J. Keller (2011). *QUDT - Quantities, Units, Dimensions and Data Types in OWL and XML.* URL: http://www.qudt.org/ (cited on pages 71, 93, 107).

Raskin, Robert G. and Michael J. Pan (2005). "Knowledge representation in the semantic web for Earth and environmental terminology (SWEET)". In: *Journal of Computers and Geosciences* 31.9, pages 1119–1125. DOI: 10.1016/j.cageo.2004.12.004 (cited on page 107).

Rijgersberg, Hajo, Mark van Assem and Jan Top (2012). "Ontology of Units of Measure and Related Concepts". In: *Semantic Web - Interoperability, Usability, Applicability (SWJ)* accepted (cited on pages 71, 107).

Rodriguez-Muro, Mariano, Roman Kontchakov and Michael Zakharyaschev (2013). "Ontology-Based Data Access: Ontop of Databases". In: *Proceedings of the 12th International Semantic Web Conference (ISWC'13).* Volume 8218.

Lecture Notes in Computer Science, pages 558–573. DOI: 10.1007/978-3-642-41335-3_35 (cited on pages 25, 49).

Rosati, Riccardo (2012). "Prexto: Query Rewriting under Extensional Constraints in DL-Lite". In: *The Semantic Web: Research and Applications – Proceedings of the 9th ESWC (ESWC'12)*. Volume 7295. Lecture Notes in Computer Science, pages 360–374. DOI: 10.1007/978-3-642-30284-8_31 (cited on page 71).

Rosati, Riccardo and Alessandro Almatelli (2010). "Improving Query Answering over DL-Lite Ontologies". In: *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR'10)*, pages 290–300 (cited on page 71).

Roweis, Sam T. (1997). "EM Algorithms for PCA and SPCA". In: *Advances in Neural Information Processing Systems 10 (NIPS 1997)*, pages 626–632 (cited on pages 111, 112, 128).

Santos, Henrique, Victor Dantas, Vasco Furtado, Paulo Pinheiro and Deborah L. McGuinness (2017). "From Data to City Indicators: A Knowledge Graph for Supporting Automatic Generation of Dashboards". In: *ESWC 2017 Proceedings, Part II*, pages 94–108 (cited on page 125).

Schmidt, Michael, Michael Meier and Georg Lausen (2010). "Foundations of SPARQL Query Optimization". In: *Proceedings of the 13th International Conference on Database Theory (ICDT'10)*, pages 4–33. DOI: 10.1145/1804669.1804675 (cited on pages 42, 71).

Schutt, Rachel and Cathy O'Neil (2013). *Doing Data Science: Straight Talk from the Frontline*. O'Reilly Media, Inc. (cited on page 3).

Siegel, Michael, Edward Sciore and Arnon Rosenthal (1994). "Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems". In: *ACM Transactions on Database Systems (TODS)* 19.2 (cited on page 125).

Sirin, Evren, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur and Yarden Katz (2007). "Pellet: A practical OWL-DL reasoner". In: *Journal of Web Semantics* 5.2, pages 51–53 (cited on page 68).

Stadtmüller, Steffen, Sebastian Speiser, Andreas Harth and Rudi Studer (2013). "Data-Fu : A Language and an Interpreter for Interaction with Read / Write Linked Data". In: *Proceedings of the 22nd International Conference on World Wide Web (WWW'13)* (cited on pages 77, 90).

Switzer, Fred S. and Philip L. Roth (2008). "Coping with Missing Data". In: *Handbook of Research Methods in Industrial and Organizational Psychology*. Blackwell Publishing Ltd, pages 310–323 (cited on page 128).

The Economist Intelligence Unit (2012). *The Green City Index: A summary of the Green City Index research series*. Siemens AG. URL: https://www.siemens.com/entry/cc/features/greencityindex_international/all/en/pdf/gci_report_summary.pdf (visited on 12/10/2017) (cited on pages 4, 68, 75).

— (2017). *Global Livability Ranking 2016*. URL: https://www.eiu.com/liveability2016 (visited on 14/02/2017) (cited on page 4).

United Nations (2015a). *Transforming our world: The 2030 Agenda for Sustainable Development*. General Assembly, A/RES/70/1. URL: https://sustainabledevelopment.un.org/post2015/transformingourworld/publication (visited on 12/10/2017) (cited on page 4).

— (2015b). *World Urbanization Prospects: The 2014 Revision, (ST/ESA/SER.A/366). Department of Economic and Social Affairs*. United Nations (cited on page 4).

Vandenbussche, Pierre-Yves, Ghislain A Atemezing, María Poveda-Villalón and Bernard Vatant (2017). "Linked Open Vocabularies (LOV): a Gateway to Reusable Semantic Vocabularies on the Web". In: *Semantic Web Journal* 8.3, pages 437–452. DOI: 10.3233/SW-160213 (cited on page 19).

Vassiliadis, Panos and Timos Sellis (1999). "A Survey of Logical Models for OLAP Databases". In: *ACM SIGMOD Record* 28.4 (cited on page 126).

Vrandečić, Denny, Christoph Lange, Michael Hausenblas, Jie Bao and Li Ding (2010). "Semantics of Governmental Statistics Data". In: *Web Science Conference (WebSci)* (cited on pages 127, 128).

Witten, Ian H. and Eibe Frank (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. 3rd edition. Morgan Kaufmann Publishers Inc. (cited on page 110).

Wood, David, Marsha Zaidman, Luke Ruth and Michael Hausenblas (2013). *Linked Data: Structured Data on the Web*. Manning (cited on page 21).

OWL Working Group, W3C (2009). *OWL 2 Web Ontology Language: Document Overview*. Available at http://www.w3.org/TR/owl2-overview/. W3C Recommendation (cited on pages 15, 19).

Zhou, Yujiao, Bernardo Cuenca Grau, Yavor Nenov, Mark Kaminski and Ian Horrocks (2015). "PAGOdA: Pay-As-You-Go Ontology Query Answering Using a Datalog Reasoner". In: *Journal of Artificial Intelligence Research* 54, pages 309–367. DOI: 10.1613/jair.4757 (cited on page 49).

Zimmermann, Antoine, Nuno Lopes, Axel Polleres and Umberto Straccia (2012). "A General Framework for Representing, Reasoning and Querying with Annotated Semantic Web Data". In: *JWS* 12, pages 72–95 (cited on page 70).

# Curriculum Vitae

## Education

| | |
|---|---|
| 03/2012–11/2017 | PhD Computer Science<br>Vienna University of Technology, Austria<br>Thesis: "Complementary Methods for the Enrichment of Linked Data" |
| 03/2007–11/2010 | Master of Science Information & Knowledge Management<br>Vienna University of Technology, Austria<br>Thesis: "Implementation and Optimisation of Queries in XSPARQL" |
| 03/2003–03/2007 | Bachelor of Science Software & Information Engineering<br>Vienna University of Technology, Austria |

## Experience Industry and Academia

| | |
|---|---|
| Since 03/2012 | *Siemens AG Österreich, Corporate Technology, Austria*<br>Research Scientist at Corporate Technology, Business Analytics and Monitoring<br>Topics: Semantic Web Technologies |
| 06/2014–09/2015 | *Institute for Information Business at Vienna University of Economics, Austria*<br>Project Assistant at Data and Knowledge Engineering Group<br>Topics: Semantic Web Technologies, Description Logics, Databases |
| 10/2012–05/2014 | *Institute of Information Systems, Vienna University of Technology, Austria*<br>Project Assistant at Database and Artificial Intelligence Group<br>Topics: Semantic Web Technologies, Description Logics, Databases |
| 12/2010–02/2012 | *Digital Enterprise Research Institute at National University of Ireland, Galway*<br>PhD Researcher at Unit for Querying and Reasoning<br>Topics: XSPARQL specification, implementation, and optimization |
| 05/2009–08/2009 | *Digital Enterprise Research Institute at National University of Ireland, Galway*<br>Research Intern for Master Thesis, XSPARQL Implementation in Java |
| Summer 2008 | *BOC Information Systems, Vienna, Austria*<br>Software Design and Development, Browser based test automation |
| Summer 2007 | *Julius Blum GmbH, Höchst, Austria*<br>Software Design and Development, Data Migration with XSLT |
| Summer 2005 | *OMICRON electronics GmbH, Klaus in Vorarlberg, Austria*<br>Software Design and Development, Parser |

| | |
|---|---|
| Summer 2004 | *Weber Informatik Lösungen, Göfis, Austria*<br>Software Development, End user systems |