Poznan University of Technology
Faculty of Computing Science
Institute of Computing Science

PhD thesis

METHODS FOR AUTOMATIC ENRICHMENT OF ONTOLOGIES FROM LINKED DATA

mgr inż. Jedrzej Potoniec

Supervisor
prof. dr hab. inż. Joanna Józefowska
Supporting Supervisor
dr inż. Agnieszka Ławrynowicz

Poznań, 2017

Abstract

In this thesis, we address the problem of helping an ontology engineer to add new axioms to an ontology. We assume that the ontology contains some vocabulary and that there exists a Linked Data dataset using the vocabulary. The goal is to generate candidate axioms that can be reviewed and added to the ontology. We propose three inductive methods, that use only SPARQL Query Language to access the dataset. In particular, this enables using a public SPARQL endpoint instead of downloading the whole dataset.

The first method generates a large set of candidate axioms and uses integer linear programming to select the most interesting subset of the axioms, that is then presented to the user. It is suitable for splitting an existing class into new classes, each with an axiomatic definition. We provide an implementation of the proposed method as a part of *RMonto*, a plugin for a data mining tool *RapidMiner*.

The second method extends an attribute exploration algorithm from Formal Concept Analysis. The algorithm generates possible subsumption axioms, that neither follow from the ontology nor contradict it, and asks the user to decide whether they are true or false. The proposed extension introduces a classifier that learns from the decisions made by the user and, once trained, it tries to replace the user. The classifier not only provides an answer, but it also assesses its confidence. If the confidence is high enough, the provided answer is used and the question is not posed to the user. Otherwise, the user is asked to provide a correct answer and the classifier is retrained. We provide an implementation in a form of a standalone Java application based on *Weka* machine learning library.

The third method, called Swift Linked Data Miner (SLDM) is a pattern mining approach. The target language of patterns covers the whole language of OWL 2 EL class expressions. The intended usage of patterns is to provide partial definition of a class chosen by the user. We discuss how to retrieve only relevant data from the dataset and to effectively organize them in memory in a form of a three-level index. We develop a set of algorithms processing the index to discover the patterns. We report on the results of a crowdsourcing experiment, that showed that the axioms generated this way are correct for a broad range of parameter settings. We also present computational properties of the algorithm and provide a plugin for *Protégé* implementing SLDM.

Streszczenie

W niniejszej rozprawie rozważa się problem wspomagania inżyniera ontologii w dodawaniu nowych aksjomatów do ontologii. Zakłada się, że ontologia zawiera pewne słownictwo oraz istnienie zbioru Powiązanych Danych korzystającego z tego słownictwa. Celem jest wygenerowanie kandydujących aksjomatów, które mogą zostać przejrzane i dodane do ontologii. Proponowane są trzy metody indukcyjne, które używają wyłącznie języka zapytań SPARQL w celu dostępu do zbioru danych. W szczególności, umożliwia to używanie publicznej końcówki SPARQL zamiast pobierania całego zbioru danych.

Pierwsza metoda generuje duży zbiór kandydujących aksjomatów i używa liniowego programowania całkowitoliczbowego w celu wybrania najbardziej interesującego podzbioru kandydatów, który jest następnie prezentowany użytkownikowi. Jest ona odpowiednia do podziału istniejącej klasy na nowe klasy, z których każda będzie miała aksjomatyczną definicję. Dostarczona jest implementacja przedstawionej metody jako element *RMonto*, wtyczki do narzędzia do eksploracji danych *RapidMiner*.

Druga metoda rozszerza algorytm eksploracji atrybutów pochodzący z Formalnej Analizy Pojęć. Algorytm generuje możliwe aksjomaty zawierania, które ani nie wynikają z ontologii, ani nie stoją w sprzeczności do niej i prosi użytkownika o określenie czy są one prawdziwe czy fałszywe. Zaproponowane rozszerzenie wprowadza klasyfikator, który uczy się z decyzji użytkownika i, po wytrenowaniu, próbuje zastąpić użytkownika. Klasyfikator dostarcza nie tylko odpowiedź, ale również ocenia swoją pewność. Jeżeli jest ona wystarczająco wysoka, wykorzystywana jest dostarczona odpowiedź i pytanie nie jest przedstawiane użytkownikowi. W przeciwnym wypadku użytkownik jest proszony o podanie właściwej odpowiedzi i klasyfikator jest uczony ponownie. Dostarczona jest implementacja w formie samodzielnej aplikacji Java opartej na bibliotece do uczenia maszynowego *Weka*.

Trzecia metoda, nazwana Swift Linked Data Miner (SLDM) jest oparta o odkrywanie wzorców. Język wzorców pokrywa cały język wyrażeń klasowych OWL 2 EL. Dyskutuje się jak pobierać wyłącznie istotne dane ze zbioru danych i jak efektywnie reprezentować je w pamięci w formie trzypoziomowego indeksu. Przedstawiona jest konstrukcja zbioru algorytmów przetwarzających indeks w celu odkrycia wzorców. Zamierzone użycie wzorców to dostarczenie cześciowych definicji klasy wskazanej przez użytkownika. Przedstawione są wyniki eksperymentu crowdsourcingowego, które pokazuję, że aksjomaty wygenerowane w ten sposób są poprawne w szerokim zakresie ustawień parametrów. Przedstawione są również własności obliczeniowe algorytmu i wtyczka do *Protégé* implementująca SLDM.

Contents

List of symbols

$\mathcal{I}$	interpretation
$\Delta^{\mathcal{I}}$	interpretation domain
$\cdot^{\mathcal{I}}$	interpretation function
A	concept name
$\top$	top concept
$\bot$	bottom concept
$\{a\}$	nominal
$\sqcap$	intersection
$\exists$	existential restriction
$\sqsubseteq$	subsumption
$\circ$	composition of relations
dom	domain of a relation
ran	range of a relation
$\equiv$	equivalence
$\mathcal{T}$	TBox
$\mathcal{A}$	ABox
$\mathcal{O}$	ontology
C, D	concept, class expression
D, E	data range
l	literal
P, Q	property
	Mathematical modelling for ontology learning
$\mathcal{P}, \mathbb{P}$	set of patterns
P_j	pattern
$\mathbb{I}$	set of IRIs
I_i	IRI
$\mathbb{X}$	selected subset of patterns
$\mathbb{R}(P)$	results of the query P
λ	minimal number of selected patterns
A	binary matrix representing a relationship between $\mathbb{P}$ and $\mathbb{I}$
x_j, y_i, z_j	variables in a mathematical model
M	large number
	Formal Concept Analysis supported by machine learning
G	complete set of objects
M	complete set of attributes
I	*has attribute* relation
$\cdot^{I}$	derivation operator

$\prec$	partial order relation between concepts
$\mathbb{K}$	set of all concepts
J	incomplete *has attribute* relation
A	set of positive attributes in a partial object description
S	set of negative attributes in a partial object description
$\rightarrow$	attribute implication
$\mathcal{L}$	set of attribute implications
$\mathcal{L}(P)$	implicational closure of P w.r.t. $\mathcal{L}$
P, Q	set of attributes
$\mathcal{J}$	implication base
$\mathcal{K}$	incomplete context
$\overline{\mathcal{K}}$	formal context
$\mathbf{x}$	vector of features of an example
y	label of an example
θ_r	rejection probability threshold
θ_a	acceptance probability threshold
μ	mapping from an attribute to a SPARQL BGP
L, R	attribute in an implication
M	set of class expressions
$<$	lectic order

Swift Linked Data Miner

$\mathcal{I}$	set of IRIs
$\mathcal{T}$	set of triples
$\mathcal{L}$	set of literals
μ	matching function
$\mathbb{G}$	RDF graph
S	proof set
σ	support
w	weighting function
$\mathfrak{l}$	length of a pattern
$\mathfrak{I}$	three-level index
θ_σ	minimal support threshold
$\theta_\mathfrak{l}$	maximal length of a pattern

Chapter 1

Introduction

1.1 Motivation

The *Semantic Web* is an idea originally described in *Scientific American* by the creator of the World Wide Web as we know today, sir Tim Berners-Lee, along with James Hendler and Ora Lassila [8]. The core idea of the Semantic Web is the requirement to enrich web resources with an explicit meaning, based on a formally defined *vocabulary* with shared understanding. This is opposed to the Web as we usually see it, where the semantics is implicit: from tables of various formats and headings, through blocks of text with ambiguous semantics of natural language, to pictures and emoticons with subjective semantics. On top of that, the tags of *Hypertext Markup Language* (HTML) are frequently abused to achieve desired visual effect (taking into account what little semantics they offer) rather than they provide useful information. For example, a table is frequently used as a way of positioning text on a web page rather than to present tabular data. All these properties make the Web a unfriendly place for an autonomous agent incapable of perceiving the world as humans do.

As these issues could not be addressed within existing technologies, a separate set of standards has been created for the Semantic Web. The simplest ones are microformats and *Resource Description Framework in Attributes* (RDFa), designed to enrich existing HTML web pages throughout additional, non-visual annotations to express the semantics [47, 9]. Then, there is *Resource Description Framework* (RDF), a standard defining a machine-understandable representation of webpages, concerned only with precise representation of information, not with its presentation [101]. RDF is supported by *SPARQL Query Language* (in short: *SPARQL*) and *SPARQL Protocol*, a standardized way to query and retrieve RDF content from the Web [33]. On top of these, Web Ontology Language (*OWL*) has been created to formalize and share semantics of the vocabulary used in RDF and to deliver decidable algorithms for deductive inference rooted in logic.

From widespread adoption of these standards, especially RDF and SPARQL, a new phenomena emerged: an area of the Web called *Linked Data* designed especially for machines [7]. An open subset of Linked Data, called *Linked Open Data*, increased almost 100 times during the last 10 years, from 12 in 2007 to 1,139 in 2017 [80]. As expected, none of these datasets are random, rather they all originate in some process of collection, transformation and maintenance. Unfortunately, it is the case that the most of the semantics for a dataset is available only in a textual description, be it a web site, a scientific paper or a set of annotations in the dataset. This fact clearly contradicts the very idea of the Semantic Web, namely providing clear semantics of the data in a fashion available both for humans and machines alike.

As things stand now, an entity publishing a Linked Open Data dataset has little incentive to

develop an ontology for the data. The publisher is already familiar with the data and does not need the ontology, while providing a rough textual description for the convenience of other human users is enough. Adding the fact that ontology engineering is a complex area requiring both expert knowledge of the data and of the modeling principles of ontologies, it is not surprising that the ontologies for Linked Open Data datasets are weak and shallow. This does not inconvenience human users too much, but makes the Semantic Web standards unsuitable for development of the envisioned autonomous agents using web-scale, logic-based reasoning.

1.2 Research problem

Creation of an ontology for a given Linked Data dataset would be much simpler if it could be partially automated. We thus consider the following setup: given is an existing Linked Data dataset, i.e. an *RDF graph* accessible using SPARQL Query Language. The dataset is described by an ontology, that may be a very shallow one, consisting of only the vocabulary or be a heavily axiomatized ontology rich in axioms, or be at any stage in between. During the process of creation of the dataset, lots of knowledge was hidden in there and we consider the following problem: how to help an ontology engineer to discover this hidden knowledge and use it to extend the ontology with additional, formal semantics. Ultimately, we do not want to replace the ontology engineer, but rather to provide help in the task at hand: suggest new axioms, that may be worth adding to the ontology.

1.3 Aim and scope of the work

In order to build a common ground in Chapter 2 we introduce the formalisms and technologies used in the Semantic Web. Section 2.1 introduces Resource Description Framework (RDF), a low-level conceptual model used to represent data in the Semantic Web. The next section introduces SPARQL, a graph-matching query language suitable for querying RDF graphs. We then formalize the notion of Linked Data and discuss briefly their properties. Section 2.4 describes *Description Logics*, a family of decidable subsets of first-order logic. In the final section, we discuss OWL 2 Web Ontology Language, a language built on top of RDF, with the semantics rooted in the Description Logics. OWL 2 is the most common way to express semantics in the Semantic Web.

Chapter 3 presents the preexisting work of other authors in the related areas. In the beginning, we describe the area of learning ontologies by processing natural language documents. We then discuss briefly techniques of learning ontologies during an interactive dialogue with the user. Further on, we describe concept learning, the approach applying the ideas of Inductive Logic Programming to the Description Logics and Web Ontology Language. In Section 3.4, we provide a description of the methods for ontology learning from a locally available RDF graph. We conclude by discussion of the techniques for learning ontologies directly from Linked Data.

Beginning from Chapter 4, we present our contribution to the field, starting with a method for detection of emerging subclasses, that are not yet named, but are supported by the data. The method employs Fr-ONT-Qu, an algorithm for detection of *frequent patterns* in RDF graphs, to discover possible subclasses, which we describe in Section 4.1. *Binary linear programming* is then used to select the most promising candidates from the set of frequent patterns. The way to generate the linear program is described in Section 4.2. We conclude the chapter by presenting a *RapidMiner* workflow implementing the presented solution.

In Chapter 5, we discuss a way to extend the *attribute exploration* algorithm used to complete ontologies with a machine learning algorithm. The aim of the machine learning algorithm is to

partially replace the user of the algorithm in the tedious process of answering queries generated by the algorithm. Its input is based on the information available in the Linked Data and it learns by observing the behaviour of the user. We begin by introducing Formal Concept Analysis in Section 5.1 and attribute exploration algorithm in Section 5.4. We then explain in Section 5.5 how to apply the algorithm to formal ontologies. Section 5.6 formalizes the *classification task* present in attribute exploration algorithm and discusses ways to address it with a machine learning *classification algorithm*. In the final section, we present a software tool, which implements the idea.

Chapter 6 presents Swift Linked Data Miner, a completely new algorithm for mining an ontology from a Linked Data dataset. The basic idea of the algorithm originated in FP-Growth, an algorithm for frequent itemset mining, with design based on a data structure specialized for an efficient enumeration of the frequent itemsets [32, 31]. In Section 6.1 and Section 6.2, we discuss a way to efficiently communicate with a *SPARQL endpoint* available in the Web. In Section 6.3, we describe a smart data structure to organize the retrieved data. Section 6.4 formalizes the task solved by SLDM as a *frequent pattern mining* task and introduces the relevant measures. The next section describes a set of algorithms to efficiently explore the data structure. We then present a plugin for *Protégé* that can be used in a daily workflow of an ontology engineer. Section 6.7 brings a description of a crowdsourcing experiment aimed at validation of the correctness of axioms mined by SLDM on *DBpedia*. In the last section, we discuss computational properties of SLDM, such as required amount of time, memory consumption and variability w.r.t various parameters settings.

We conclude in Chapter 7, where we summarizes the results discussing the presented methods and outline the possible future directions of research in the area of ontology mining from Linked Data.

Chapter 2

The Semantic Web

2.1 Resource Description Framework

Resource Description Framework (RDF) is a framework providing a graph-based representation for web resources [101]. The purpose of RDF is to describe some *universe of discourse*. The basic concept of RDF is an *RDF triple*, which consists of a *subject*, a *predicate* and an *object*. Usually, it is treated as a simple sentence stating that the subject is related to the object through a relation denoted by the predicate. The subject and the object denote resources in the described universe, while the predicate denotes a binary relation between resources of the universe. A set of RDF triples is called an RDF graph, the subjects and objects of the triples of the graph collectively form the set of *nodes*. Every *part of a triple*, i.e. a subject, a predicate, an object, must be one of *RDF terms*: either an *Internationalized Resource Identifier* (IRI), a *blank node* or a *literal*. In addition to that, the subject of a triple must be either an *IRI* or a blank node and the predicate must be an IRI.

An IRI is an resource identifier following the syntax defined in RFC3987 [21]. The scope of IRI meaning is global, that is the same IRI in two separate RDF graphs should refer to the same thing in the described universe of discourse. It is worth pointing out, that any valid URL (*Uniform Resource Locator*) is also a valid IRI, and thus it is common to use web addresses as IRIs in an RDF graph.

Thorough this work, we use Turtle syntax to represent RDF [18]. In particular, we denote an IRI by placing it in angle brackets, e.g. `<http://dbpedia.org/ontology/author>`. In case of a document containing multiple IRIs starting with a common string, it is convenient to use prefixes. The common string is then frequently referred to as a *namespace* and a prefix is a short replacement for the namespace. For example, the namespace `http://dbpedia.org/ontology/` is frequently denoted by the prefix `dbo:` and in such a case `dbo:author` is a shorter form of `<http://dbpedia.org/ontology/author>`. In this work, we use a set of common prefixes presented in Table 2.1. We also use `a` to denote the IRI `rdf:type` used as a predicate, e.g. the triple `dbr:PKP_class_EU07 a dbo:Locomotive` is equivalent to the triple `dbr:PKP_class_EU07 rdf:type dbo:Locomotive`.

A blank node is also a resource identifier, but with the scope of a single RDF graph. The same blank node, but in two separate RDF graphs, may denote two different resources in the universe of discourse. In this work, we denote an unspecified blank node by `[]` and a specific blank node by an IRI with the special prefix `_:`, e.g. `_:bnode1`.

A literal denotes a datum and consists of a concrete value, such as a string or an integer, and its datatype. The datatype is also an IRI and is necessary to distinguish between an integer *1* and a

Table 2.1: A set of common prefixes used in this work.

prefix	namespace
:	http://example.com/ex#
dbo:	http://dbpedia.org/ontology/
dbr:	http://dbpedia.org/resource/
dbp:	http://dbpedia.org/property/
xsd:	http://www.w3.org/2001/XMLSchema#
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
owl:	http://www.w3.org/2002/07/owl#

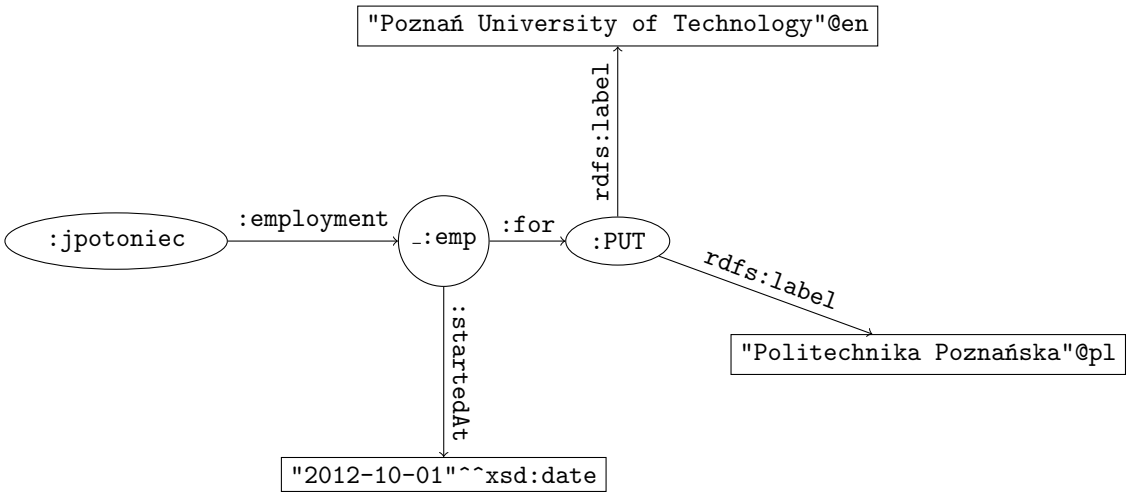


Figure 2.1: A sample RDF graph G_1, describing an employment relationship between `:jpotoniec` and `:PUT`, which started at 2012-10-01. `:PUT` has assigned textual labels in two languages, Polish and English.

string *"1"*. If the datatype is not specified, the default value `xsd:string` is assumed. Optionally, a literal may contain a language tag, which means that the literal contains a plain text expressed in the language specified by the tag. In such a case, the datatype is assumed to be `rdf:PlainLiteral`. In this work, we denote literals by placing them in double quotes. A datatype or a language tag are specified after a literal, separated from it by, respectively, `^^` or `@`. For example, `"1"` is a literal with the default datatype `xsd:string` and `"1"^^xsd:string` is the same literal with the datatype explicitly specified, while `"1"^^xsd:integer` is a literal with the same textual representation, but datatype `xsd:integer`. Finally, `"Hello"@en` is a literal with the language tag `en` denoting that the text in the literal is in English. As the datatype of the literal is unspecified, the default datatype `rdf:PlainLiteral` is assumed.

A sample RDF graph is presented in Figure 2.1. The graph consists of five triples and contains six nodes: two IRIs (`:jpotoniec`, `:PUT`), a single blank node (`_:emp`), two literals with a language tag and a single literal with datatype `xsd:date`. There are also four IRIs denoting predicates (`:employment`, `:startedAt`, `:for`, `rdfs:label`).

Serialization of the graph in Turtle syntax is presented in Table 2.2. A triple in Turtle syntax is written in the following order: the subject, the predicate, the object. Each part of a triple is separated by spaces and the triple ends with a dot, as in the first row of the serialization. If consecutive triples share the subject, it can be omitted from the second triple onward and the triples are separated with semicolons instead of dots, see rows 2 and 3 of the serialization. If the consecutive triples share the subject and the predicate, they can be both omitted from the second

Table 2.2: A Turtle serialization of the graph G_1 presented in Figure 2.1.

```
:jpotoniec  :employment  _:emp                                  .
_:emp       :startedAt   "2012-10-01"^^xsd:date                 ;
            :for         :PUT                                   .
:PUT        rdfs:label   "Politechnika Poznańska"@pl            ,
                         "Poznań University of Technology"@en   .
```

triple onward and the triples are separated with colons instead with dots, as in the last two rows of the serialization.

2.2 SPARQL Query Language

Any kind of data representation requires a query language and RDF is no exception. Its graph-oriented nature requires a query language able to express graph patterns. While many were proposed, e.g. RQL [45], SeRQL [13] or TRIPLE [82] to name just a few, SPARQL [33] is the W3C standard and probably the most popular query language for RDF in the Semantic Web community. Following the specification of SPARQL [33], we present the parts of SPARQL relevant to this work.

The set of terms allowed in SPARQL queries consists of RDF terms and variables. A *variable* consists of an arbitrary name preceded by `?` or `$`, e.g. `?var`. A blank node in a SPARQL query also serves as a variable, not as a reference to a specific blank node in the queried graph. It is consistent with a graph scope of blank nodes in RDF, with the query representing a graph separate from the queried graph.

The core parts of SPARQL are triple patterns and graph patterns. A *triple pattern* is an RDF triple with an arbitrary number of parts replaced by variables. A *solution mapping* is a mapping from variables to RDF terms. An *instance mapping* is a mapping from blank nodes to RDF terms. Mapping μ is a *solution* for a triple pattern w.r.t. a graph if there exists a solution mapping σ and an instance mapping such that the triple pattern after applying the solution mapping and the instance mapping yields a triple, which is present in the graph. Mapping μ is then σ limited to the variables occurring in the triple pattern.

A *basic graph pattern* (BGP) is a set of triple patterns and μ is a solution for BGP if and only if it is a solution for every triple pattern in the BGP. BGPs in SPARQL are expressed using the Turtle syntax.

Recall the graph G_1 from Figure 2.1 and Table 2.2. For example, a triple pattern to obtain all labels for `:PUT` is `:PUT rdfs:label ?label`, where `?label` is the variable to be mapped to the actual labels. There exist two solutions for this triple pattern w.r.t. the graph, one mapping `?label` to `"Politechnika Poznańska"@pl` and the second mapping `?label` to `"Poznań University of Technology"@en`. To obtain all resources being subjects of the employment relationship, one may use the following pattern: `?resource :employment []`. In the solution w.r.t. to the graph, `?resource` is mapped to `:jpotoniec` and the blank node `[]` to `_:emp`. The blank nodes in a query are different blank nodes than these used in the graph, and thus the pattern `_:emp rdfs:label ?label` has two solutions w.r.t. the graph G_1. Finally, to obtain all the employees and labels of their employers, one may use the following *BGP*

```
?employee :employment _:emp .
_:emp rdfs:label ?employer .
```

Table 2.3: A sample query selecting all employees and labels of their employers.

```
SELECT ?employee ?employer
WHERE
{
    ?employee :employment _:emp .
    _:emp rdfs:label ?employer .
}
```

Table 2.4: The result of the query from Table 2.3 w.r.t. the graph presented in Figure 2.1.

?employee	?employer
:jpotoniec	"Politechnika Poznańska"@pl
:jpotoniec	"Poznań University of Technology"@en

The solutions can be further limited using filter expressions `FILTER(expression)`. If a filter expression in a query evaluates to false or raises an error for a particular solution, the solution is dropped. SPARQL offers a wide selection of functions and operators in filter expressions. They enable checking if a value is an IRI, a blank node or a literal using functions `isIRI`, `isBlank`, `isLiteral`, respectively. The syntax for the filter expressions is borrowed from C programming language, with `<=`, `!=` etc. representing the comparison operators and `!`, `&&` and `||` for negation, conjunction and disjunction, respectively. For example, in order to check if an RDF term mapped to a variable `?x` is an IRI or a literal greater than 5, one may use the following expression: `FILTER(isIRI(?x) || ?x > "5"^^xsd:integer)`.

A minimal SPARQL query consists of a head, specifying what to do with the solutions, and a body with a BGP. There are four *verbs* allowed in the head: SELECT, ASK, CONSTRUCT and DESCRIBE. SELECT is suitable for obtaining a set of solutions for a given pattern, while ASK provides only a binary information if there exists a solution for the pattern. CONSTRUCT enables constructing an RDF graph on the basis of the solutions for the BGP, while semantics of DESCRIBE is not specified and its behaviour varies between implementations. Throughout this work we use only SELECT queries, and thus we describe below their syntax in a more detailed manner, while omitting the other forms of queries.

The basic form of a SPARQL SELECT query is `SELECT head WHERE { pattern }`, where `pattern` is a basic graph pattern optionally with filter expressions, while `head` is a list of variables present in the pattern and defines a projection over the solutions. A sample SELECT query to obtain all employees and the labels of their employers is presented in Table 2.3. A result of a SELECT query is a set of solutions. The result of the sample query over the RDF graph from Figure 2.1 is presented in Table 2.4.

While SPARQL Query Language may be used to query an RDF graph embedded in an application, it is the most useful when coupled with SPARQL Protocol, which defines how to pose a SPARQL query over a computer network [99]. The protocol uses HTTP (Hypertext Transfer Protocol [24]) to transfer the query and its result and XML (Extensible Markup Language [88]) to serialize the results. A database containing an RDF graph is called an *RDF store* and a service able to answer SPARQL queries is called a SPARQL endpoint and it is usually identified by its URL.

2.3 Linked Data

Linked Data is a term coined by sir Tim Berners-Lee in 2006, in order to point out that the Semantic Web is not about putting data on the Web, but rather about interlinking them to form the *Web of Data*, so people and machine exploring a piece of data can discover other, relevant pieces [7]. Linked Data should follow four basic principles:

1. Use URIs as identifiers.
2. Use HTTP URIs, that are valid HTTP address.
3. Provide useful information in the document pointed by an URI, preferably in RDF.
4. Include links to other resources.

Soon after that, at the beginning of 2007, it became apparent that it is also required for the data to be available for reuse and thus a term *Linked Open Data* (LOD) appeared [10]. Basically, these are Linked Data with an open licence, i.e. such that allows reusing them for free. A five-star system for rating Linked Data was proposed [7]:

1 star any format, but an open licence;

2 stars 1 star plus machine-readable structured data;

3 stars 2 stars plus an open file format;

4 stars 3 stars plus usage of URIs, RDF and SPARQL;

5 stars 4 stars plus outgoing links to other Linked Open Data datasets.

The methods proposed in this work do not depend on openness of data, but they do depend on the data being available using a SPARQL endpoint. In case of Linked Open Data, it requires them to be at least 4 stars LOD.

Figure 2.2 depicts a subset of the *Linked Open Data cloud* available in the Web as of April 2014 [80]. It shows 570 datasets of various sizes, each having at least 1000 triples. They come from different domains of interest, such as governmental data, linguistics or life sciences. The edges in the diagram represent interlinking between the connected datasets.

The center of the diagram is occupied by DBpedia, a semantic version of *Wikipedia* [2, 11, 52]. The transformation from Wiki markup language to RDF is performed by a set of extractors that concentrate mostly on Wikipedia infoboxes. DBpedia plays a major role in the Linked Open Data cloud as it provides information on a very wide range of topics and so it is relatively easy for publishers of data from various domains of interest to provide links to DBpedia, thus following one of the Linked Data principles. As DBpedia originates in an encyclopedia, it covers a lot of topics understandable for general population, like books, movies or famous people. Along with its size, this makes DBpedia a popular and useful dataset in validating and benchmarking algorithms and applications designed for the Semantic Web.

2.4 Description Logics

Description Logics (DLs) are a knowledge representation formalism suitable for describing the universe of discourse by first collecting a set of relevant concepts for the description, and then using them to specify properties of objects in the universe of discourse [4]. DLs are equipped

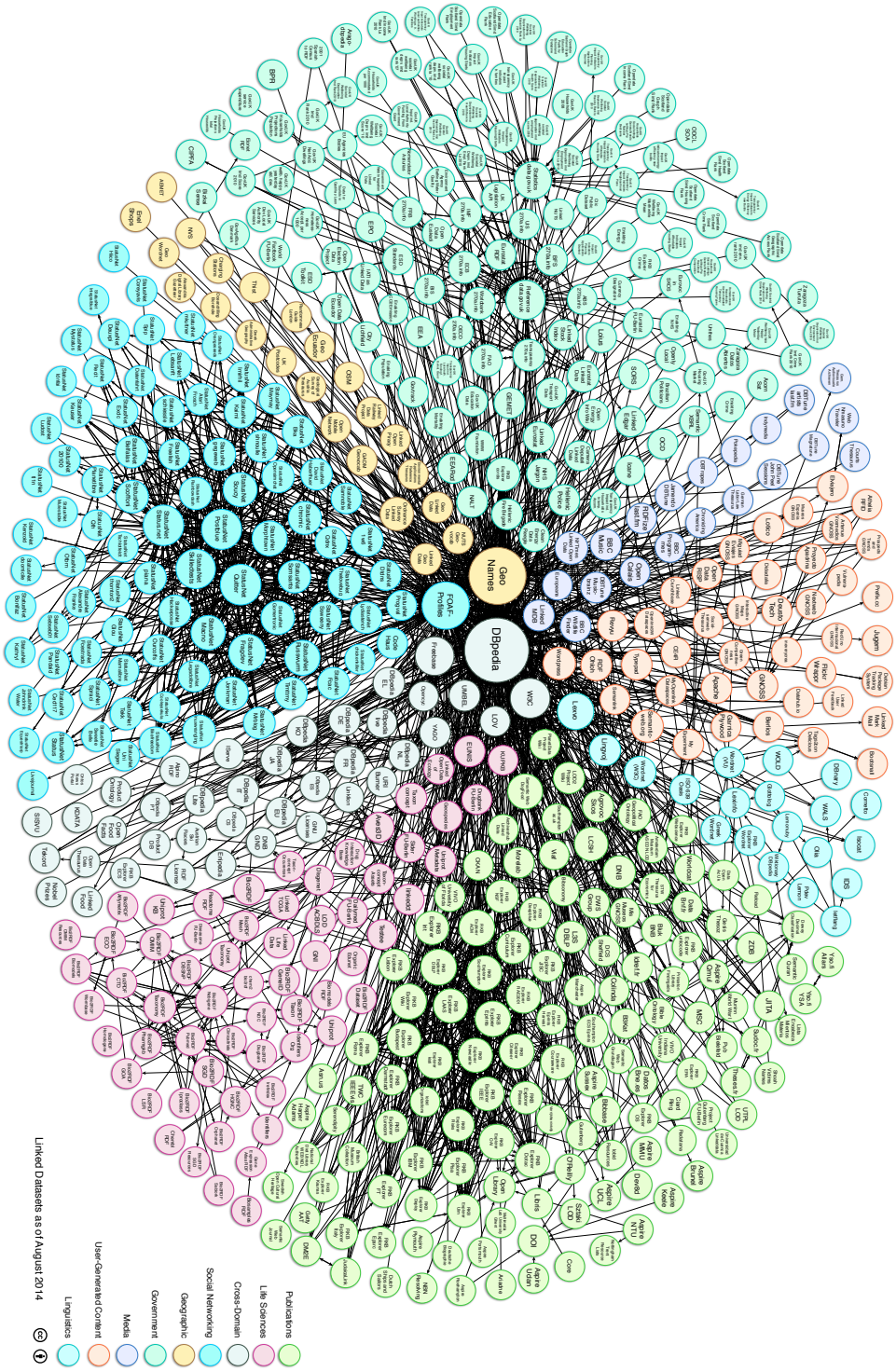


Figure 2.2: *The Linking Open Data cloud diagram*, showing some of the Linked Open Data datasets available in April 2014 [80]

with a formal semantics, rooted in the first-order logic and decidable reasoning procedures, which guarantee obtaining the correct answer to a posed query in finite time. While there are many DLs, we are especially interested $\mathcal{EL}^{++}$ [3, 6], which serve as a base for *OWL 2 EL* [62], being a *profile* of OWL 2 Web Ontology Language and described in the next section.

Names occurring in a DL *knowledge base* can be divided into three disjoint sets: a set of *individual names*, a set of *role names* and a set of *concept names*. Their semantics is defined using an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where the *interpretation domain* $\Delta^{\mathcal{I}}$ is a non-empty set of individuals, i.e. the objects of the universe of discourse, while the *interpretation function* $\cdot^{\mathcal{I}}$ is a function mapping every individual name a to an individual $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, every role name r to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and every concept name A to a set of individuals $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$.

A concept C is any expression following the grammar defined below:

$C \leftarrow A\|$	concept name
$\top\|$	top concept
$\bot\|$	bottom concept
$\{a\}\|$	nominal
$C \sqcap D\|$	intersection
$\exists r.C$	existential restriction

The *top concept* $\top$ is the most general concept, and thus $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$, while the *bottom concept* $\bot$ is the least general one, i.e. $\bot^{\mathcal{I}} = \emptyset$. $\{a\}$ is a special concept containing, by its construction, only a single individual $\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$. $C \sqcap D$ is an intersection of two concepts, and thus $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$. Finally, the semantics of $\exists r.C$ is given by the following formula:

$$(\exists r.C)^{\mathcal{I}} = \left\{x \in \Delta^{\mathcal{I}} \colon \exists y \ \left((x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\right)\right\} \tag{2.1}$$

A *general concept inclusion* (GCI) $C \sqsubseteq D$ and a role inclusion $r_1 \circ r_2 \circ \ldots \circ r_n \sqsubseteq r$ are interpreted as subset relations, respectively, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and $r_1^{\mathcal{I}} \circ r_2^{\mathcal{I}} \circ \ldots \circ r_n^{\mathcal{I}} \subseteq r^{\mathcal{I}}$. $\circ$ operator denotes a composition of relations, i.e.

$$r_1^{\mathcal{I}} \circ r_2^{\mathcal{I}} = \left\{(x,z) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \colon \exists y \ \left((x,y) \in r_1^{\mathcal{I}} \wedge (y,z) \in r_2^{\mathcal{I}}\right)\right\} \tag{2.2}$$

A *domain restriction* $dom(r) \sqsubseteq C$ and a *range restriction* $ran(r) \sqsubseteq C$ define a set of elligible values for, respectively, the first and the second argument of relation r:

$$[dom(r) \sqsubseteq C]^{\mathcal{I}} = [r^{\mathcal{I}} \subseteq C^{\mathcal{I}} \times \Delta^{\mathcal{I}}] \qquad [ran(r) \sqsubseteq C]^{\mathcal{I}} = [r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C^{\mathcal{I}}] \tag{2.3}$$

In an $\mathcal{EL}^{++}$ knowledge base, the set of all GCIs, role inclusions, domain and range restrictions is referred to as the *TBox* and denoted by $\mathcal{T}$. The *ABox*, denoted by $\mathcal{A}$, contains *concept assertions* $C(a)$ and *role assertions* $r(a,b)$, with the following semantics:

$$[C(a)]^{\mathcal{I}} = [a^{\mathcal{I}} \in C^{\mathcal{I}}] \qquad [r(a,b)]^{\mathcal{I}} = [(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}] \tag{2.4}$$

We introduce a few additional names, that are abbreviations for already defined operators. A concept C is *equivalent to* a concept D, denoted by $C \equiv D$ if, and only if, $C \sqsubseteq D$ and $D \sqsubseteq C$. A concept C is *disjoint with* a concept D if, and only if, $C \sqcap D \sqsubseteq \bot$. A role r is *reflexive*, if every individual is in the relation r with itself. Reflexivity is a special case of composition for $n = 0$.

An interpretation $\mathcal{I}$ is a *model* of the TBox $\mathcal{T}$ (resp. the ABox $\mathcal{A}$) if for each element of $\mathcal{T}$ (resp. $\mathcal{A}$), the semantics of the element is satisfied. An interpretation $\mathcal{I}$ is a model of an $\mathcal{EL}^{++}$ knowledge base $(\mathcal{T}, \mathcal{A})$ if it is a model of the TBox $\mathcal{T}$ and a model of the ABox $\mathcal{A}$.

Following [4], we define a set of standard reasoning tasks w.r.t. to an $\mathcal{EL}^{++}$ knowledge base $(\mathcal{T}, \mathcal{A})$:

satisfiability A concept C is *satisfiable* w.r.t. to $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$.

subsumption A concept C is *subsumed by* a concept D (denoted by $C \sqsubseteq D$) w.r.t. to $\mathcal{T}$ if in every model $\mathcal{I}$ of $\mathcal{T}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

equivalence A concept C is equivalent to a concept D (denoted by $C \equiv D$) w.r.t. to $\mathcal{T}$ if in every model $\mathcal{I}$ of $\mathcal{T}$ $C^{\mathcal{I}} = D^{\mathcal{I}}$.

disjointness A concept C is disjoint with a concept D w.r.t. to $\mathcal{T}$ if in every model $\mathcal{I}$ of the TBox, $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$.

consistency checking The ABox is *consistent* with the TBox if there exists a model of the TBox, that is also a model of the ABox.

entailment (instance checking) An assertion $C(a)$ (resp. $R(a, b)$) is *entailed* by $\mathcal{A}$, if every model of $\mathcal{A}$ also satisfies the assertion.

In [3, 6, 57] the authors provide a tractable algorithm for *subsumption* checking. Baader et al. [4] showed that satisfiability, equivalence and disjointness can be reduced to subsumption by simple syntactic operations. The presence of nominals in $\mathcal{EL}^{++}$ makes it also possible to realize *consistency checking* and *entailment* using a subsumption algorithm [22].

A computer program realizing the reasoning tasks is called a *reasoner*. *Pellet* [85], *HermiT* [63] and *ELK* [46] are state of the art examples of DL reasoners.

2.5 OWL 2 Web Ontology Language

OWL 2 Web Ontology Language (in short: *OWL 2*) is the W3C standard for modelling and exchanging ontologies, i.e. conceptual models, for the Semantic Web. It is equipped with a formally defined semantics rooted in the Description Logics. An OWL 2 ontology can be represented as an RDF graph and thus it is usually serialized using one of the RDF serializations. OWL 2 is a very expressive formalism, with the complexity of reasoning as high as N2EXPTIME-complete [62]. Even though the reasoners are highly optimized nowadays and usually work quite fast, the complexity stops them from being adapted to large problems. Fortunately, OWL 2 is equipped with three profiles: OWL 2 RL, OWL 2 QL and OWL 2 EL. OWL 2 RL is a profile with a rule-based, PTIME-complete reasoning algorithm, while OWL 2 QL has a very limited expresivity, but the reasoning algorithms are LOGSPACE-complete. Finally, OWL 2 EL, rooted in Description Logic $\mathcal{EL}^{++}$, is a profile especially suitable for expressing large ontologies due to the available language constructs on one hand, and the PTIME reasoning complexity on the other hand [62]. As an ontology created semi-automatically can easily become large it is desirable to select the profile taking this into account, and thus we concentrate on OWL 2 EL.

An OWL 2 EL *datatype map* is a 6-tuple $D = (N_{DT}, N_{LS}, N_{FS}, \cdot^{DT}, \cdot^{LS}, \cdot^{FS})$, where:

- N_{DT} is a set of 18 datatypes, listed in Table 2.5. These datatypes, along with `rdfs:Literal`, are the only datatypes supported by OWL 2 EL.
- N_{LS} is a function that assigns each datatype from the set N_{DT} its *lexical space*, that is the set of all valid textual representations of the values of the datatype;
- N_{FS} is a function that assigns every datatype a set of pairs (F, v), where F is a *constraining facet* and v is called a *constraining value* and is an arbitrary value from the *value space* of the datatype.

- $\cdot^{DT}$ is a function that assigns every datatype a set of values called the value space of the datatype.
- $\cdot^{LS}$ is a function that assigns every pair consisting of a lexical value and a datatype, a *data value* from the value space of the datatype.
- $\cdot^{FS}$ is a function that, for every datatype DT, assigns every pair $(F, v) \in N_{FS}(DT)$ a subset of the value space DT^{DT}.

Table 2.5 presents the *lexical spaces* (i.e., the function N_{LS}) and the value spaces (i.e., the function $\cdot^{DT}$) in OWL 2 EL.

An OWL 2 EL vocabulary is a tuple $V = (V_C, V_{OP}, V_{DP}, V_I, V_{DT}, V_{LT}, V_{FA})$, where the symbols denote, respectively, the set of all *classes*, *object properties*, *data properties*, *individuals*, *datatypes*, *literals* and *facets*.

An OWL 2 EL interpretation is a tuple $I = (\Delta_I, \Delta_D, \cdot^C, \cdot^{OP}, \cdot^{DP}, \cdot^I, \cdot^{DT}, \cdot^{LT}, \cdot^{FA}, NAMED)$ with the meaning of the components defined below:

- Δ_I is a nonempty *object domain*.
- Δ_D is a nonempty *data domain*, disjoint with the object domain.
- $\cdot^C$ is a *class interpretation function* assigning every class expression a subset of the object domain, such that $(\texttt{owl:Thing})^C = \Delta_I$ and $(\texttt{owl:Nothing})^C = \emptyset$. `owl:Thing` corresponds to the top concept $\top$ in DL and `owl:Nothing` to the bottom concept $\bot$.
- $\cdot^{OP}$ is an *object property interpretation function* assigning every object property a subset of $\Delta_I \times \Delta_I$, such that $(\texttt{owl:topObjectProperty})^{OP} = \Delta_I \times \Delta_I$ and $(\texttt{owl:bottomObjectProperty})^{OP} = \emptyset$.
- $\cdot^{DP}$ is a *data property interpretation function* assigning every data property a subset of $\Delta_I \times \Delta_D$, such that $(\texttt{owl:topDataProperty})^{DP} = \Delta_I \times \Delta_D$ and $(\texttt{owl:bottomDataProperty})^{DP} = \emptyset$.
- $\cdot^I$ is an *individual interpretation function* assigning every individual an element of the object domain.
- $\cdot^{DT}$ is a *datatype interpretation function* assigning every literal a subset of the data domain, such that $(\texttt{rdfs:Literal})^{DT} = \Delta_D$
- $\cdot^{LT}$ is a *literal interpretation function*, assigning every literal the corresponding value from the value space of the datatype of the literal.
- $\cdot^{FA}$ is a *facet interpretation function*, assigning every facet a subset of the value space of the datatype constrained by the facet.
- $NAMED$ is a subset of the object domain corresponding to a set of all named individuals in the vocabulary.

An OWL 2 EL ontology is a set of axioms optionally extended with ontology annotations and import statements. Our primary interest lays in the axioms, as they are the only part of the ontology concerned with semantics. The annotations are merely information provided usually for a human user convenience (e.g., textual labels and comments), and the import statements are

Table 2.5: A description of the lexical spaces and value spaces of the datatypes allowed in OWL 2 EL [89, 68, 101, 34].

datatype	lexical space	value space
`rdf:XMLLiteral`	A set of all well-formed XML documents.	A set of all DOM DocumentFragments [30].
`owl:real`	not specified	The set of all real numbers.
`owl:rational`	Strings of the form `nom/den`, where `nom` is a `xsd:integer` and `den` is a positive `xsd:integer` without the plus sign.	The set of all rational numbers.
`xsd:decimal`	A typical computer representation using 0-9 as digits and `.` as a decimal separator, optionally with `+` or `-` in the front. If the sign is not specified, `+` is assumed.	The set of all numbers that can be obtained by dividing an integer by a non-negative power of ten.
`xsd:integer`	As `xsd:decimal`, but without the decimal separator.	The set of all integer numbers
`xsd:nonNegativeInteger`	As `xsd:integer`, but with `-` disallowed.	The set of all non-negative integers: $\{0, 1, 2, \ldots\}$
`xsd:string`	An arbitrary, finite sequence of characters allowed by XML	Identical to the lexical space
`xsd:anyURI`	As `xsd:string`.	Identical to the lexical space
`xsd:normalizedString`	As `xsd:string`, but with the following characters disallowed: carriage return, line feed, tab.	Identical to the lexical space
`xsd:token`	As `xsd:normalizedString`, but without leading or trailing spaces and without any consecutive substring of two or more spaces.	Identical to the lexical space
`xsd:Name`	As `xsd:token`, but with further characters disallowed, as defined by *Name* grammar in [88].	Identical to the lexical space
`xsd:NCName`	As `xsd:Name`, but with further characters disallowed, as defined by *NCName* grammar in [88].	Identical to the lexical space
`xsd:NMTOKEN`	As `xsd:token`, but with further characters disallowed, as defined by *Nmtoken* grammar in [88].	Identical to the lexical space
`rdf:PlainLiteral`	The set of all expressions of the form `"xxx"` and `"xxx"@lang`, where `xxx` is a `xsd:string` and `lang` is a (possibly empty) language tag conforming with [70].	The same as `xsd:string` plus the set of all pairs a string and a lowercase language tag.
`xsd:hexBinary`	Arbitrary, finite sequences of pairs of hexadecimal digits (i.e., 0-9, a-f, A-F)	Arbitrary, finite binary data.
`xsd:base64Binary`	The set of all correct Base64 strings [42].	Arbitrary, finite binary data.
`xsd:dateTime`	Intuitively, the lexical space can be represented as `yyyy-mm-ddThh:mm:ss.sss`$\pm$`zzz`. The formal specifications of both the lexical	
`xsd:dateTimeStamp`	space and the value space are quite complex and the interested reader should refer to [89] for details.	

directives used to guide tools processing the ontology where to find other, relevant definitions of the terms used in the ontology.

We provide a full list of OWL 2 EL data ranges, class expressions and axioms in Appendix A. We present them using the Manchester syntax [38] and Turtle syntax [18], and give their formal semantics according to [29].

Chapter 3

State of the art

3.1 Learning ontologies from text

One of the main bridges between natural language semantics and formal semantics is *FrameNet*, a set of structures called *frames* [56]. Each frame describes a certain situation (e.g., an act of buying), along with entities participating in the situation (e.g., a buyer and a seller) and its context (e.g., a thing, which is being bought). Each frame is accompanied by a set of relevant words that evoke the frame, i.e. enable its detection in a body of text.

The other very popular resource is *WordNet*, a lexical database of English, built around *synsets*, that represent related meanings of words [60]. Synsets are linked with each other using various lexical relations and are classified into general categories called *supersenses*.

Coppola et al. [19] present a multistep methodology for building a domain-specific *FrameNet* frames and using them to construct a relevant ontology. In the first step, frames are detected in a body of text: the text is tokenized, then keywords are detected, which are later used to select candidate frames. The candidates are disambiguated to select a single frame. Further on machine learning techniques are employed to align parts of the frame with appropriate parts of a sentence. The next step employs running a supersense tagger, which assigns supersenses to the parts of the sentence selected in the previous step. The third step processes the obtained set of frames and their corresponding tagging to detect co-occurrence *patterns* and provide a final set of frames specializations relevant to the domain. During the last step, the obtained frames are translated to OWL 2 creating a domain ontology.

Hoffart et al. [36] describe *YAGO*, an automatically built knowledge-base, originating in *Wikipedia*, *WordNet* and *GeoNames*[1]. The extraction process is mainly based on a set of declarative rules, divided into four categories. Factual rules are a representation of basic relations used by *YAGO*. Implication rules enable a simple, deductive reasoning service for deriving new facts based on these already present in *YAGO*. Replacement rules are used for cleansing and disambiguation, such as replacing string `USA` with `the United States of America`. Finally, extraction rules are tailored to the markup code used in Wikipedia and allow for extracting new facts from Wikipedia. An alignment between *YAGO*, *WordNet* and *GeoNames* is then computed to extend *YAGO* with additional facts.

Mitchell et al. [61] present a system called *NELL*, an abbreviation for *Never-Ending Language Learning*. It is a machine learning system working in an infinite loop. Initially, the system was seeded with a set of classes, each with a small set of instances and relations between these classes. The system was also given a large corpora of web pages. Now, during every iteration of the loop,

[1] `http://www.geonames.org/`

the system extends its corpus based on the current state of the Web, and applies machine learning techniques to discover patterns constituting new classes and relations, which are then verified against already gathered knowledge. The results are also periodically supervised by humans, to increase correctness of the obtained results.

SOFIE [90] and *PROSPERA* [65] are tools aiming at automatic extraction of ontological facts from a corpora of texts and linking them to an existing ontology. They are based on a three-step approach: first, a set of patterns is collected from a text corpora, based on provided examples and counterexamples. Then, the patterns are generalized by transforming them into sets of n-grams to concisely represent occurring subpatterns. Finally, a reasoning procedure is applied to ensure consistency within the set of facts as well as with the background ontology.

Novácek et al. [66] discuss a framework for integrating distributional semantics, emerging during statistical analysis of a large corpora of text, with the Semantic Web technologies. In the beginning, a set of RDF triples is extracted from the text. The triples are then converted into 4-dimensional tensor, which is then aggregated into more compact, 3-dimensional tensor representation, effectively creating a distributional semantics for the concepts used in the original text. Then, in a similar process, an existing ontology is transformed into the same representation. Using such a representation for both sources of knowledge enables fuzzy reasoning using numerical algorithms for tensors. The results of the reasoning can then be transformed back into a direct representation, to create new axioms for the ontology.

3.2 Ontology engineering by interaction with a user

A concept of games with a purpose is applicable to various fields, where simple tasks, that do not require expert knowledge, must be executed by humans [95]. If such a task is disguised as a computer game, providing entertainment to the user, the level of engagement of the user is higher and the user executes the task much more eagerly. Following this concept, *OntoGame*, a gaming framework for the Semantic Web, was created [83]. It consists of eight games solving various tasks related to the parts of the Semantic Web concerned with general knowledge. For example, the aim of *OntoPronto* is to extend *PROTON* ontology [84]. A player is asked to read a paragraph of text from *Wikipedia*, decide whether it describes an instance or a class and assign a *PROTON* concept that best describes it. The game is designed for two players, who collect points for reaching consensus. Another example is *Concept Game*, a *Facebook* game for validating commonsense knowledge obtained from text mining [35].

Wohlgenannt et al. in [100] report on a *Protégé* plugin *uComp*, that enables posting microtasks related to ontology engineering to crowdsourcing platforms such as *Amazon Mechanical Turk*[2] or *CrowdFlower*[3]. A wide range of ontology engineering tasks can be addressed using the four types of tasks offered by *uComp*: the crowd can be asked to judge degree of relatedness between concepts, verify if a given relationship exists between given concepts, name a relationship between two given concepts or verify if a given concept is relevant for the considered domain. The authors report that replacing an ontology engineer with the crowd does not significantly decrease the quality of the final ontology, while the costs can be lowered by as much as 83%.

Inel et al. [40] introduce *CrowdTruth*, a crowdsourcing framework applicable to a wide range of annotation and extraction tasks. A possible usage in ontology learning covers extraction of relations from text or annotating text with relations to provide further examples to a learning

[2] `http://www.mturk.com`
[3] `http://www.crowdflower.com`

system. It also delivers a set of metrics to assess the disagreement between the workers realizing a given task.

3.3 Concept learning

Fanizzi et al. [23] present *DL-FOIL*, a version of the *FOIL* algorithm [77] adapted to the Description Logics. The algorithm addresses the following concept learning problem: given an ontology, a set of positive examples and a set of negative examples, find a concept C such that $C(p)$ follows from the ontology for every positive example p and $C(n)$ does not follow from the ontology for every negative example n. It uses two refinement operators: a downward refinement operator to specialize currently considered concept and an upward refinement operator to generalize it. The algorithm refines the concept as long as the quality metrics are not high enough or the maximal complexity of the concept is not exceeded, whichever comes first.

Lehmann et al. [51, 16] introduce *DL-Learner*, a well-developed learning system targeting the Semantic Web. *DL-Learner* is capable of solving various concept learning tasks: standard supervised learning, the same as *DL-FOIL*; positive only learning, where negative examples are not provided and the algorithm must generalize only over positive examples, yet without yielding too general concept; class learning, where the goal is to find a formal description of a class that is already present in the ontology. *DL-Learner* also uses refinement operators and offers a set of learning algorithms, designed to address specific OWL 2 profiles and/or learning tasks. It integrates existing reasoners and, to some extent, is capable of using remote knowledge base over a SPARQL endpoint. Bühmann et al. [14] described an extension of *DL-Learner* taking textual descriptions of resources into account to increase precision of the mined concepts.

Galárraga et al. [27, 28] describe *AMIE*, an algorithm based on Inductive Logic Programming for mining association rules from large knowledge bases. The rules are mined under Partial Completeness Assumption, i.e., it is assumed that if there are some triples with a given subject and predicate in an ontology, then there are all such triples there. The mined rules are Horn rules using only binary predicates, that is they are always of the following form $p_1 \wedge \ldots \wedge p_n \rightarrow c$. Due to their properties, the mined rules can be transformed to OWL 2, to extend an ontology.

Lisi and Esposito [55] consider a similar setup for mining rules in DL+log, a framework integrating Description Logics with Datalog [78]. The aim is to learn a definition of a new class or a new predicate, using the $\mathcal{SHIQ}$ Description Logic and Datalog with negation, given sets of positive and negative examples and an existing ontology.

3.4 Learning ontologies from local data

Fu and Han [26] describe a method that is mining association rules in a multi-relational setting of a database. The shape of the mined rules is constrained by templates provided by the user. Variables in the templates can replace a relation name or an argument of a relation. While not explicitly grounded in the Semantic Web technologies, an approach as such could be easily adopted to mine an ontology from a database by appropriate setting of the templates.

Völker and Niepert [94] present an approach to mine an OWL 2 EL ontology from an RDF graph. The RDF graph is first transformed to a set of database tables, corresponding to various shapes of the mined axioms. The database contains only boolean values and is constructed so that a row corresponds to an individual in the graph and each column corresponds to a class expression. Then, the database is used as an input to the Apriori algorithm for association rules mining. The obtained rules are then used as axioms in the final ontology. This work was extended further in

[25, 93] to enable mining disjointness axioms. The authors discuss three methods, one based on correlation and two based on mining negative association rules using different approaches.

SDType is an inductive approach to assigning types to individuals based on an existing usage of the types rather than on the fixed schema [69]. It constructs a conditional probability distribution based on co-occurrence of predicates and types in the considered RDF graph and assigns new types using the distribution. The distribution can be seen as a non-OWL ontology taking uncertainty into account.

Sazonau et al. [79] describe an approach to learning *general concept inclusions* (GCIs) from data available in an ontology, taking into account the background knowledge already present in the ontology. It considers a space of hypotheses consisting of arbitrary sets of axioms not entailed by the ontology. From the space, it selects these hypotheses that are non-dominated w.r.t. the predefined set of measures and fulfill additional requirements set by the user. The result is a Pareto frontier of the hypotheses.

3.5 Learning ontologies from Linked Data

Learning ontologies directly from Linked Data did get much attention. A preliminary work of Bühmann and Lehmann [15] describe a two-step approach. In the first step a repository of ontologies is mined to extract frequent patterns present in already existing ontologies. The patterns are then converted to SPARQL query templates. In the second step, a vocabulary is retrieved from a SPARQL endpoint and used to construct queries out of the templates. The queries are then used to check for existence of the corresponding patterns in the endpoint. The approach does not consider any well-defined class of patterns.

Li and Sima [53] present a similar approach focused on OWL 2 EL. The authors present an algorithm for mining an ontology without nested class expressions using SPARQL `COUNT` queries. The efficiency of the algorithm is then increased by parallelization and subgraph detection in the mined RDF graph.

Downloading and processing an RDF graph as a whole is a demanding task, as it is usually a very large and dynamic structure. In case of stream data it may be challenging even to define what should be understood as a whole RDF graph. This shifts us towards approaches able to operate by posing only queries to the graph and processing only a piece of information at a time. Both approaches presented above tackle some aspects of this problem, but they applicability and scalability is questionable, as they heavily rely on SPARQL 1.1 constructs such as `COUNT` and `GROUP BY`, known to be computationally expensive to evaluate. Moreover, the first approach concentrates only on ontological constructs popular in some ontology repositories and thus it is prone to omit some valid, but less frequent constructs. The second approach completely omits more complex class expressions, that may be valuable for an ontology engineer. We thus find it valuable to propose new approaches, described in Chapters 4–6 of this thesis, that either address some specific problems in ontology learning or have more desirable properties than the existing approaches.

Chapter 4

Mathematical modelling for ontology learning

Throughout this chapter, we consider an ontology learning problem where we aim at dividing an existing ontology class into a set of subclasses such that the subclasses cover as many individuals belonging to the class as possible while maintaining overlap between the subclasses as low as possible. Moreover, we would like to have formal definition of the subclasses using equivalence, i.e. `owl:equivalentClass`, and the vocabulary already present in the ontology. A diagram giving an overview of the solution is presented in Figure 4.1. In Section 4.1 we describe Fr-ONT-Qu, a pattern mining algorithm proposed in [50]. The patterns generated by Fr-ONT-Qu are then processed by a *mathematical model* described in Section 4.2, and presented earlier in [74], in order to select the best subset of the patterns according to the criteria above. Both Fr-ONT-Qu and the mathematical model are implemented in *RMonto*, a plugin to *RapidMiner*, which is described in Section 4.4.

4.1 Fr-ONT-Qu algorithm

Fr-ONT-Qu is a pattern mining algorithm suitable for mining patterns from an RDF graph available in a SPARQL endpoint [50]. Fr-ONT-Qu originates from Fr-ONT, a frequent pattern mining algorithm for ontologies, using $\mathcal{EL}^{++}$ description logic as the language to express patterns [49]. The patterns mined with Fr-ONT-Qu are expressed as SPARQL SELECT queries with a single variable in the head and contain only a WHERE clause, which consists of a basic graph pattern (BGP) and optional filter expressions. Due to the nature of the algorithm, the graph corresponding to the BGP is a tree rooted in the variable from the head of the pattern and all the filter expressions are of form *variable* `>=` *numeric literal* or *variable* `<=` *numeric literal*.

The algorithm operates on a set of patterns, which in the beginning consists of a single pattern with an empty BGP. It executes the following steps in a loop:

1. Each of the patterns is refined by adding a new triple pattern or a filter expression, such that the requirements listed above are fulfilled.

2. The generated patterns are posed to the SPARQL endpoint in order to decide if they are frequent.

3. The best-first search principle is employed: a subset of the frequent patterns is selected according to some strategy and the next iteration of the algorithm is executed using the selected patterns.

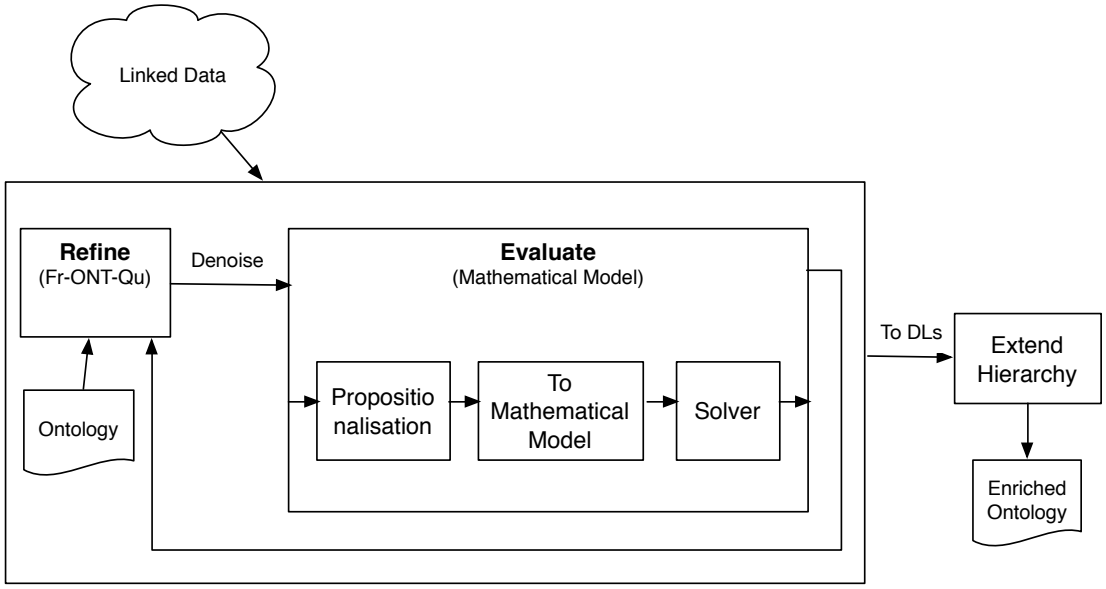


Figure 4.1: An overview of the proposed solution. First, Fr-ONT-Qu generates a set of patterns using an ontology and Linked Data. Noise, that is the patterns with a very low support, are removed in the process. The remaining patterns are then used to perform propositionalisation and generate a binary matrix, which is used as an input for the mathematical model. Solving the model yields a subset of the patterns, which can then be refined by Fr-ONT-Qu in the next iteration of the loop. Reprinted from [74].

In the original setup, we used the mined patterns in a classification task, and thus the selection strategy used a measure taking into account the labels in the classification task. However, while employing Fr-ONT-Qu to ontology learning, we used strategy based on a mathematical model described in the next section.

The search space in the graph pattern mining task is large, thus Fr-ONT-Qu is guided by a *declarative bias* defined by the user. The bias defines a vocabulary to be used during the refinement process, threshold for deciding whether a pattern is frequent, ranges of values to be used in filter expressions. In order to further limit the search space, the vocabulary may be partially ordered, ensuring that more general concepts are used before more specific ones during the refinement.

4.2 Mathematical model

Let $\mathbb{P} = \{P_1, P_2, \ldots, P_n\}$ be a set of patterns mined by Fr-ONT-Qu, i.e. SPARQL SELECT queries with a single variable in the head. Let `C` be an ontology class which is to be extended with new subclasses. Consider an RDF graph accessible using a SPARQL endpoint. Let $\mathbb{I} = \{I_1, I_2, \ldots, I_m\}$ be a set of IRIs that belong to class `C` w.r.t. the endpoint, i.e. the results of the query

```
SELECT ?x
WHERE
{
    ?x a C .
}
```

Denote by $\mathbb{R}(P)$ the set of results for a query $P \in \mathbb{P}$. While $\mathbb{R}(P)$ may contain elements from outside of the set $\mathbb{I}$, we are interested only in the elements from $\mathbb{I}$, and thus we assume that $\mathbb{R}(P) \subseteq \mathbb{I}$.

The goal is to select a subset $\mathbb{X} = \{P_{i_1}, \ldots, P_{i_k}\}$ of $\mathbb{P}$ such that the number of IRIs from $\mathbb{I}$ present in the results of exactly one of the selected queries is maximized. In other words, we want to maximize the size of the following set:

$$\{s \in \mathbb{I}\colon\ |\{P \in \mathbb{X}\colon s \in \mathbb{R}(P)\}| = 1\} \tag{4.1}$$

This follows immediately from the goal of our method: to split an existing class into subclasses with a minimum overlap. It is easy to see, that without restrictions, this problem can have a trivial solution with $\mathbb{X}$ containing a single, broad pattern. This contradicts our goal, as it does not split the class. We thus introduce a parameter λ, which is a minimal number of patterns in $\mathbb{X}$.

The description above expresses an ontology learning problem as an *optimization problem*. To solve it, binary *linear programming* (LP) may be employed. While solving such a problem is known to be NP-hard, the state of the art solvers (e.g. *Gurobi*[1] or *lp_solve*[2]) tend to work quite well in practice. Moreover, it is usually the case that one can define a time limit for solving and after the limit is exceeded, the best solution found so far is returned.

Let us first introduce the symbols used in the mathematical model. The relationship between $\mathbb{P}$ and $\mathbb{I}$ is definied by binary matrix A:

$$A_{i,j} = \begin{cases} 1 & I_i \in \mathbb{R}(P_j) \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \{1, 2, \ldots, m\}\, \forall j \in \{1, 2, \ldots, n\} \tag{4.2}$$

We denote by $\lambda \leq n$ the parameter, that defines a minimal number of selected patterns. We suggest setting it to a small value, like 2 or 3. Moreover, by M we denote a very large number (w.r.t. to the size of the problem), e.g. $M = \max\{m, n\} + 1$.

The model refers to three groups of variables. First, variables x_j define the solution, i.e. the set of selected patterns $\mathbb{X}$:

$$x_j = \begin{cases} 1 & P_j \in \mathbb{X} \\ 0 & \text{otherwise} \end{cases} \qquad \forall j \in \{1, 2, \ldots, n\} \tag{4.3}$$

The next set of variables y_i define a set of IRIs from $\mathbb{I}$ such that every of these IRIs is present in the results for at least one of the selected patterns

$$y_i = \begin{cases} 1 & \exists j\ (P_j \in \mathbb{X} \wedge I_i \in \mathbb{R}(P_j)) \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \{1, 2, \ldots, m\} \tag{4.4}$$

Finally, the set of variables z_i define a set of IRIs from $\mathbb{I}$ such that every of these IRIs is present in the results for at least two of the selected patterns

$$z_i = \begin{cases} 1 & \exists j \exists k\ (j \neq k \wedge P_j, P_k \in \mathbb{X} \wedge I_i \in \mathbb{R}(P_j) \wedge I_i \in \mathbb{R}(P_k)) \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \{1, 2, \ldots, m\} \tag{4.5}$$

Consider the difference $y_i - z_i$. From the definition of the variables it follows that $y_i - z_i = 1$ if, and only if, $\mathbb{X}$ contains exactly one pattern P_j such that $I_i \in \mathbb{R}(P_j)$. From this, it is straightforward to express the number of the elements of the set defined in Equation 4.1 as

$$\sum_{i=1}^{m} (y_i - z_i) \tag{4.6}$$

[1] `http://www.gurobi.com/`
[2] `http://lpsolve.sourceforge.net/`

This is a linear expression w.r.t. the variables and we will use it as the *optimization goal* in the model.

To properly define all three sets of variables, we must connect y_i and z_i to x_j. We begin with giving the necessary equations for y_i. First, we must ensure that $y_i = 1$ if the condition from the upper part of Equation 4.4 is fulfilled. Observe that $A_{i,j}x_j = 1$ if, and only if, both conditions hold:

- the pattern P_j is selected, i.e. $x_j = 1$;
- I_i is in the set of results for P_j, i.e. $A_{i,j} = 1$, which is a proxy for $I_i \in \mathbb{R}(P_j)$.

We are interested in $y_i = 1$ if I_i is in the results of any of the selected patterns, that is if $A_{i,j}x_j = 1$ for any j. We can express it with a set of $m \cdot n$ inequalities

$$y_i \geq A_{i,j}x_j \qquad \forall i \in \{1, 2, \ldots, m\} \, \forall j \in \{1, 2, \ldots, n\} \tag{4.7}$$

or, alternatively, with a set of m inequalities

$$My_i \geq \sum_{j=1}^{n} A_{i,j}x_j \qquad \forall i \in \{1, 2, \ldots, m\} \tag{4.8}$$

The second variant works as follow: if some of the multiplications $A_{i,j}x_j$ is equal to 1, then the sum is greater than 0 and less than M, which is by definition larger than n. For the inequality to be satisfied, My_i must be greater than the sum in the left-hand side, and so y_i must be equal to 1.

Both of these variants guarantee that $y_i = 1$ if any of the relevant patterns is selected, but they do not guarantee that $y_i = 0$ if no relevant pattern is selected. In terms of the variables and matrix A, it means that $y_i = 0$ if all of the multiplications $A_{i,j}x_j$ equal to 0, thus their sum must also equal to 0, i.e.

$$y_i \leq \sum_{j=1}^{n} A_{i,j}x_j \qquad \forall i \in \{1, 2, \ldots, m\} \tag{4.9}$$

The right-hand sides of Equation 4.8 and Equation 4.9 define the number of the selected patterns, for which I_i is in the set of results for any of these patterns. To define the variables z_j, we will use the same expression, but with 1 subtracted. In this way, we obtain the expression

$$-1 + \sum_{j=1}^{n} A_{i,j}x_j \tag{4.10}$$

that is positive if I_i is in the results of at least two patterns and nonpositive otherwise. Following the same line of reasoning as before, we obtain

$$Mz_i \geq -1 + \sum_{j=1}^{n} A_{i,j}x_j \qquad \forall i \in \{1, 2, \ldots, m\} \tag{4.11}$$

This equation guarantees that $z_i = 1$ if at least two times $A_{i,j}x_j = 1$, i.e. at least two patterns having I_i in their results are selected to $\mathbb{X}$.

Again, we must ensure that $z_i = 0$ if

$$-1 + \sum_{j=1}^{n} A_{i,j}x_j \leq 0 \tag{4.12}$$

If we subtract 1 and multiply by -1 sideways, we arrive at

$$2 - \sum_{j=1}^{n} A_{i,j}x_j \geq 1 \tag{4.13}$$

Here we face a problem of the same kind as before, i.e. we must ensure that $1 - z_i = 1$ if $2 - \sum_{j=1}^{n} A_{i,j}x_j$ is positive. Using the same reasoning, we get

$$M(1 - z_i) \geq 2 - \sum_{j=1}^{n} A_{i,j}x_j \qquad \forall i \in \{1, 2, \ldots, m\} \tag{4.14}$$

If $z_i = 0$, then the left hand side is of the inequality is M and is greater than the right hand side. If $z_i = 1$, then the left hand side is 0 and thus the sum $\sum_{j=1}^{n} A_{i,j}x_j$ must be at least 2 for the inequality to be satisfied.

Finally, to ensure that at least λ patterns are selected, we use the following inequality

$$\sum_{j=1}^{n} x_j \leq \lambda \tag{4.15}$$

Finally, we arrive at the following linear model, consisting of $4m + 1$ inequalities:

$$\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{m}(y_i - z_i) \\
\text{subject to} \quad & My_i \geq \sum_{j=1}^{n} A_{i,j}x_j && \forall i \in \{1, 2, \ldots, m\} \\
& y_i \leq \sum_{j=1}^{n} A_{i,j}x_j && \forall i \in \{1, 2, \ldots, m\} \\
& Mz_i \geq -1 + \sum_{j=1}^{n} A_{i,j}x_j && \forall i \in \{1, 2, \ldots, m\} \\
& M - Mz_i \geq 2 - \sum_{j=1}^{n} A_{i,j}x_j && \forall i \in \{1, 2, \ldots, m\} \\
& \sum_{j=1}^{n} x_j \leq \lambda \\
& \text{all variables binary}
\end{aligned}$$

Extending the ontology requires translation of the patterns from SPARQL to OWL 2, using the following steps:

1. The BGP and filter expressions are extracted from a pattern.
2. The BGP is represented as a graph: each triple pattern constitutes a single arc, from a node corresponding to a subject to the node corresponding to an object and labeled with the predicate.
3. For each filter expression `FILTER(var op value)`:
   a) a new node for `value` is added;
   b) an arc, labeled with the operator `op`, from the node corresponding to the variable `var` to the new node is added.

Due to the specific way Fr-ONT-Qu works, the obtained graph is a tree and may be rooted in the node corresponding to the variable from the head of the pattern. An example of a pattern and the corresponding tree is presented in Figure 4.2.

The tree is then transformed into an OWL 2 EL *class expression*, using the function presented in Algorithm 4.1.

```
SELECT DISTINCT ?x
WHERE
{
  ?x a dbo:Book .
  ?x dbo:autor ?y .
  ?y a dbo:Writer .
  ?y dbo:birthYear ?z .
  FILTER (?z >= "1901"^^xsd:integer)
}
```

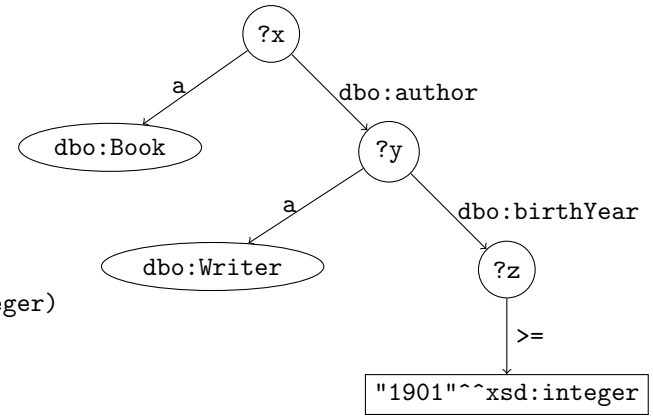


Figure 4.2: A sample pattern and the corresponding tree, obtained by extending the graph corresponding to the BGP with the filter expressions. The tree is rooted in `?x`, as this is the variable from the head of the query.

```
1  function TreeToClassExpression(n)
2      if n is a leaf then return owl:Thing
3      O ← ∅                              // a set of class expressions and data ranges
4      foreach child node m of n do
5          if l_a(n, m) is rdf:type then
6              assert m is a leaf and l_n(m) is a class name
7              O ← O ∪ {l(m)}
8          end
9          else if l_a(n, m) is >= or <= then
10             assert m is a leaf and l_n(m) is a literal
11             T ← the datatype of l_n(m) // a temporary variable
12             O ← O ∪ { T[l_a(n, m) l_n(m)] } // add a data range with a ≤ or ≥ facet
13         end
14         else if l_n(m) is not a variable then
15             O ← O ∪ {l_a(n, m) VALUE l(m)}
16         end
17         else
18             assert l_n(m) is a variable
19             O ← O ∪ {l_a(n, m) SOME TreeToClassExpression(m)}
20         end
21     end
22     return Elements of O joined with AND // a single class expression or a data
           range consisting of an intersection of the elements of O
23 end
```

Algorithm 4.1: The algorithm transforming a tree corresponding to a pattern into a OWL 2 EL class expression in Manchester syntax. We denote by $l_a(m, n)$ a label of the arc from m to n, and by $l_n(m)$ a label of the node m.

Table 4.1: A sample RDF graph, being a small excerpt of DBpedia and concerning five locomotives in use or used in the past in Poland: ET22, EU07, OKl27, SM42 and SU46.

dbr:PKP_class_ET22	dbo:builder	dbr:Pafawag	;
	a	dbo:Locomotive	.
dbr:PKP_class_EU07	dbo:builder	dbr:H._Cegielski_-_Poznań_S.A.	,
		dbr:Pafawag	;
	a	dbo:Locomotive	.
dbr:PKP_class_OKl27	dbo:builder	dbr:H._Cegielski_-_Poznań_S.A.	;
	a	dbo:Locomotive	.
dbr:PKP_class_SM42	dbo:builder	dbr:Fablok	;
	a	dbo:Locomotive	.
dbr:PKP_class_SU46	dbo:builder	dbr:H._Cegielski_-_Poznań_S.A.	;
	a	dbo:Locomotive	.

Table 4.2: Matrix A corresponding to the RDF graph presented in Table 4.1 and the patterns presented in Section 4.3

IRI \ pattern	1	2	3	4
dbr:PKP_class_ET22	1	1	0	0
dbr:PKP_class_EU07	1	1	1	0
dbr:PKP_class_OKl27	1	0	1	0
dbr:PKP_class_SM42	1	0	0	1
dbr:PKP_class_SU46	1	0	1	0

4.3 Example

Consider a simple RDF graph presented in Table 4.1. Let $\mathbb{I}$ = {dbr:PKP_class_ET22, dbr:PKP_class_EU07, dbr:PKP_class_OKl27, dbr:PKP_class_SM24, dbr:PKP_class_SU46}, i.e. the number of IRIs $m = 5$. Assume that $\lambda = 2$ and consider the following set $\mathbb{P}$ of $n = 4$ patterns:

1. `SELECT ?x WHERE { ?x a dbo:Locomotive }`
2. `SELECT ?x WHERE { ?x dbo:builder dbr:Pafawag }`
3. `SELECT ?x WHERE { ?x dbo:builder dbr:H._Cegielski_-_Poznań_S.A. }`
4. `SELECT ?x WHERE { ?x dbo:builder dbr:Fablok }`

The corresponding matrix A is presented in Table 4.2.

The mathematical model corresponding to the example is presented in Equation 4.16. Solving the presented model, we arrive at the solution presented in Table 4.3, which corresponds to selection of the following patterns:

2. `SELECT ?x WHERE { ?x dbo:builder dbr:Pafawag }`
3. `SELECT ?x WHERE { ?x dbo:builder dbr:H._Cegielski_-_Poznań_S.A. }`
4. `SELECT ?x WHERE { ?x dbo:builder dbr:Fablok }`

The goal function is equal to 4, as dbr:PKP_class_EU07 is covered by patterns 2 and 3 and thus $z_2 = 1$ and $y_2 - z_2 = 0$. After the patterns are converted, we obtain the final class expressions:

2. dbo:builder SOME dbr:Pafawag

Equation 4.16: The mathematical model corresponding to the presented example

$$
\begin{aligned}
\text{maximize} \quad & \sum_{i=1}^{5}(y_i - z_i) \\
\text{subject to} \quad & My_1 \geq x_1 + x_2 \\
& My_2 \geq x_1 + x_2 + x_3 \\
& My_3 \geq x_1 + x_3 \\
& My_4 \geq x_1 + x_4 \\
& My_5 \geq x_1 + x_3 \\
& y_1 \leq x_1 + x_2 \\
& y_2 \leq x_1 + x_2 + x_3 \\
& y_3 \leq x_1 + x_3 \\
& y_4 \leq x_1 + x_4 \\
& y_5 \leq x_1 + x_3 \\
& Mz_1 \geq -1 + x_1 + x_2 \\
& Mz_2 \geq -1 + x_1 + x_2 + x_3 \\
& Mz_3 \geq -1 + x_1 + x_3 \\
& Mz_4 \geq -1 + x_1 + x_4 \\
& Mz_5 \geq -1 + x_1 + x_3 \\
& M - Mz_1 \geq 2 - x_1 - x_2 \\
& M - Mz_2 \geq 2 - x_1 - x_2 - x_3 \\
& M - Mz_3 \geq 2 - x_1 - x_3 \\
& M - Mz_4 \geq 2 - x_1 - x_4 \\
& M - Mz_5 \geq 2 - x_1 - x_3 \\
& x_1 + x_2 + x_3 + x_4 \geq 2 \\
& \text{all variables binary}
\end{aligned}
$$

Table 4.3: The solution to the mathematical model presented in Equation 4.16

y_1	1	z_1	0	x_1	0
y_2	1	z_2	1	x_2	1
y_3	1	z_3	0	x_3	1
y_4	1	z_4	0	x_4	1
y_5	1	z_5	0		

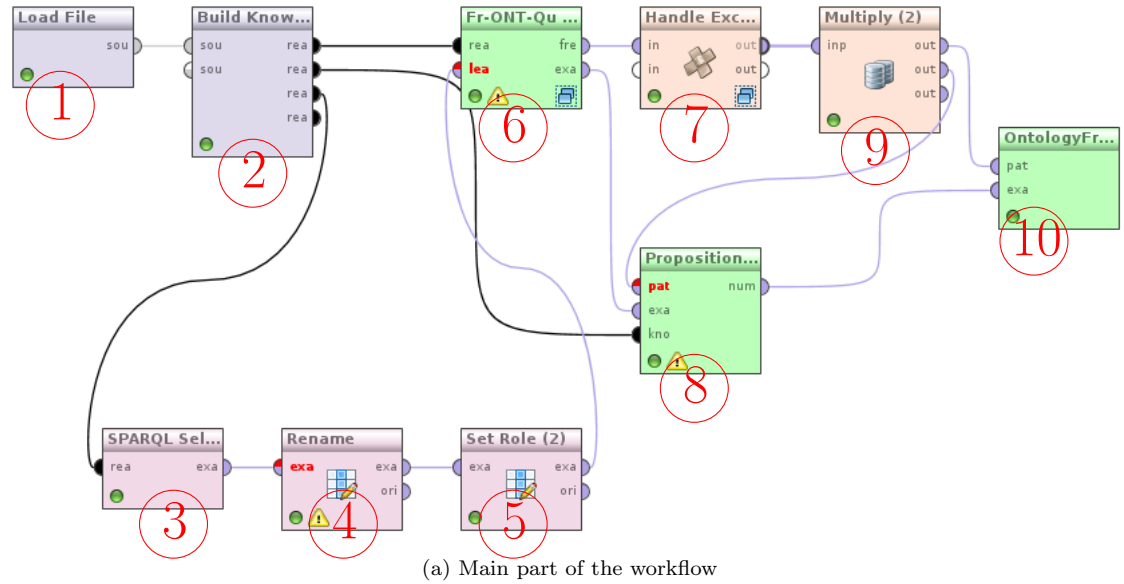
3. `dbo:builder` SOME `dbr:H._Cegielski_-_Poznań_S.A.`

4. `dbo:builder` SOME `dbr:Fablok`

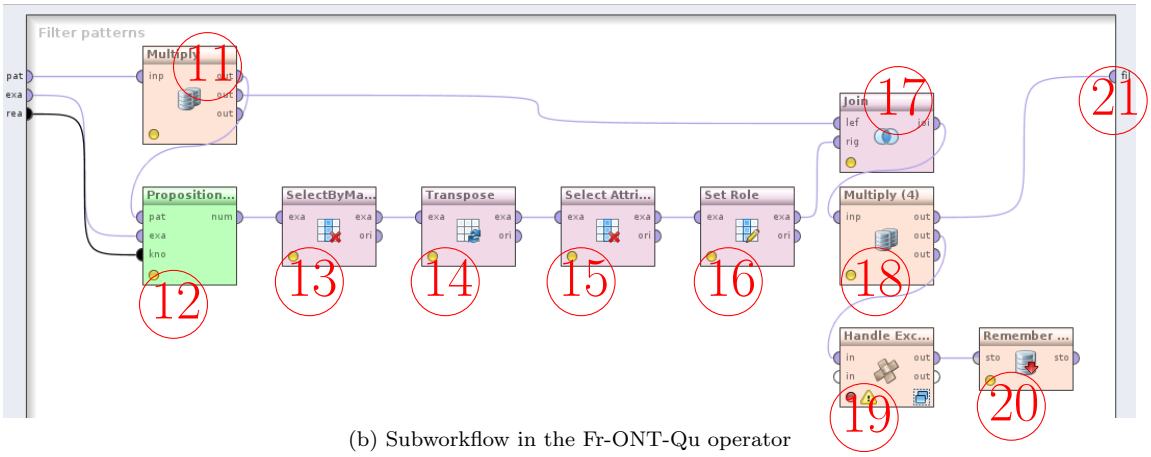
The user should now name these class expressions introducing them as new classes to the ontology.

4.4 Plugin to *RapidMiner*

RapidMiner is a tool for constructing data mining and machine learning workflows by visual means [59]. A single execution unit in a workflow is called an *operator*, which roughly corresponds to a function in a programming language. Each operator retrieves data from its input ports, process them accordingly and delivers the results to its output ports. The connections between the ports of the operators in a workflow define the flow of data.



(a) Main part of the workflow



(b) Subworkflow in the Fr-ONT-Qu operator

Figure 4.3: A sample workflow using the presented mathematical model to extend an ontology. Data are loaded with operator Load File (1) and into in-memory RDF store (2). A set of IRIs is selected using a SPARQL query (3-5) and Fr-ONT-Qu is run (6). In each iteration of Fr-ONT-Qu, binary matrix is constructed (12) and used as an input to the mathematical model (13). The result of the execution of the model is then used to filter the input patterns (14-17). Finally, the selected patterns are remembered for future use (19-20) and returned to Fr-ONT-Qu (21). After Fr-ONT-Qu finishes, the patterns selected by the execution of the mathematical models are recalled (7), a binary matrix is generated (8), which is then used to generate the final ontology (10). Operators 9, 11, 18 are used to connect multiple inputs to a single output.

While participating in European Union Framework Programme 7 project *e-LICO: An e-Laboratory for Interdisciplinary Collaborative Research in Data Mining and Data-Intensive Science*[3], we developed *RMonto*, a plugin extending *RapidMiner* with capabilities of processing Semantic Web data [73]. Both Fr-ONT-Qu and the mathematical model described above are available as operators in *RMonto*. A sample workflow using these operators is presented in Figure 4.3

[3] `http://e-lico.eu/`

Chapter 5

Formal Concept Analysis supported by machine learning

5.1 Formal Concept Analysis

Formal Concept Analysis (FCA) is an area of mathematics founded to formalize a discourse about concepts and their hierarchies. Below, we present a brief introduction to FCA following [98]. Every *formal concept* is represented simultaneously in two ways: by its *intension*, i.e., a set of criteria to recognize whether an object belongs to the concept and by its *extension*, i.e., a set of objects belonging to the concept. A *formal context* is a triple (G, M, I) such that G is a set of objects, M is a set of binary attributes (i.e., an object either has certain attribute or not) and $I \subseteq G \times M$ is a binary relation such that $(g, m) \in I$ (or, using the infix notation, gIm) if, and only if, the object g has the attribute m. A *derivation operator* $\cdot^I$ is defined for any $X \subseteq G$ and any $Y \subseteq M$:

$$X^I = \{m \in M \colon \forall g \in X \colon (g, m) \in I\} \tag{5.1}$$

$$Y^I = \{g \in G \colon \forall m \in Y \colon (g, m) \in I\} \tag{5.2}$$

A formal concept is a pair (A, B) such that A is the extension, i.e. a set of objects $A \subseteq G$, while B is the intension, i.e. a set of attributes $B \subseteq M$ and the following equalities hold: $A = B^I$ and $B = A^I$. In other words, A is the set of all objects having all the attributes of B and B is a set of all common attributes of the objects of A.

It is convenient to present a formal context as a table, where a row corresponds to an object, a column corresponds to an attribute and in the cells the relation I is denoted. An example is presented in Table 5.1.

On top of the formal concepts, we define a subconcept–superconcept relation as

$$(A_1, B_1) \prec (A_2, B_2) \iff A_1 \subset A_2 \iff B_1 \supset B_2 \tag{5.3}$$

That is, *the concept* (A_1, B_1) *is a subconcept of the concept* (A_2, B_2) if the set of objects of the first set is a subset of the set of objects of the second set, or using the notion of attributes, if the set of attributes of the first concept is a superset for the set of attributes of the second concept. In other words, one concept is more specific than the other, if it describes a subset of objects of the other one or requires more attributes than the other one.

Denote by $\mathbb{K}$ the set of all concepts in the context (G, M, I). The $\prec$ relation is a partial order and thus forms the *concept lattice* of the context (G, M, I). *The most general (top) concept* in the lattice is given by Equation 5.4, while *the most specific (bottom) concept* is given by Equation 5.5. The former is a concept defined by the attributes common to all concepts, while the later is a

Table 5.1: A sample formal context about means of transportation. It consists of five objects `train`, `boat`, `amphibian`, `sleight` , `trolleybus` and four attributes `has wheel`, `can sail`, `has pantograph`, `requires infrastructure`. If there is × in a cell, it means that the attribute and the object are in the relation I.

objects \ attributes	has wheels	can sail	has pantograph	requires infrastructure
train	×		×	×
boat		×		
amphibian	×	×		
sleight				
trolleybus	×		×	×

concept defined by the objects common to all concepts.

$$\left(\left(\bigcap_{(A,B)\in\mathbb{K}} B \right)^I , \bigcap_{(A,B)\in\mathbb{K}} B \right) \tag{5.4}$$

$$\left(\bigcap_{(A,B)\in\mathbb{K}} A, \left(\bigcap_{(A,B)\in\mathbb{K}} A \right)^I \right) \tag{5.5}$$

Following the example from Table 5.1, the corresponding concept lattice is presented in Figure 5.1.

5.2 Handling incomplete knowledge

In this section we follow [17] and present an approach to deal with incomplete knowledge in Formal Concept Analysis, a necessary step to apply FCA to OWL 2 ontologies.

The relation I in a formal context enables us to express only a binary information: either the relation holds gIm, i.e. the object g has the attribute m, or it does not $\neg(gIm)$ and then we can not distinguish if the object g does not have the attribute m or if it is unknown if the object g has the attribute m. Formal context is extended to store the appropriate information as follows. An *incomplete formal context* is a quadruple $(G, M, \{+, ?, -\}, J)$ such that G is a set of objects, M is a set of attributes and $J \subseteq G \times M \times \{+, ?, -\}$ is a ternary relation such that:

- $(g, m, +) \in J$ if the object g has the attribute m;
- $(g, m, -) \in J$ if the object g does not have the attribute m;
- $(g, m, ?) \in J$ if it is unknown if the object g has the attribute m.

Moreover, we require that for every pair $(g, m) \in G \times M$ there exists exactly one of the abovementioned triples in J, i.e. J is a function from $G \times M$ to $\{+, ?, -\}$.

It is easy to notice that there are multiple ways one can transform an incomplete formal context into a formal context. The transformation following the motivation given above would be to define

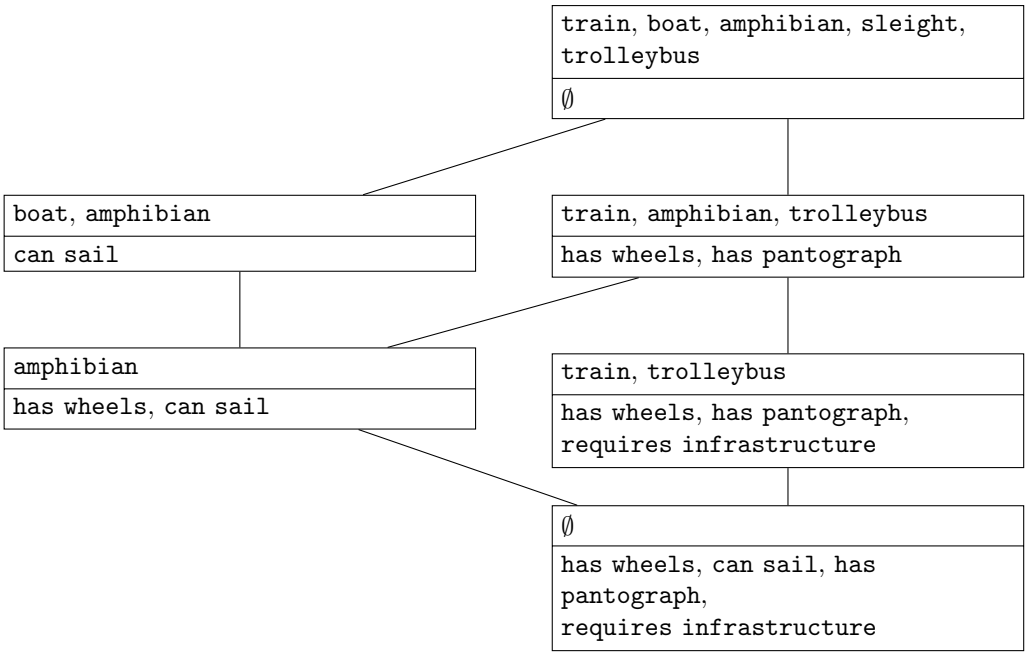


Figure 5.1: The concept lattice corresponding to the example from Table 5.1. The top part of each node contains the extensional part of the corresponding concept, i.e., a set of objects, while the bottom part contains the intensional part, i.e., a set of attributes. The most general concept is placed at the top of the lattice and the most specific concept is placed at the bottom.

the relation I as including all the situations where there is $+$ in the third position in J, i.e.

$$I = \{(g, m) \colon (g, m, +) \in J\} \tag{5.6}$$

that is the distinction between unknown and negative information is forgotten. Of course that would defeat our purpose and we are interested in a different setup. A formal context (G, M, I) is called a *completion* of an incomplete formal context $(G, M, \{+, ?, -\}, J)$ if both conditions hold:

- every triple $(g, m, +)$ implies that g has the attribute m

$$\{(g, m) \colon (g, m, +) \in J\} \subseteq I \tag{5.7}$$

- every triple $(g, m, -)$ implies that g is not in the relation I with m

$$\{(g, m) \colon (g, m, -) \in J\} \cap I = \emptyset \tag{5.8}$$

The intuition is that there are many possible completions, i.e. every ? can become either $+$ or $-$, and we do not perform any commitment at this stage yet. Of course, if $\{(g, m) \colon (g, m, ?) \in J\} = \emptyset$, then there exists exactly one completion of the incomplete formal context.

Consider an incomplete formal context $(G, M, \{+, ?, -\}, J)$. A *partial object description* is a triple (g, A, S) such that:

- $g \in G$ is an object of the incomplete formal context;
- A is the set of all attributes possessed by g;

$$A = \{m \in M \colon (g, m, +) \in J\} \tag{5.9}$$

- S is the set of all attributes such that g does not have this attribute.

$$S = \{m \in M \colon (g, m, -) \in J\} \tag{5.10}$$

Table 5.2: A sample incomplete formal context about means of transportation. It consists of five objects `train`, `boat`, `amphibian`, `sleight` , `trolleybus` and four attributes `has wheel`, `can sail`, `has pantograph`, `requires infrastructure`. The symbols in the cells corresponds to the third argument of the relation J, i.e. $+$ means that the object has the attribute, $-$ means that the object does not have the attribute and ? means that it is unknown.

objects \ attributes	has wheels	can sail	has pantograph	requires infrastructure
train	+	−	+	+
boat	?	+	−	?
amphibian	+	+	?	?
sleight	−	−	−	−
trolleybus	+	−	+	+

A set of partial object descriptions $\{(g_1, A_1, S_1), \ldots, (g_n, A_n, S_n)\}$ corresponds exactly to the incomplete formal context $(G, M, \{+, ?, -\}, J)$ such that:

- G consists of all the objects of the set: $G = \{g_1, \ldots, g_n\}$
- J contains triples with $+$ for the attributes in A_i, triples with $-$ for the attributes of S_i and triples with ? for the

$$J = \bigcup_{i=1}^{n} (\{(g_i, m, +) \colon m \in A_i\} \cup \{(g_i, m, -) \colon m \in S_i\} \cup \{(g_i, m, ?) \colon m \in M \backslash (A_i \cup S_i)\}) \tag{5.11}$$

A *full object description* is a partial object description (g, A, S) such that $A \cup S = M$, i.e. the set $\{m \in M \colon (g, m, ?) \in J\}$ is empty. The set of full object descriptions for all objects $g \in G$ corresponds to a single formal context, as it defines an incomplete formal context with the set $\{(g, m) \in G \times M \colon (g, m, ?) \in J\}$ empty, and thus having a single completion.

A sample incomplete context is presented in Table 5.2, using the same attributes and objects as the previous example in Table 5.1 and Figure 5.1. One of the partial object descriptions in the context is (`boat`, {`can sail`}, {`has pantograph`}). This partial object description is not a full object description, as it does not contain the attributes `has wheels` and `requires infrastructure`. A full object description presented in the example is (`train`, {`has wheels`, `requires infrastructure`, `has pantograph`}, {`can sail`}). The previous example from Table 5.1 is a completion of this incomplete context, but not the only one. Another one could be constructed, e.g., by replacing all the questions marks ? in Table 5.2 with pluses $+$.

A partial object description (g, A, S) *is extended by* a partial object description (g, A', S') if $A \subseteq A'$ and $S \subseteq S'$. We generalize the notion to incomplete formal contexts and say that an incomplete formal context K_1 is extended by an incomplete formal context K_2 if every partial object description in K_1 is extended by some partial object description in K_2. For example, the partial object description (`boat`, {`can sail`}, {`has pantograph`}) from Table 5.2 is extended by the full object description (`boat`, {`can sail`}, {`has wheels`, `has pantograph`, `requires infrastructure`}) from Table 5.1.

5.3 Attribute implication

We say that *a set of attributes X implies a set of attributes Y*, denoted by the *attribute implication* $X \to Y$, if and only if, $X^I \subseteq Y^I$. For example, the set of attributes $X = \{$`requires infrastructure`, `has pantograph`$\}$ implies the set of attributes $Y = \{$`has wheels`$\}$, as $X^I = \{$`train`, `trolleybus`$\} \subseteq \{$`train`, `amphibian`, `trolleybus`$\} = Y^I$.

An attribute implication $L \to R$ is *refuted by* a partial object description if $L \subseteq A$, but $R \cap S \neq \emptyset$. We say that the implication is refuted by an incomplete formal context if it is refuted by any partial object description in the context. For example, the implication $\{$`has wheels`$\} \to \{$`has pantograph`$\}$ is refuted by `amphibian` in the formal context presented in Table 5.1, but is not refuted by the incomplete formal context from Table 5.2, as for all the objects with the attribute `has wheels`, i.e. `train`, `amphibian`, `trolleybus`, none of them is known to not have the attribute `has pantograph`.

Let $\mathcal{L}$ be a set of implications and $P \subseteq M$ a set of attributes. The *implicational closure* of P w.r.t. the set $\mathcal{L}$, denoted by $\mathcal{L}(P)$ is the smallest set $Q \subseteq M$ such that

- it contains all the attributes of P: $P \subseteq Q$;
- for every implication in the set $\mathcal{L}$, if premises of the implication are in the set Q, so are the conclusions.

$$\forall (L \to R) \in \mathcal{L} \ (L \subseteq Q \to R \subseteq Q) \tag{5.12}$$

An implication $L \to R$ *follows* from a set of implication $\mathcal{J}$ if $R \subseteq \mathcal{J}(L)$, that is R is a subset of the implicational closure of L w.r.t. $\mathcal{J}$. A set of implications $\mathcal{J}$ is an *implication base* for a set of implications $\mathcal{L}$ if the following three conditions are jointly satisfied:

- soundness – $\mathcal{L}$ contains every implication following from $\mathcal{J}$;
- completeness – every implication in $\mathcal{L}$ follows from $\mathcal{J}$;
- minimality – no strict subset of $\mathcal{J}$ has both of the abovementioned properties w.r.t $\mathcal{L}$.

Assume any linear order on the set M: $m_1 < m_2 < \ldots < m_n$. For any $A, B \subseteq M$, A *precedes* B *in the order* i, denoted $A <_i B$, if and only if:

- B contains m_i while A does not: $m_i \in B \backslash A$
- A and B both contain exactly the same attributes preceding m_i

$$A \cap \{m_1, \ldots, m_{i-1}\} = B \cap \{m_1, \ldots, m_{i-1}\} \tag{5.13}$$

The *lectic order* on the set M is the union of the orders $1, \ldots, n$:

$$< \equiv \bigcup_{i=1}^{n} <_i \tag{5.14}$$

Recall the example presented in Table 5.2. A sample lectic order along with the corresponding orders $1, 2, 3, 4$ are presented in Table 5.3

Table 5.3: An example of lectic order for the following four attributes: `has wheels` < `can sail` < `has pantograph` < `requires infrastructure`. The header of a row or a column is a positional encoding of the set, with the leftmost position corresponding to the attribute `has wheels`, second to `can sail`, third to `has pantograph` and the rightmost to `requires infrastructure`. Value 1 in a given position means that the corresponding attribute is present in the set and 0 means otherwise, e.g. 0100 denotes the set {`can sail`}, while 1110 denotes the set {`has wheels`, `can sail`, `has pantograph`}. A row corresponds to set A and a column to set B. An empty cell means that A does not precede B in the lectic order, while presence of a number i means that A precedes B in the order i and thus in the lectic order. For example, the set {`can sail`} precedes the set {`can sail`, `has pantograph`} in the order of 3, because both sets are identical w.r.t the first two attributes, i.e. they both do not contain the attribute `has wheels` and they both contain the attribute `can sail`, but they differ on the third attribute, as the first set does not contain `has pantograph`.

A \ B	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000		4	3	3	2	2	2	2	1	1	1	1	1	1	1	1
0001			3	3	2	2	2	2	1	1	1	1	1	1	1	1
0010				4	2	2	2	2	1	1	1	1	1	1	1	1
0011					2	2	2	2	1	1	1	1	1	1	1	1
0100						4	3	3	1	1	1	1	1	1	1	1
0101							3	3	1	1	1	1	1	1	1	1
0110								4	1	1	1	1	1	1	1	1
0111									1	1	1	1	1	1	1	1
1000										4	3	3	2	2	2	2
1001											3	3	2	2	2	2
1010												4	2	2	2	2
1011													2	2	2	2
1100														4	3	3
1101															3	3
1110																4
1111																

5.4 Attribute exploration algorithm

Attribute exploration is a generic name for any algorithm that aims to obtain full knowledge about attribute implications in a given formal context. This knowledge is obtained by an interaction with the user, usually by presenting to the user an implication and asking to decide either if the implication holds or to give a counterexample.

Algorithm 5.1 presents an attribute exploration algorithm based on [5], which was designed with OWL 2 ontologies in mind. The assumption of the algorithm is that there exists some formal context $\overline{\mathcal{K}}$, known to the user, but unknown to the algorithm. The algorithm is given only an incomplete formal context $\mathcal{K}$, such that its completion is $\overline{\mathcal{K}}$. The goal of the algorithm is to compute the implication base of the set of all implications, that are not refuted by the formal context $\overline{\mathcal{K}}$ posing as few questions to the user as possible. The main principle of operation is to generate the premises (left-hand side) of an implication by iterating over the lectic order and computing the implicational closure of the premises and then filling the conclusions (right-hand side) with the largest set of attributes, which is not refuted by the incomplete context $\mathcal{K}$.

Recall the incomplete context from Table 5.2 and assume the attributes are ordered as presented in the example about the lectic order in Table 5.3.

In the beginning of the execution of the algorithm, P is an empty set and thus in line 4 K_P is compute as the difference of two sets: the first set is the whole set M = {`has wheels`, `can sail`, `has pantograph`, `requires infrastructure`}; the second set contains all the attributes that oc-

Data: $M = \{m_1, \ldots, m_n\}$ an ordered set of attributes, $\mathcal{K}$ an incomplete formal context, such that its completion is a formal context $\overline{\mathcal{K}}$
Result: $\mathcal{L}$ is the base of the set of all implications, that are not refuted by $\overline{\mathcal{K}}$

1 $\mathcal{L} \leftarrow \emptyset$
2 $P \leftarrow \emptyset$
3 **while** $P \neq M$ **do**
4 $\quad K_P \leftarrow M \setminus \bigcup_{(g,A,S) \in \mathcal{K} \wedge P \subseteq A} S$
5 $\quad newP \leftarrow$ **true**
6 $\quad$ **if** $K_P \neq P$ **then**
7 $\quad\quad$ Ask the user if the implication $P \rightarrow K_P \setminus P$ holds w.r.t. the context $\overline{\mathcal{K}}$
8 $\quad\quad$ **if** *yes* **then**
9 $\quad\quad\quad \mathcal{L} \leftarrow \mathcal{L} \cup \{P \rightarrow K_P \setminus P\}$
10 $\quad\quad$ **end**
11 $\quad\quad$ **else**
12 $\quad\quad\quad$ Ask the user to extend $\mathcal{K}$ such that the implication $P \rightarrow K_P \setminus P$ is refuted by the context
13 $\quad\quad\quad newP \leftarrow$ **false**
14 $\quad\quad$ **end**
15 $\quad$ **end**
16 $\quad$ **if** $newP$ **then**
17 $\quad\quad$ **for** $j \leftarrow n, \ldots, 2$ **do**
18 $\quad\quad\quad P' \leftarrow \mathcal{L}((P \cap \{m_1, \ldots, m_{j-1}) \cup \{m_j\})$
19 $\quad\quad\quad$ **if** $P <_j P'$ **then**
20 $\quad\quad\quad\quad P \leftarrow P'$
21 $\quad\quad\quad\quad$ **break**
22 $\quad\quad\quad$ **end**
23 $\quad\quad$ **end**
24 $\quad$ **end**
25 **end**

Algorithm 5.1: An attribute exploration algorithm based on [5].

cur in the set of negative attributes S for any partial object description. In other words, the second set consists of all the attributes for which "$-$" occurs at least once in the incomplete formal context from Table 5.2. As this is the case for all the attributes, the set K_P becomes empty, and the execution goes to line 17 to find next suitable set P. The iteration starts with $j = 4$ and P is extended with the corresponding attribute, i.e., `requires infrastructure`. The set $\mathcal{L}$ is empty and thus $P' = \{$`requires infrastructure`$\}$. The corresponding header for this set in Table 5.3 is 0001, and we can observe that it is greater than 0000 w.r.t. the order 4. P is then replaced by P' and the next iteration of the loop begins.

In line 4, only the partial object descriptions for `train` and `trolleybus` are taken into account, as only these two objects are known to have the attribute `requires infrastructure`. The sum now becomes $\{$`can sail`$\}$ and thus $K_P = \{$`has wheels`, `has pantograph`, `requires infrastructure`$\}$. As $P \neq K_P$, the user is asked in line 7 if the attribute implication $\{$`requires infrastructure`$\} \rightarrow \{$`has wheels`, `has pantograph`$\}$ is true w.r.t. the formal context from Table 5.1. It is true, the user answers so, and thus in line 9 the set of implications is extended with the implication. The algorithm now moves to line 17 and $j = 4$. P' is computed as the implicational closure of the set $\{$`requires infrastructure`$\}$ and become $P' = \{$`requires infrastructure`, `has wheels`, `has pantograph`$\}$. In Table 5.3, the corresponding row for P is denoted with 0001, and the column corresponding to P' with 1011. We observe that $P <_4 P'$ does not hold, and thus next iteration of the for loop begins with $j = 3$. The implicational closure of the set $\{$`has pantograph`$\}$ is computed and $P' = \{$`has pantograph`$\}$ with the corresponding column denoted by 0010. Ta-

Table 5.4: A sample incomplete formal context about means of transportation, obtained from the incomplete formal context in Table 5.2 during the execution of the attribute exploration algorithm presented in Algorithm 5.1.

objects \ attributes	has wheels	can sail	has pantograph	requires infrastructure
train	+	−	+	+
boat	−	+	−	?
amphibian	+	+	?	?
sleight	−	−	−	−
trolleybus	+	−	+	+

ble 5.3 confirms that $P <_3 P'$, P' becomes P and next iteration of the loop begins.

The set K_P is computed in line 4 as M minus the set $\{\texttt{can sail}\}$, i.e. again $K_P = \{\texttt{has wheels}, \texttt{has pantograph}, \texttt{requires infrastructure}\}$. $P \neq K_P$, and so the user is asked if $\{\texttt{has pantograph}\} \to \{\texttt{has wheels}, \texttt{requires infrastructure}\}$. The user again confirms, as stated in Table 5.1, the set $\mathcal{L}$ in line 9 is extended with the attribute implication and now $\mathcal{L} = \{\{\texttt{requires infrastructure}\} \to \{\texttt{has wheels}, \texttt{has pantograph}\}, \{\texttt{has pantograph}\} \to \{\texttt{has wheels}, \texttt{requires infrastructure}\}\}$. Computation of the set P' now begin with $j = 4$ in line 17:

$$\begin{aligned} P' &= \mathcal{L}(\{\texttt{has pantograph}\} \cup \{\texttt{requires infrastructure}\}) \\ &= \{\texttt{has wheels}, \texttt{has pantograph}, \texttt{requires infrastructure}\} \end{aligned} \tag{5.15}$$

The corresponding column in Table 5.3 for P' is 1011 and the corresponding row for P is 0010, and we observe that $P' <_4 P$ does not hold. For $j = 3$, $P' = \mathcal{L}(\emptyset \cup \{\texttt{has pantograph}\})$ and due to the implicational closure we get the same set as before, but $P' <_3 P$ also does not hold. For $j = 2$, $P' = \mathcal{L}(\emptyset \cup \{\texttt{can sail}\}) = \{\texttt{can sail}\}$. The corresponding column is 0100, and Table 5.3 confirms that $P <_2 P'$, thus P' becomes P and the next iteration of the algorithm begins.

In line 4, the set K_P is computed using only `boat` and `amphibian` as $K_P = M \backslash \{\texttt{has pantograph}\} = \{\texttt{has wheels}, \texttt{can sail}, \texttt{requires infrastructure}\}$. $P \neq K_P$, and the user is asked in line 7 if $\{\texttt{can sail}\} \to \{\texttt{has wheels}, \texttt{requires infrastructure}\}$. The user answers no, and must provide a counterexample, i.e. construct an object that can sail but either does not have wheels or does not require infrastructure. The user decides to provide only a minimal amount of information and states that boat does not have wheels, obtaining a new incomplete formal context, depicted in Table 5.4.

During the rest of the execution of the algorithm, the following attribute implications are considered:

1. The user is asked to decide whether $\{\texttt{can sail}\} \to \{\texttt{requires infrastructure}\}$. It is not, so the user completes the partial object description of the `boat` with a statement that the `boat` does not `require infrastructure`.

2. The user is asked to decide if $\{\texttt{has wheels}\} \to \{\texttt{has pantograph}, \texttt{requires infrastructure}\}$. The user answers negatively and completes the description of amphibian with negative in-

formation about the attributes has pantograph and requires infrastructure. At this point the incomplete formal context is no longer incomplete and it looks like the formal context presented in Table 5.1.

3. The set $P = \{\texttt{has wheels}, \texttt{has pantograph}, \texttt{requires infrastructure}\}$ is considered, but $K_P = P$ and the implication $P \to K_P \backslash P$ is ignored.

4. Next in the order is the set $P = \{\texttt{has wheels}, \texttt{can sail}\}$, for which again $K_P = P$.

5. Finally, $P = M$ and the algorithm terminates.

The set $\mathcal{L}$ consists of two implications

$$\begin{aligned}\mathcal{L} = \{&\{\texttt{requires infrastructure}\} \to \{\texttt{has wheels}, \texttt{has pantograph}\},\\ &\{\texttt{has pantograph}\} \to \{\texttt{has wheels}, \texttt{requires infrastructure}\}\}\end{aligned} \tag{5.16}$$

and is the implication base for the set of all attribute implications in the formal context presented in Table 5.1.

5.5 Application of Formal Concept Analysis to completing formal ontologies

Following the concept presented in [5] we present a method to use the attribute exploration algorithm described above to complete an ontology or its part. Consider a consistent ontology $\mathcal{O}$ and denote by G the set of all named individuals in the ontology. Let M be an arbitrary, finite set of class expressions. The ontology and the sets G and M jointly constitute an incomplete formal context:

$$\mathcal{K} = \{(g, A, S) \colon g \in G \wedge A = \{C \in M \colon \mathcal{O} \models C(g)\} \wedge S = \{C \in M \colon \mathcal{O} \models \neg C(g)\}\} \tag{5.17}$$

The incomplete formal context may then be used as the input to Algorithm 5.1. Every attribute implication $L \to R$ discovered in the process may be expressed as a general concept inclusion, i.e. for $L = \{L_1, \ldots, L_n\}$, $R = \{R_1, \ldots, R_m\}$, the implication can be rewritten as $L_1 \sqcap \ldots \sqcap L_n \sqsubseteq R_1 \sqcap \ldots \sqcap R_m$. In order to maintain consistency between the incomplete formal context and the ontology and leverage a reasoner, it is useful to add the implication base $\mathcal{L}$ to the ontology, i.e. whenever the set $\mathcal{L}$ is to be extended in line 9 of Algorithm 5.1, the corresponding general concept inclusion is added to the ontology. The same goes for counterexamples: whenever the user provides a counterexample, it is asserted to the ontology.

The assumption of the algorithm must be fulfilled: the user must know the completion of the incomplete formal context and use it to answer the questions posed by the algorithm. This completion corresponds to a model of the ontology and as the completion can not change during the execution of the algorithm, neither the underlying model can. In other words, the considered model must be a model of every ontology obtained during the execution of the algorithm, including the one which is provided in the input.

After the execution of the algorithm every possible general concept inclusion formed using only the intersection operator and the class expressions of M either is asserted in the obtained ontology, logically follows from the obtained ontology or asserting it would make the ontology inconsistent. The class expressions in the set M are the way to guide the algorithm and decide which part of the ontology is relevant to the user.

5.6 Classification task in Formal Concept Analysis

A classification task is to assign an object, described by a vector of *features*, to one of a finite number of predefined *classes*. A special case, when there are only two classes, is called *binary classification task*. A function that realizes such a task is called a *classifier*. A machine learning approach to obtain a classifier is to *learn* it from *training examples*, that is a set of *labelled examples*, i.e. pairs $(\mathbf{x}, y)$ such that $\mathbf{x}$ is a vector of features and y is the expected class for the object described by $\mathbf{x}$. A classification algorithm is an algorithm that, given a set of training examples, finds the best function f from a space of functions such that a *classification error* on the training set is minimized. The space of functions depends on the particular classification algorithm. We call a result of classification *correct* if $y = f(\mathbf{x})$ and *incorrect* otherwise. A possible formulation of the classification error is *misclassification rate*, defined as the ratio of the number of examples classified incorrectly to the overall number of examples.

The main problem of the attribute exploration algorithm is that it explores the user more than the attributes: the user must answer a huge number of questions, with the worst-case number being exponential w.r.t. the number of attributes. We proposed in [71, 76] a method to use Linked Data and machine learning to support the user in this tedious task. The idea is not to completely replace the user, making the method fully automated, but rather answer a question when Linked Data provide enough evidence to either confirm or reject the question, and ask the user otherwise. Such an approach introduces a stack of experts to the algorithm: a reasoner, a classifier and the user. First, the algorithm consults the reasoner to check whether the attribute implication is already entailed by the ontology. If it is not, the classifier is run and if its knowledge is too limited, it redirects the question to the human expert, who must provide a conclusive answer.

To enable the redirections, it is not enough for the classifier to return a crisp, binary answer: accept or reject the attribute implication. Instead, the classifier should estimate a probability p of the implication being correct. If the probability is above or equal to some threshold θ_a, then the implication is accepted without consulting the user. If the probability is below or equal to some threshold θ_r, then a counterexample is automatically generated and the implication is rejected. Otherwise, i.e. when $\theta_r < p < \theta_a$, the classifier is unsure of the decision and the user is to be asked. When $\theta_r = 1 - \theta_a$, the algorithm is fully automated and the user is never asked.

A formal algorithm realizing the described idea is presented in Algorithm 5.2. The input to the algorithm is an ontology $\mathcal{O}$ and a set of class expressions M, which are passed to the attribute exploration algorithm presented in Algorithm 5.1. The algorithm also requires two thresholds θ_a and θ_r, that are the thresholds for the probability of, respectively, accepting and rejecting an attribute implication. The result of the execution of the algorithm is the ontology $\mathcal{O}$ completed w.r.t. the set of all general concept inclusions that can be constructed from the elements of the set M using the intersection operator.

To fully realize the algorithm, the decision about the classification algorithm must be made. First, not all classification algorithms are able to provide classifiers estimating the probabilities well. For example, classifier learned with the *Naïve Bayes* classification algorithm are known to be bad estimators, as their learning is concerned only with the outcome of classification, not the probabilities, and thus mistakes in probability estimation are not penalized as long as the highest probability is assigned to the correct class [103]. Conversely, widely recognized and well-known *Support Vector Machines* (SVMs) in a typical setup do not provide any probability estimation, only a crisp answer, but using additional algorithm on top of a SVM can generate the probability [102]. Finally, *Logistic Regression*, also a widely recognized and popular tool, is constructed around estimating the probabilities and thus is a perfect choice in our case [1].

Data: An ontology $\mathcal{O}$, a set of class expressions M, two thresholds θ_a and θ_r
Result: The ontology $\mathcal{O}$ completed w.r.t. the set of class expressions M

Generate an incomplete formal context from the ontology $\mathcal{O}$ and the set M
while *it is possible to generate a new implication using Algorithm 5.1* **do**
  Generate the new implication $\{L_1, \ldots, L_m\} \to \{R_1, \ldots, R_n\}$
  for $i \leftarrow 1, \ldots, n$ **do**
    if $\{L_1, \ldots, L_m\} \to \{R_i\}$ *is not refuted* **then**
      $C \leftarrow L_1 \sqcap \ldots \sqcap L_m \sqsubseteq R_i$
      if $\mathcal{O} \models C$ **then**
        $\mathcal{L} \leftarrow \mathcal{L} \cup \{\{L_1, \ldots, L_m\} \to R_i\}$
      end
      else
        Use the classifier to compute probability p of the implication being correct
        if $p \geq \theta_a$ **then**
          $\mathcal{L} \leftarrow \mathcal{L} \cup \{\{L_1, \ldots, L_m\} \to R_i\}$
          $\mathcal{O} = \mathcal{O} \cup \{C\}$
        end
        else if $1 - p \geq \theta_r$ **then**
          Generate a new anonymous individual a
          $\mathcal{O} = \mathcal{O} \cup \{L_1(a), \ldots, L_m(a), \neg R_i(a)\}$
        end
        else
          Ask the user if the axiom C should follow from the ontology
          Add the answer to the classifier
          if *yes* **then**
            $\mathcal{L} \leftarrow \mathcal{L} \cup \{\{L_1, \ldots, L_m\} \to R_i\}$
            $\mathcal{O} = \mathcal{O} \cup \{C\}$
          end
          else
            Ask the user to alter the ABox to provide a counterexample
          end
        end
      end
    end
  end
end

Algorithm 5.2: An algorithm realizing the idea of a stack of experts

Usually, functions considered by a classification algorithm expect a vector of numerical features, while an attribute implication is a pair of sets of attributes. To address this discrepancy, we propose to use a set of measures used for association rules, but computed not on the ontology itself, but rather on a relevant Linked Data dataset, accessible through a SPARQL endpoint. To query the dataset, we must obtain a mapping between the elements of the set of class expressions M and SPARQL basic graph patterns. The baseline approach is to use a Turtle serialization of an attribute and use it directly as a part of SPARQL query. As the vocabulary used by the ontology may not be the same as the vocabulary used in the Linked Data dataset, ontology mapping techniques may be employed here to achieve better results. Finally, the user should be asked to verify and correct the mappings. We use $\mu(\texttt{?x}, C)$ to denote the mapping for the attribute C, where `?x` is the variable used in the mapping. For example, for the attribute `has pantograph` a viable mapping to DBpedia is:

$$\mu(\texttt{?x}, \texttt{has pantograph}) = \texttt{?x dbp:collectionmethod dbr:Pantograph_(rail)} \tag{5.18}$$

In general, there are various possible representations for this attribute as a class expression, for

example it could be:

- a data property with a boolean value: `hasPantograph` VALUE `"true"^^xsd:boolean`;
- a class: `HasPantograph`
- an object property with a fixed IRI: `dbp:collectionmethod` VALUE `dbr:Pantograph_(rail)`

Whichever is the case, the final mapping must reflect the structure of the Linked Data dataset in use rather than the modelling principles of the considered ontology.

Consider an attribute implication $\{L_1, \ldots, L_m\} \to \{R_i\}$ and let p be the number of objects assigned all the attributes $L_1, \ldots, L_m$, c be the number of objects assigned the attribute R_i and pc be the number of objects assigned all the attributes $L_1, \ldots, L_m$ and the attribute R_i. Following the naming convention of [12], we use the following measures as the numeric features of an implication:

coverage p

prevalence c

support pc

recall

$$\frac{pc}{c}$$

confidence

$$\frac{pc}{p}$$

Once the mappings are established, we may proceed with the computation of the features. The general idea is to generate SPARQL queries using the following template:

```
SELECT (COUNT(DISTINCT ?x) as ?c)
WHERE { ?x ... }
```

During specialization of the template, the ellipsis is replaced by an appropriate combination of mappings. To compute the values p, c and pc, we pose three queries to the SPARQL endpoint:

- Computing p as the binding to the variable `?p`

```
SELECT (COUNT(DISTINCT ?x) as ?p)
WHERE
{
    μ(?x, L1)
    ...
    μ(?x, Lm)
}
```

- Computing c as the binding to the variable `?c`

```
SELECT (COUNT(DISTINCT ?x) as ?p)
WHERE
{
    μ(?x, Ri)
}
```

Table 5.5: A summary of the features used to describe an attribute implication.

features computed using the SPARQL endpoint	
coverage	p
prevalence	c
support	pc
recall	$\frac{pc}{c}$
confidence	$\frac{pc}{p}$
features computed using the ontology and a reasoner	
coverage	p
prevalence	c
support	pc
recall	$\frac{pc}{c}$
confidence	$\frac{pc}{p}$
syntactic features	
normalized length of the set of premises	$\frac{m}{\lvert M\rvert}$

- Computing pc as the binding to the variable `?pc`

```
SELECT (COUNT(DISTINCT ?x) as ?pc)
WHERE
{
    μ(?x, L1)
    ...
    μ(?x, Lm)
    μ(?x, Ri)
}
```

This set of five features can be further extended with a set of similar features, but based on the ontology itself. The very same measures can be computed using a reasoner and the ontology, without usage of the mappings and the SPARQL endpoint. Finally, the shape of the implication itself may be an important feature, and thus we add, as a feature, the normalized length of the set of premises $\{L_1, \ldots, L_m\}$, i.e. $\frac{m}{|M|}$. Overall, this yields 11 numerical features, presented in Table 5.5, which can be used with any general-purpose classification algorithm.

One must also notice that the set of training examples is created during the work of the algorithm and is not available before the classifier is first used. One approach is to retrain the classifier every time a new example is available. The approach works with any classification algorithm, but for some of them it may be slow. The other approach is to use algorithms able to learn the classifier incrementally, only taking into account the example that just arrived [92]. In particular, it is possible to alter Logistic Regression to enable incremental learning [43].

Consideration must be given to the fact that the classes may be heavily imbalanced and thus the classifier may tend to favour one class over the other one, to the point that it completely ignores the minority class. Observe that accepting an incorrect attribute implication (i.e., adding an incorrect subsumption to the ontology) has much more serious consequences than rejecting a correct implication. Due to the reasoning consequences, adding an incorrect subsumption to the ontology may immediately yield it completely unsuitable, e.g. by making it inconsistent. Conversely, omitting a valid subsumption does not increase its completeness, thus even in the worst case it does not make it less suitable for its purpose than it already was in the beginning. To

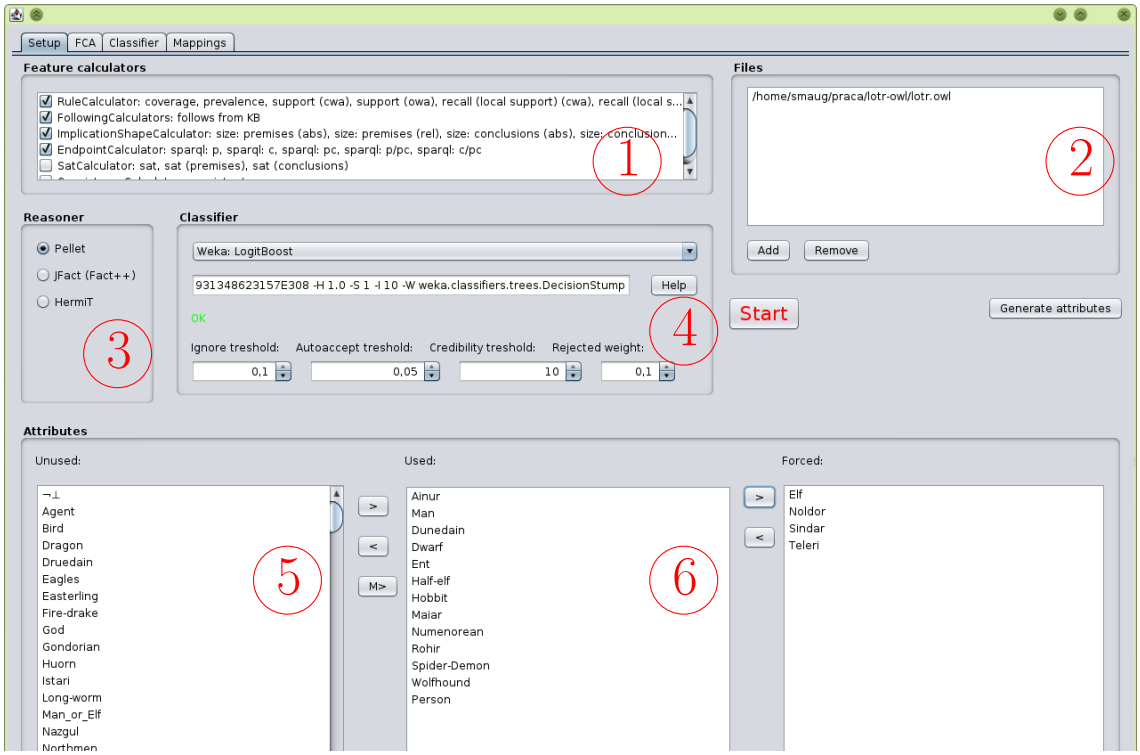


Figure 5.2: The Setup tab of the tool for completing ontologies using the attribute exploration algorithm with a stack of experts. The user configures there: features to use (1), files with the ontology (2), a reasoner (3) and a classification algorithm (4). From the list of class expressions (5), the user selects the set M of attributes (6).

remedy this issue, we may assign weights to the *learning examples* and use cost-sensitive learning, an approach supported by most of the classification algorithms [54].

5.7 Software tool

We have implemented Algorithm 5.2 in *Java* and published the implementation to *GitHub*: `https://github.com/jpotoniec/FCA-ML`. In the user interface, there are four tabs which separate different functions:

Setup tab presented in Figure 5.2, where the user selects the class expressions to the set of attributes M, load the ontology and configures the classification algorithm.

FCA tab presented in Figure 5.3 and Figure 5.4, where the user interacts with the algorithm.

Classifier tab presented in Figure 5.5. The user can see there the performance of the classifier, the training examples and the attribute implications considered so far.

Mappings tab presented in Figure 5.6, where the mappings to the Linked Data dataset can be specified.

The application uses OWL API[1] [37] to process the ontology and to interact with a reasoner and relies on *Weka*[2] to provide classification algorithms.

[1] `http://owlapi.sourceforge.net`
[2] `http://www.cs.waikato.ac.nz/ml/weka/`

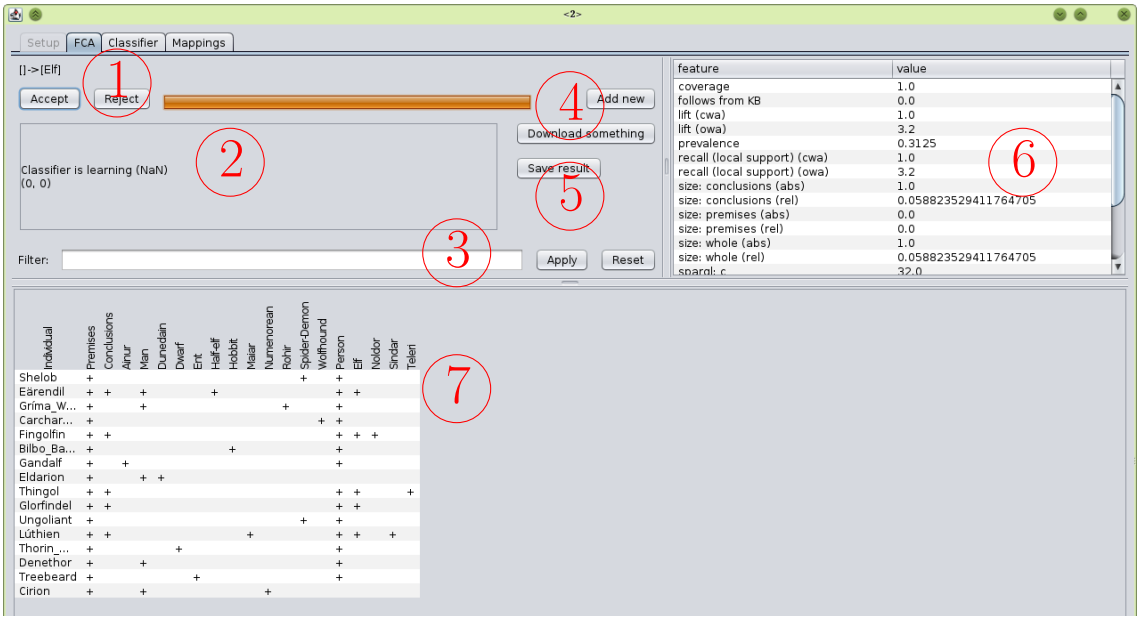


Figure 5.3: The FCA tab of the tool for completing ontologies using the attribute exploration algorithm with a stack of experts, right after the algorithm was started. It presents the current implication and possible decisions (1), the status of the classifier (2), the search box for finding individuals by name (3), the buttons for extending incomplete formal context with new individuals (4), the button to export already discovered implications to a file (5), the features of the current implication (6) and the incomplete formal context (7).

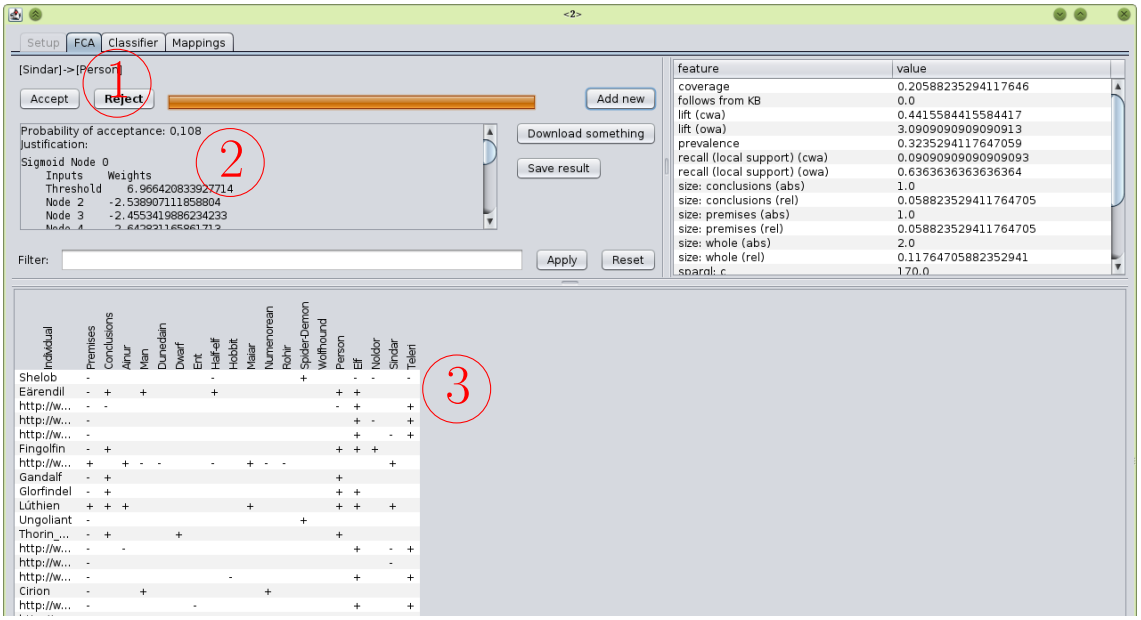


Figure 5.4: The FCA tab of the tool for completing ontologies using the attribute exploration algorithm with a stack of experts during the run of the algorithm. The answer *reject* is suggested with the bold font by the classifier (1), the parameters of the classifier are displayed (2) and the incomplete formal context was altered and new individuals were added (3).

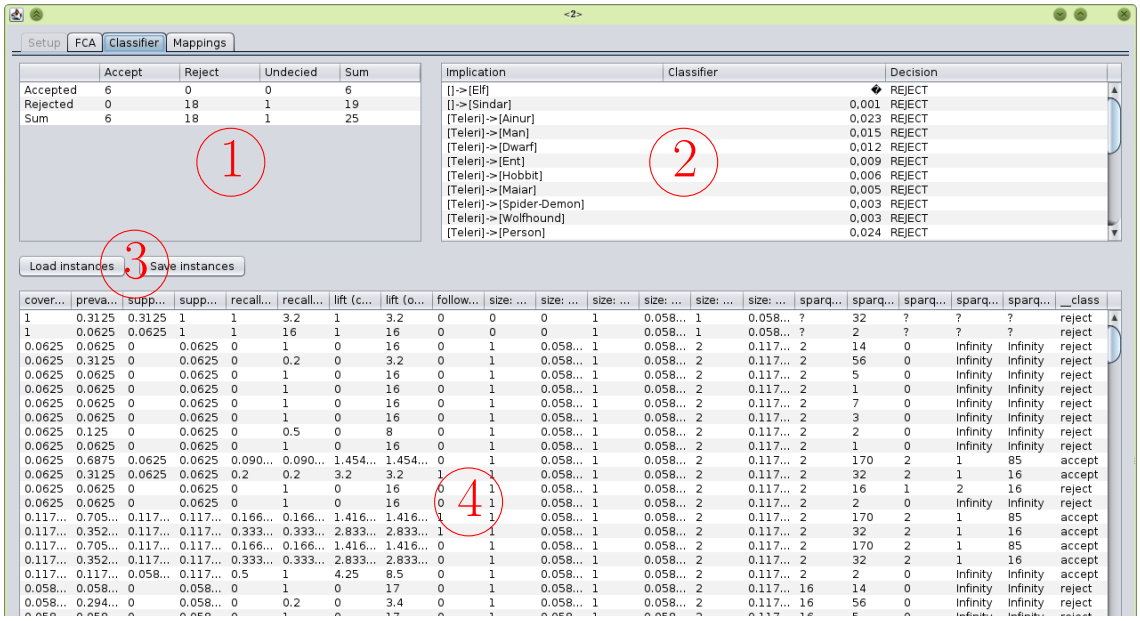


Figure 5.5: The Classifier tab of the tool for completing ontologies using the attribute exploration algorithm: confusion matrix comparing the decisions suggested by the classifier with the decisions of the user (1), the history of the implications, along with the probabilities estimated by the classifier and the final decision (2), the buttons for saving and loading the training examples (3), the training examples, that is implications transformed to vectors of numerical features and labelled with a correct decision (4).

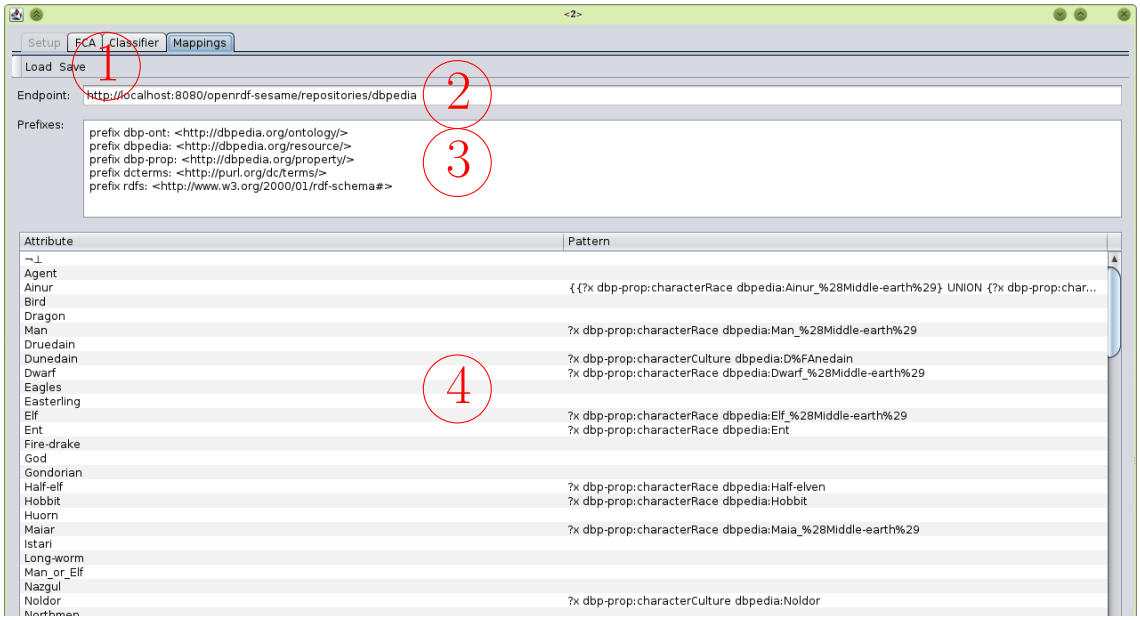


Figure 5.6: The Mappings tab of the tool for completing ontologies using the attribute exploration algorithm: the buttons for loading and saving mappings (1), the address of the SPARQL endpoint (2), the prefixes used in mappings (3), the mappings (4).

Chapter 6

Swift Linked Data Miner

Swift Linked Data Miner is an algorithm developed specifically to mine OWL 2 EL class expressions directly from Linked Data, using only a SPARQL endpoint as a way to access the data. A mined class expression is then used as a right-hand side of a *class inclusion axiom*. The rough idea of the algorithm is to download a relevant part of the RDF graph in a SPARQL endpoint, process it quickly and proceed with downloading the next relevant part. To enable efficient processing, an appropriate organization of data in the memory is required. We proposed SLDM in [72] and presented its implementation in [75].

6.1 Data retrieval from a remote RDF graph

Lets start the consideration with a set of IRIs $\mathcal{I}$. The triples describing these IRIs must be first downloaded from the SPARQL endpoint and then organized in memory. We are interested in mining class expressions, so it is enough to download these triples where the subject is a member of the set $\mathcal{I}$. A naïve approach would be to pose the following SPARQL query for every IRI $s \in \mathcal{I}$:

```
SELECT ?p ?o
WHERE
{
    s ?p ?o .
}
```

Obviously, this would lead to posing $|\mathcal{I}|$ queries to a SPARQL endpoint, and quite possibly to opening the same number of network connections. In case of larger sets, e.g. consisting of 50000 IRIs, such a behaviour would be completely unacceptable: the algorithm would spend too much time posing queries and waiting for responses, instead of processing the actual data.

As an example, consider the following set of IRIs $\mathcal{I}^{ex}$, consisting of five DBpedia IRIs referring to various locomotives used in Poland: $\mathcal{I}^{ex} = \{$`dbr:PKP_class_EU07`, `dbr:PKP_class_SU46`, `dbr:PKP_class_OK127`, `dbr:PKP_class_SM42`, `dbr:PKP_class_ET22`$\}$. A sample query to retrieve all the triples with `dbr:PKP_class_EU07` as subject would be

```
SELECT ?p ?o
WHERE
{
    dbr:PKP_class_EU07 ?p ?o .
}
```

The other end of the spectrum would be to download all the triples from the SPARQL endpoint and perform the selection locally, i.e. pose the query `SELECT ?s ?p ?o WHERE { ?s ?p ?o . }` and download all the data at once. This is a feasible solution only for a SPARQL endpoint containing a very small RDF graph, which can be reasonably retrieved this way.

Fortunately, there exists a spectra of middle grounds due to the `VALUES` keyword, which allows specification of bindings for a given variable. Consider the following query template:

```
SELECT ?s ?p ?o
WHERE
{
    ?s ?p ?o .
    VALUES ?s {ℐ}
}
```

This query selects all the triples in the SPARQL endpoint such that their subjects are members of the set $\mathcal{I}$. The query for the example set $\mathcal{I}^{ex}$ is

```
SELECT ?s ?p ?o
WHERE
{
    ?s ?p ?o .
    VALUES ?s {dbr:PKP_class_EU07 dbr:PKP_class_SU46 dbr:PKP_class_OK127
               dbr:PKP_class_SM42 dbr:PKP_class_ET22}
}
```

In theory, this is a perfect solution, which enables us to download all the required triples using only a single SPARQL query and thus minimizing the required amount of network communication and giving the SPARQL endpoint the maximal capability of optimizing the query. Unfortunately, there is a caveat: the configuration of the SPARQL endpoint may enforce it to close the connection e.g. after sending a given number of triples or after processing the query for so long. In this, we are faced with a trade-off between the number of posed queries and their complexity.

To account for this, we introduce a parameter *split* and split the set $\mathcal{I}$ into subsets $\mathcal{I}_1, \ldots, \mathcal{I}_{nqueries}$ such that:

- each subset consists of at most *split* members:

$$\forall i \in \{1, \ldots, nqueries\} \ |\mathcal{I}_i| \leq split \tag{6.1}$$

- the number of subsets is minimal, that is each subset, except at most one, has exactly *split* members:

$$nqueries = \left\lceil \frac{|\mathcal{I}|}{split} \right\rceil \tag{6.2}$$

- the subsets cover all the IRIs from the set $\mathcal{I}$:

$$\bigcup_{i=1}^{nqueries} \mathcal{I}_i = \mathcal{I} \tag{6.3}$$

- no IRI is in more than one set:

$$\forall i, j \in \{1, \ldots, nqueries\} \ (i \neq j \rightarrow \mathcal{I}_i \cap \mathcal{I}_j = \emptyset) \tag{6.4}$$

Then, the query for each of the subsets is constructed and posed to the SPARQL endpoint, yielding a disjoint set of triples, that can be finally concatenated to obtain the same set of triples as in the query without splitting.

Recall the set $\mathcal{I}^{ex}$ and assume $split = 3$. There are two subsets to be created, one of three items and the other one of two. For example, the corresponding queries could be:

```
SELECT ?s ?p ?o
WHERE
{
    ?s ?p ?o .
    VALUES ?s {dbr:PKP_class_EU07 dbr:PKP_class_SU46 dbr:PKP_class_OKl27 }
}

SELECT ?s ?p ?o
WHERE
{
    ?s ?p ?o .
    VALUES ?s {dbr:PKP_class_SM42 dbr:PKP_class_ET22}
}
```

The parameter *split* allows for covering the whole range of possibilities, from a single query for an IRI (i.e., $split = 1$) to a single query for the whole set of IRIs (i.e., $split = |\mathcal{I}|$).

6.2 Sampling strategies

If the set of IRIs is large, even such an iterative approach can take too much time and be too demanding for the SPARQL endpoint. In such a case, sampling from the set of IRIs $\mathcal{I}$ should help to facilitate the issues. We propose three sampling strategies and denote by n the desired sample size (e.g., 1000). We assume that $|\mathcal{I}| > n$ and if the assumption is not met, the whole set $\mathcal{I}$ should be used without sampling.

The *uniform strategy* is the simplest of the three. The set $\mathcal{I}$ is shuffled and the first n IRIs are considered, while the rest is discarded. This is an equivalent of simple random sampling without replacement and does not require posing any additional queries to the endpoint.

The *predicates counting strategy* makes more informed choice, by using the number of different predicates in triples of a given IRI as a probability estimate. First, SPARQL queries constructed from the following template are posed.

```
SELECT ?s (COUNT(DISTINCT ?p) AS ?c)
WHERE
{
    ?s ?p [] .
    VALUES ?s  {...}
}
GROUP BY ?s
```

The ellipsis is replaced by subsets of the set of IRIs, depending on the *split* parameter, as described earlier. The queries assign every IRI s an integer value c, which is the number of different predicates in triples in the graph such that s is their subject. Then, the values c are normalized by division by

their sum, i.e. now the sum of all c is equal to 1 and c can be treated as a probability distribution. Using this probability distribution, a sample of size n is drawn without replacement.

The third strategy, called the *triples counting strategy* operates on a similar basis, but it counts different triples instead of different predicates. First, the following template is used to pose SPARQL queries:

```
SELECT ?s (COUNT(?p) AS ?c)
WHERE
{
    ?s ?p ?o .
    VALUES ?s  {...}
}
GROUP BY ?s
```

In comparison with the previous template,ishas been replaced by the variable `?o`. The queries constructed from this template assign to each IRI s an integer value c that is the number of triples in the graph that have s in the subject position. The rest of the strategy is the same: c is normalized to create a probability distribution, which is then used to perform sampling.

6.3 Data organization in memory

The in-memory organization of the retrieved triples must facilitate their quick and efficient exploration. It is thus desirable to organize the triples in a *three-level index* such that predicates are on the first level, the objects are on the second level and the subjects are on the third level. This organization is crucial from the point of view of efficiency, but it does not influence the soundness of the algorithm.

Let $\mathcal{T} = \{(s, p, o)\}$ be a set of triples and $P = \{p_1, \ldots, p_l\}$ the set of all predicates occurring in the triples. The first level of the index is a map (e.g., a hash map [20]) such that the keys are the predicates from the set P and the values are pointers to the second level. Let us denote by $O_p = \{o_1, \ldots, o_m\}$ the set of all objects occurring in the triples of $\mathcal{T}$ and having p as the predicate:

$$O_p = \{o \colon \exists s \, (s, p, o) \in \mathcal{T}\} \tag{6.5}$$

The second level, pointed by the first level for the predicate p, maps each object $o \in O_p$ to a vector in the third level. Let $S_{p,o} = \{s_1, \ldots, s_n\}$ be the set of all subjects occurring in the triples of $\mathcal{T}$ such that p is the predicate and o is the object:

$$S_{p,o} = \{s \colon (s, p, o) \in \mathcal{T}\} \tag{6.6}$$

The vector in the third level of the index, pointed by p in the first level and by o in the second level, consists of all the subjects of $S_{p,o}$ in an arbitrary order. An abstract example of such an index is presented in Figure 6.1.

Algorithm 6.1 presents a function used to build the three-level index from a set of triples. The index is first initialized to an empty map in line 2 and then the iteration over the triples commences in line 3. For every triple (s, p, o), the function checks that the predicate p is present in the index in line 4, and if it is not, an appropriate entry with an empty map for the second level is added in line 5. Next, it checks in line 7 whether the object o is present in the second level for the predicate p. If it is not, in line 8 the object o is added with an empty vector to store the third level of the index. Finally, the subject of the triple is added to the appropriate third level of the index in line 10. After the loop completes, the index is returned in line 12.

```
1  function BuildIndex(T)
2      I ← an empty map
3      foreach s, p, o ∈ T do
4          if p ∉ I then
5              I[p] ← an empty map
6          end
7          if o ∉ I[p] then
8              I[p][o] ← ∅
9          end
10         I[p][o] ← I[p][o] ∪ {s}
11     end
12     return I
13 end
```

Algorithm 6.1: A function to build the three-level index given a set of triples $\mathcal{T}$

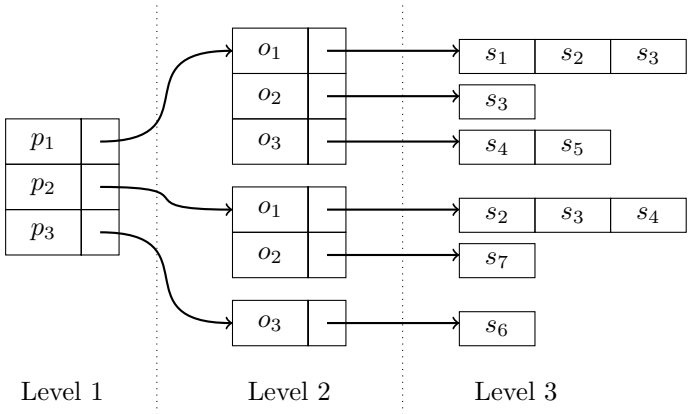


Figure 6.1: The three-level index for the set of triples $\mathcal{T} = \{(s_1, p_1, o_1), (s_2, p_1, o_1), (s_3, p_1, o_1), (s_3, p_1, o_2), (s_4, p_1, o_3), (s_5, p_1, o_3), (s_2, p_2, o_1), (s_3, p_2, o_1), (s_4, p_2, o_1), (s_7, p_2, o_2), (s_6, p_3, o_3)\}$. The first level consists of a single map from all the predicates, the second of maps from the objects, one map for every predicate in the first level, and the third of vectors of the subjects, one for every pair predicate–object. The arrows depict the pointers between the levels. To retrieve all the triples with the predicate p_2 and object o_1 one must find p_2 in the first level, follow the pointer to the second level, find o_1 in the map there and follow the pointer to the vector in the third level, finally to discover that there are three such triples, with subjects s_2, s_3 and s_4.

Table 6.1: A sample RDF graph, being a small excerpt of DBpedia and concerning five locomotives used in Poland: ET22, EU07, OKl27, SM42 and SU46.

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix dbp: <http://dbpedia.org/property/> .
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix dbc: <http://dbpedia.org/resource/Category:> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

dbr:PKP_class_ET22          dbo:builder        dbr:Pafawag                    ;
                            dbo:numberBuilt    "1183"^^xsd:integer            ;
                            dct:subject        dbc:3000_V_DC_locomotives      ,
                                               dbc:Co-Co_locomotives          ;
                            rdf:type           dbo:Locomotive                 ,
                                               dbo:MeanOfTransportation       .
dbr:PKP_class_EU07          dbo:builder        dbr:H._Cegielski_-_Poznań_S.A. ,
                                               dbr:Pafawag                    ;
                            dbo:numberBuilt    "483"^^xsd:integer             ;
                            dct:subject        dbc:3000_V_DC_locomotives      ,
                            rdf:type           dbo:Locomotive                 ,
                                               dbo:MeanOfTransportation       .
dbr:PKP_class_OKl27         dbo:builder        dbr:H._Cegielski_-_Poznań_S.A. ;
                            dbo:numberBuilt    "122"^^xsd:integer             ;
                            dct:subject        dbc:2-6-2T_locomotives         ;
                            rdf:type           dbo:Locomotive                 ,
                                               dbo:MeanOfTransportation       .
dbr:PKP_class_SM42          dbo:builder        dbr:Fablok                     ;
                            dbo:engineType     dbr:V8_engine                  ;
                            dbo:numberBuilt    "1822"^^xsd:integer            ;
                            dct:subject        dbc:Bo-Bo_locomotives
                            rdf:type           dbo:Locomotive                 ,
                                               dbo:MeanOfTransportation       .
dbr:PKP_class_SU46          dbo:builder        dbr:H._Cegielski_-_Poznań_S.A. ;
                            dbo:numberBuilt    "54"^^xsd:integer              ;
                            dct:subject        dbc:Co-Co_locomotives          ;
                            rdf:type           dbo:Locomotive                 ,
                                               dbo:MeanOfTransportation       .
dbc:3000_V_DC_locomotives   rdf:type           skos:Concept                   .
dbc:2-6-2T_locomotives      rdf:type           skos:Concept                   .
dbc:Bo-Bo_locomotives       rdf:type           skos:Concept                   .
dbc:Co-Co_locomotives       rdf:type           skos:Concept                   .
```

Recall the sample set of IRIs $\mathcal{I}^{ex}$. Table 6.1 contains a small RDF graph about these five IRIs and Figure 6.2 presents the three-level index corresponding to the triple from the graph having one of the five IRIs as the subject.

6.4 Basic definitions

6.4.1 Pattern

Definition 6.4.1 (pattern)**.** A pattern is an arbitrary OWL 2 EL class expression or *data range*.

Intuitively, we say that an individual or a literal a *matches* the pattern C w.r.t. the RDF

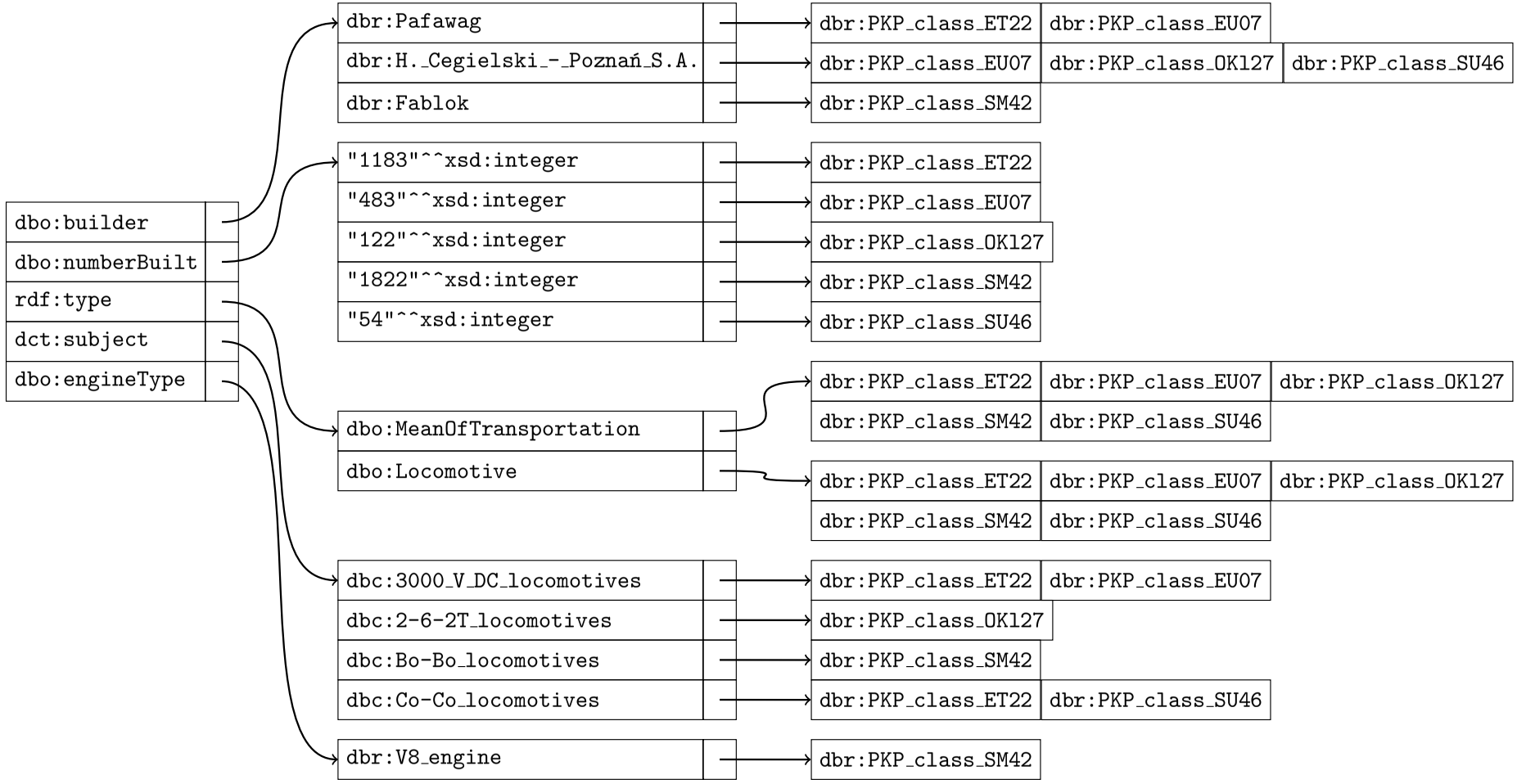


Figure 6.2: The three-level index corresponding to the sample set of IRIs $\mathcal{I}^{ex}$ = {dbr:PKP_class_EU07, dbr:PKP_class_SU46, dbr:PKP_class_OK127, dbr:PKP_class_SM42, dbr:PKP_class_ET22} and built from the relevant triples of the RDF graph presented in Table 6.1.

graph $\mathbb{G}$ if the graph contains triples that support the assertion $C(a)$. In order to formally define this notion, we define the *matching function* $\mu_{\mathbb{G}}$ such that $\mu_{\mathbb{G}}(a, C) = 1$ if a matches C w.r.t. the graph $\mathbb{G}$ and $\mu_{\mathbb{G}}(a, C) = 0$ otherwise. Whenever the considered RDF graph is obvious from the context, we omit the subscript and write $\mu(a, C)$.

The matching function is defined inductively, starting from the simplest expressions. We first concentrate on OWL 2 EL *data ranges*. Let D be a datatype and l a literal. It is reasonable to assume that the literal l matches the pattern expressed as the datatype D if, and only if, l^{LT} belongs to the *value space* of D:

$$\mu_{\mathbb{G}}(l, D) = \begin{cases} 1 & l^{LT} \in \text{ value space of } D \\ 0 & \text{otherwise} \end{cases} \tag{6.7}$$

For example, 1 belongs to the value space of `xsd:nonNegativeInteger` and so

$$\mu(\texttt{"1"\^\^xsd:integer}, \texttt{xsd:nonNegativeInteger}) = 1 \tag{6.8}$$

while the text "Lore ipsum" does not belong to the value space of `xsd:integer` thus

$$\mu(\texttt{"Lore ipsum"\^\^xsd:string}, \texttt{xsd:integer}) = 0 \tag{6.9}$$

Consider an enumeration of literals $\{m\}$. We want a literal l to match the pattern if, and only if, l and m correspond to the same value, i.e. l^{LT} and m^{LT} are the same values:

$$\mu_{\mathbb{G}}(l, \{m\}) = \begin{cases} 1 & l^{LT} = m^{LT} \\ 0 & \text{otherwise} \end{cases} \tag{6.10}$$

For example,

$$\mu(\texttt{"1"\^\^xsd:nonNegativeInteger}, \{\texttt{"1"\^\^owl:real}\}) = 1 \tag{6.11}$$

as both of the literals are interpreted as the number 1, while

$$\mu(\texttt{"1"\^\^xsd:nonNegativeInteger}, \{\texttt{"1"\^\^xsd:string}\}) = 0 \tag{6.12}$$

as the second argument is interpreted as the text *1*.

Now let D and E be OWL 2 EL data ranges and consider the data range $D \text{ AND } E$. A literal l should match the pattern $D \text{ AND } E$ if, and only if, it matches both of the data ranges. What it exactly means for a literal l to match a data range D or E is defined by the matching function, thus an inductive definition is required: l matches the pattern $D \text{ AND } E$ if, and only if, $\mu(l, D) = 1$ and $\mu(l, E) = 1$:

$$\mu_{\mathbb{G}}(l, D \text{ AND } E) = \mu(l, D) \cdot \mu(l, E) \tag{6.13}$$

For example,

$$\mu(\texttt{"1"\^\^xsd:integer}, \texttt{xsd:nonNegativeInteger AND owl:real}) = \mu(\texttt{"1"\^\^xsd:integer}, \texttt{xsd:nonNegativeInteger}) \cdot \mu(\texttt{"1"\^\^xsd:integer}, \texttt{owl:real}) = 1 \cdot 1 = 1 \tag{6.14}$$

while

$$\mu(\texttt{"1"\^\^xsd:integer}, \texttt{xsd:string AND owl:real}) = \mu(\texttt{"1"\^\^xsd:integer}, \texttt{xsd:string}) \cdot \mu(\texttt{"1"\^\^xsd:integer}, \texttt{owl:real}) = 0 \cdot 1 = 0 \tag{6.15}$$

As we are interested only in the superclass expressions, the analysis by Magka et al. [57] enables us to extend the set of considered data ranges with inequality facets over numeric datatypes, e.g.,

to define a data range consisting of all integers greater than 5 `xsd:integer[>=5]`. Moreover, Sperberg-McQueen et al. [89] defined the datatypes for time representation `xsd:dateTime` and `xsd:dateTimeStamp` as being internally numbers and thus allowed us to push the boundary even further. Strictly speaking, by doing so we leave the OWL 2 EL profile, but we do not lose its most interesting property, i.e. tractable time reasoning.

Let D be one of the following datatypes: `owl:real`, `owl:rational`, `xsd:decimal`, `xsd:integer`, `xsd:dateTime`, `xsd:dateTimeStamp` and M be a literal of the type D. A data range $D[<= M]$ is a limitation of the value space of the datatype D to the values no greater than M. The formal semantics of $D[<= M]$ is given by the following equation

$$(D[<= M])^{DT} = \{v \in D^{DT} : v \leq M^{LT}\} \tag{6.16}$$

From this, we can define the corresponding matching function. A literal l should match the pattern $D[<= M]$ if it matches the datatype D and the relation between the values of l and M is satisfied:

$$\mu_{\mathbb{G}}(l, D[<= M]) = \begin{cases} \mu_{\mathbb{G}}(l, D) & l^{LT} \leq M^{LT} \\ 0 & \text{otherwise} \end{cases} \tag{6.17}$$

For example, $10 \leq 20$ and so

$$\mu(\texttt{"10"\^\^owl:real}, \texttt{xsd:integer}[<= 20]) = \mu(\texttt{"10"\^\^owl:real}, \texttt{xsd:integer}) = 1 \tag{6.18}$$

while $10 \not\leq 5$ and thus

$$\mu(\texttt{"10"\^\^owl:real}, \texttt{xsd:integer}[<= 5]) = 0 \tag{6.19}$$

Now let D be one of the following datatypes: `xsd:nonNegativeInteger`, `owl:real`, `owl:rational`, `xsd:decimal`, `xsd:integer`, `xsd:dateTime`, `xsd:dateTimeStamp`, i.e. all the datatypes listed for the data range $D[<= M]$ plus the datatype `xsd:nonNegativeInteger`. Let m be a literal of the datatype D. A data range $D[>= m]$ is a limitation of the value space of the datatype D to the values no less than m. The formal semantics of $D[>= m]$ is given by the following equation

$$(D[>= m])^{DT} = \{v \in D^{DT} : v \geq m^{LT}\} \tag{6.20}$$

Following the same idea as previously, we can define the corresponding matching function as:

$$\mu_{\mathbb{G}}(l, D[>= m]) = \begin{cases} \mu_{\mathbb{G}}(l, D) & l \geq m \\ 0 & \text{otherwise} \end{cases} \tag{6.21}$$

For example, as $10 \geq 5$

$$\mu(\texttt{"10"\^\^owl:real}, \texttt{xsd:integer}[>= 5]) = \mu(\texttt{"10"\^\^owl:real}, \texttt{xsd:integer}) = 1 \tag{6.22}$$

while $10 \not\geq 20$ and so

$$\mu(\texttt{"10"\^\^owl:real}, \texttt{xsd:integer}[>= 20]) = 0 \tag{6.23}$$

This concludes the definition of the matching function for OWL 2 EL data ranges and we now move to OWL 2 EL class expressions, starting with the simplest: assume that C is a class. An individual a should match the pattern C w.r.t. the RDF graph $\mathbb{G}$ if, and only if, the graph contains information that a belongs to the class C, i.e. there is a triple $(a, \texttt{rdf:type}, C)$ in the graph:

$$\mu_{\mathbb{G}}(a, C) = \begin{cases} 1 & (a, \texttt{rdf:type}, C) \in \mathbb{G} \\ 0 & \text{otherwise} \end{cases} \tag{6.24}$$

The sample RDF graph from Table 6.1 contains the triple (`dbr:PKP_class_ET22`, `rdf:type`, `dbo:Locomotive`) and so

$$\mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dbo:Locomotive}) = 1 \tag{6.25}$$

while it does not contain the triple (`dbr:PKP_class_ET22`, `rdf:type`, `skos:Concept`) and so

$$\mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{skos:Concept}) = 0 \tag{6.26}$$

Let C and D be class expressions. The idea presented for the intersection of data ranges applies to the intersection of class expressions $C \text{ AND } D$, and so the corresponding function definition is very similar to Equation 6.13:

$$\mu_{\mathbb{G}}(a, C \text{ AND } D) = \mu(a, C) \cdot \mu(a, D) \tag{6.27}$$

With respect to the sample RDF graph from Table 6.1,

$$\begin{aligned} \mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dbo:Locomotive AND dbo:MeanOfTransporation}) = \\ \mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dbo:Locomotive}) \cdot \\ \mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dbo:MeanOfTransporation}) = 1 \cdot 1 = 1 \end{aligned} \tag{6.28}$$

while

$$\begin{aligned} \mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dbo:Locomotive AND skos:Concept}) = \\ \mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dbo:Locomotive}) \cdot \mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{skos:Concept}) = \\ 1 \cdot 0 = 0 \end{aligned} \tag{6.29}$$

Now consider a pair of individuals a and b. The individual a should match the pattern consisting of the enumeration of individuals $\{b\}$ if, and only if, a and b are the same individual or if it follows from the graph that they are the same individual. The latter can be expressed only using `owl:sameAs`.

$$\mu_{\mathbb{G}}(a, \{b\}) = \begin{cases} 1 & a = b \\ 1 & (a, \texttt{owl:sameAs}, b) \in \mathbb{G} \vee (b, \texttt{owl:sameAs}, a) \in \mathbb{G} \\ 0 & \text{otherwise} \end{cases} \tag{6.30}$$

With respect to the sample RDF graph from Table 6.1,

$$\mu(\texttt{dbr:PKP\_class\_ET22}, \{\texttt{dbr:PKP\_class\_ET22}\}) = 1 \tag{6.31}$$

as both arguments refer to the same IRI, while

$$\mu(\texttt{dbr:PKP\_class\_ET22}, \{\texttt{dbr:PKP\_class\_EU07}\}) = 0 \tag{6.32}$$

as the IRIs in the arguments are different and the graph contains neither the triple (`dbr:PKP_class_ET22`, `owl:sameAs`, `dbr:PKP_class_EU07`) nor (`dbr:PKP_class_EU07`, `owl:sameAs`, `dbr:PKP_class_ET22`).

We now consider more complex class expressions, i.e. restrictions, that contain also a property and we begin with a value restriction. Let p be an object property and a, b individuals. The individual a should match the pattern $p \text{ VALUE } b$ if, and only, if there is a triple in the graph stating that a is in the relation p with b, i.e. the triple (a, p, b).

$$\mu_{\mathbb{G}}(a, p \text{ VALUE } b) = \begin{cases} 1 & (a, p, b) \in \mathbb{G} \\ 0 & \text{otherwise} \end{cases} \tag{6.33}$$

With respect to the sample RDF graph from Table 6.1

$$\mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dbo:builder} \text{ VALUE } \texttt{dbr:Pafawag}) = 1 \tag{6.34}$$

because there is the triple (`dbr:PKP_class_ET22`, `dbo:builder`, `dbr:Pafawag`) in the graph, while the triple (`dbr:PKP_class_ET22`, `dbo:builder`, `dbr:Fablok`) is not there and thus

$$\mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dbo:builder} \text{ VALUE } \texttt{dbr:Fablok}) = 0 \tag{6.35}$$

Now let r be a data property, a an individual and l a literal. Following the same line of reasoning, we define the matching function for the pattern r VALUE l:

$$\mu_{\mathbb{G}}(a, r \text{ VALUE } l) = \begin{cases} 1 & (a, r, l) \in \mathbb{G} \\ 0 & \text{otherwise} \end{cases} \tag{6.36}$$

Triple (`dbr:PKP_class_ET22`, `dbo:builder`, `"1183"^^xsd:integer`) occurs in the sample RDF graph from Table 6.1 and so

$$\mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dbo:numberBuilt} \text{ VALUE } \texttt{"1183"\^\^xsd:integer}) = 1 \tag{6.37}$$

while there is no triple (`dbr:PKP_class_ET22`, `dbo:numberBuilt`, `"483"^^xsd:integer`) and so

$$\mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dbo:numberBuilt} \text{ VALUE } \texttt{"483"\^\^xsd:integer}) = 0 \tag{6.38}$$

Consider a self-restriction p SELF, where p is an object property. We want an individual a match the pattern if, and only if, a is in the relation p with itself, i.e. the graph contains the triple (a, p, a):

$$\mu_{\mathbb{G}}(a, p \text{ SELF}) = \begin{cases} 1 & (a, p, a) \in \mathbb{G} \\ 0 & \text{otherwise} \end{cases} \tag{6.39}$$

There is no example of such a pattern in the graph presented in Table 6.1, as no triple there has the same subject and object.

Finally, we must consider the last and most complex pattern: the existential quantification. Let p be an object property and C a pattern. An individual a should match the pattern p SOME C if, and only if, it follows from the graph that a is connected with p to some individual b that matches the pattern C. That is, we look for any b such that there is a triple (a, p, b) and $\mu_{\mathbb{G}}(b, C) = 1$:

$$\mu_{\mathbb{G}}(a, p \text{ SOME } C) = \begin{cases} 1 & \exists b \ [(a, p, b) \in \mathbb{G} \wedge \mu_{\mathbb{G}}(b, C) = 1] \\ 0 & \text{otherwise} \end{cases} \tag{6.40}$$

Triple (`dbr:PKP_class_ET22`, `dct:subject`, `dbc:Co-Co_locomotives`) occurs in the sample RDF graph from Table 6.1 and so

$$\begin{aligned} \mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dct:subject} \text{ SOME } \texttt{skos:Concept}) = \\ \mu(\texttt{dbc:Co-Co\_locomotives}, \texttt{skos:Concept}) = 1 \end{aligned} \tag{6.41}$$

Conversely, there is no individual b in the graph such that there is the triple (`dbr:PKP_class_ET22`, `dct:subject` in the graph and $\mu(b, \texttt{dbo:Locomotive}) = 1$, so

$$\mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dct:subject} \text{ SOME } \texttt{dbo:Locomotive}) = 0 \tag{6.42}$$

The same line of reasoning can be applied to the pattern r SOME D, where r is a data property and D is a data range:

$$\mu_{\mathbb{G}}(a, r \text{ SOME } D) = \begin{cases} 1 & \exists l \ [(a, r, l) \in \mathbb{G} \wedge \mu_{\mathbb{G}}(l, D) = 1] \\ 0 & \text{otherwise} \end{cases} \tag{6.43}$$

Triple (`dbr:PKP_class_ET22`, `dbo:numberBuilt`, `"1183"^^xsd:integer`) occurs in the sample RDF graph from Table 6.1 and so

$$\begin{aligned}\mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dbo:numberBuilt}\ \text{SOME}\ \texttt{xsd:integer}) = \\ \mu(\texttt{"1183"\^\^xsd:integer}, \texttt{xsd:integer}) = 1\end{aligned} \tag{6.44}$$

Conversely, there is no literal l such that the triple (`dbr:PKP_class_ET22`, `dbo:numberBuild`, l) is in the graph and $\mu(l, \texttt{xsd:string}) = 1$, so

$$\mu(\texttt{dbr:PKP\_class\_ET22}, \texttt{dbo:numberBuilt}\ \text{SOME}\ \texttt{xsd:string}) = 0 \tag{6.45}$$

This concludes the analysis of all possible data ranges and class expressions in OWL 2 EL. The function itself does not include any sort of reasoning, but it is enough to replace checking the existence of triples in the graph with checking it with the reasoner to enable as much or as little reasoning capabilities as the user requires.

6.4.2 Pattern frequency

In order to define a frequent pattern, we must first define a measure of frequency.

Definition 6.4.2 (proof set)**.** Let $\mathcal{I}$ be a set of IRIs and $\mathcal{L}$ be a set of literals. A *proof set* S of a pattern C is the subset of the set $\mathcal{I} \cup \mathcal{L}$ such that its members match pattern C:

$$S = \{s \in \mathcal{I} \cup \mathcal{L} \colon \mu_{\mathbb{G}}(s, C) = 1\} \tag{6.46}$$

In other words, $\mu_{\mathbb{G}}(\cdot, C)$ is the indicator function of the proof set S for the pattern C w.r.t the RDF graph $\mathbb{G}$.

Let w be a function assigning to members of set $\mathcal{I} \cup \mathcal{L}$ a weight from the range $[0, 1]$, i.e. $w \colon \mathcal{I} \cup \mathcal{L} \to [0, 1]$. We call such a function a *weighting function*. The purpose of the weighting function is to convey the knowledge about the importance of a given individual or a literal to the algorithm. The default weighting function is $w^{def}(s) = \frac{1}{|\mathcal{I} \cup \mathcal{L}|}$ for all $s \in \mathcal{I} \cup \mathcal{L}$, but these values change as the algorithm goes on. More details on the weighting function is given later, in Subsection 6.5.4.

Definition 6.4.3 (support)**.** Let C be a pattern and S its *proof set*. The *support* of the pattern C w.r.t. the weighting function w is given with the following equation:

$$\sigma(C, w) = \sum_{s \in S} w(s) = \sum_{s \in \mathcal{I} \cup \mathcal{L}} w(s) \mu_{\mathbb{G}}(s, C) \tag{6.47}$$

Observe, that for the default weighting function this definition becomes equivalent to the relative support measure known from frequent itemset mining [12]:

$$\sigma(C, w^{def}) = \sum_{s \in S} \frac{1}{|\mathcal{I} \cup \mathcal{L}|} = \frac{|S|}{|\mathcal{I} \cup \mathcal{L}|} \tag{6.48}$$

Definition 6.4.4 (frequent pattern)**.** Given a *minimal support threshold* θ_σ, a frequent pattern w.r.t. the weighting function w is any pattern C such that $\sigma(C, w) \geq \theta_\sigma$.

Recall the sample set of IRIs $\mathcal{I}^{ex}$ and assume the *minimal support threshold* $\theta_\sigma = 0.5$. The *proof set* for the pattern $C = \texttt{dbo:builder}\ \text{VALUE}\ \texttt{dbr:H.\_Cegielski\_-\_Poznań\_S.A.}$ consists of the following three items: `dbr:PKP_class_EU07`, `dbr:PKP_class_OKl27`, `dbr:PKP_class_SU46`. If we assume that w_1 is the default weighting function for this set, i.e.

$$w_1(s) = \frac{1}{5} \qquad \forall s \in \mathcal{I}^{ex} \tag{6.49}$$

then the *support* of the pattern w.r.t. the function w_1 is

$$\sigma(C, w_1) = w_1(\texttt{dbr:PKP\_class\_EU07}) + w_1(\texttt{dbr:PKP\_class\_OKl27}) + w_1(\texttt{dbr:PKP\_class\_SU46}) = \frac{3}{5} \tag{6.50}$$

In such a case, C is a frequent pattern. Consider now the pattern $D = \texttt{dbo:builder}$ VALUE $\texttt{dbr:Pafawag}$. Its proof set consists of $\texttt{dbr:PKP\_class\_ET22}$ and $\texttt{dbr:PKP\_class\_EU07}$, and so the support is

$$\sigma(D, w_1) = \frac{2}{5} < \theta_\sigma \tag{6.51}$$

that is D is not a frequent pattern. If we assume a different weighting function w_2, such that

$$w_2(\texttt{dbr:PKP\_class\_ET22}) = \frac{2}{6} \tag{6.52}$$

$$w_2(\texttt{dbr:PKP\_class\_EU07}) = w_2(\texttt{dbr:PKP\_class\_SU46}) = w_2(\texttt{dbr:PKP\_class\_SM42}) = w_2(\texttt{dbr:PKP\_class\_OKl27}) = \frac{1}{6} \tag{6.53}$$

then D is also a frequent pattern, as its support equals to

$$\sigma(D, w_2) = w_2(\texttt{dbr:PKP\_class\_ET22}) + w_2(\texttt{dbr:PKP\_class\_EU07}) = \frac{3}{6} \geq \theta_\sigma \tag{6.54}$$

Definition 6.4.5 (frequent predicate)**.** A *frequent predicate* is a predicate p such that it can be a part of a frequent pattern. For this, sum of the weights of subjects present in a triple with the predicate p must reach at least the minimal support threshold θ_σ. The proof set for the predicate p is defined as

$$S = \{s \in \mathcal{I} \cup \mathcal{L} \colon \exists o\, (s, p, o) \in \mathbb{G}\} \tag{6.55}$$

We define the support consistently with the support for patterns:

$$\sigma(p, w) = \sum_{s \in S} w(s) \tag{6.56}$$

and we require that $\sigma(p, w) \geq \theta_\sigma$.

With respect to the graph presented in Table 6.1, the set of IRIs $\mathcal{I}^{ex}$ and the weighting function w_1,

$$\sigma(\texttt{dbo:builder}, w_1) = 1 \tag{6.57}$$

and thus $\texttt{dbo:builder}$ is a *frequent predicate*, while $\texttt{dbo:engineType}$ is not a frequent predicate as

$$\sigma(\texttt{dbo:engineType}, w_1) = \frac{1}{5} \tag{6.58}$$

6.4.3 Pattern length

The *length of a pattern* C $\mathfrak{l}(C)$ is the maximal number of patterns nested using the existential quantification plus 1. For example, $\mathfrak{l}(\texttt{dbo:Locomotive}) = 1$, as there is no nesting, while $\mathfrak{l}(\texttt{dbo:Locomotive}$ AND $\texttt{dct:subject}$ SOME $\texttt{dbc:Co-Co\_locomotives}) = 2$, because there is a single level of nesting. Let A be a class, C_1 and C_2 class expressions, D a data range, p an object property, r a data property, a an individual and l a literal. The complete recursive definition of the function $\mathfrak{l}$ is given below:

Definition 6.4.6 (Lenght of a pattern)**.**

$$\begin{aligned}
\mathfrak{l}(D) &= 1 \\
\mathfrak{l}(A) &= 1 \\
\mathfrak{l}(C_1 \text{ AND } C_2) &= \max\{\mathfrak{l}(C_1), \mathfrak{l}(C_2)\} \\
\mathfrak{l}(\{a\}) &= 1 \\
\mathfrak{l}(p \text{ VALUE } a) &= 1 \\
\mathfrak{l}(r \text{ VALUE } l) &= 1 \\
\mathfrak{l}(p \text{ SELF}) &= 1 \\
\mathfrak{l}(p \text{ SOME } C_1) &= 1 + \mathfrak{l}(C_1) \\
\mathfrak{l}(r \text{ SOME } D) &= 1
\end{aligned}$$

6.5 The algorithm

6.5.1 Mining frequent datatypes

We start the construction of the algorithm for mining the frequent patterns by considering the patterns of the following forms: a datatype D and limitations of the value space of D. To discover such patterns, we propose a function presented in Algorithm 6.2. The function uses four temporary data maps, initialized in lines 2–5. The maps use as the keys the datatype names, and as we established in Section 2.5, there are at most 19 datatypes to consider in OWL 2 EL, so the key space is very limited. The map σ is used to store the support for the datatype, the map S to store the proof sets, the map m to store the minimal value over all given literals of the datatype, while the map M to store the maximal value.

After the maps are prepared, every literal l of the input $\mathcal{L}$ is analyzed in lines 6–31. First, the set D of the possible datatypes for the literal is computed. If the literal has an explicitly defined datatype, the set D in line 8 is defined to contain only this datatype. If the type is not asserted, but the literal has a language tag, according to Hawke et al. [34] the datatype is `rdf:PlainLiteral` and only this we put in the set D in line 11. Otherwise, we iterate over all 19 datatypes in lines 15–19 and check if the textual representation of the literal is in the lexical space of the considered datatype.

Then, for every datatype in set D, the support is incremented by the weight of the literal in line 22, and the literal is added to the proof set of the datatype in line 23. The next line checks if the datatype is one of the seven datatypes that can be used in the patterns of form $D[>= m]$ and, if so, for each of them the minimal value in the datatype is computed and stored in the map m in line 25. Moreover, if the datatype can be used in the patterns of form $D[<= M]$, in line 27 we compute the maximal value in the datatype. The maps m and M are initialized with, respectively, ∞ and $-\infty$, so the first encountered literal will replace these infinities with finite values and for the datatypes that can not be used in the patterns of form $D[<= M]$ or $D[>= m]$, the maps keep their default values.

After all the literals were analyzed, the algorithm proceeds with producing the patterns. In lines 32–45 it iterates over all the datatypes that gathered enough support to cross the minimal support threshold θ_σ. If the maximal value in the datatype is above negative infinity, i.e., is a proper literal value, we know that DT is one of the six datatypes allowed in the patterns of form $D[>= m]$ and $D[<= M]$, and thus the assertion in line 36 is allowed and we

can produce and return the pattern DT AND $DT[<= max]$ AND $DT[>= min]$ in line 37, along with the corresponding proof set. Otherwise, if the minimal value is below infinity, we know that DT must be `xsd:nonNegativeInteger`, as it is the only datatype, for which we compute the minimal value, but not the maximal value and thus the algorithm produces the pattern `xsd:nonNegativeInteger` AND `xsd:nonNegativeInteger`$[>= min]$ and its proof set in line 40. Finally, if neither of the minimal and maximal values is computed, the algorithm in line 43 produces the simplest pattern, containing only the datatype name and its proof set.

Recall the sample RDF graph from Table 6.1 and let $\mathcal{L}$ be the set of all literals occurring in the graph, i.e. $\mathcal{L} = \{$`"1183"^^xsd:integer, "483"^^xsd:integer, "122"^^xsd:integer, "1822"^^xsd:integer, "54"^^xsd:integer`$\}$. During execution of the function `MineDatatype(`$\mathcal{L}$, w^{def}`)`, all of the literals are explicitly typed and the maps contain only a single key `xsd:integer`: σ[`xsd:integer`] $= 1$, S[`xsd:integer`] $= \mathcal{L}$, m[`xsd:integer`] $= 54$ and M[`xsd:integer`] $= 1822$. It generates a single pattern, `xsd:integer` AND `xsd:integer`$[>= 54]$ AND `xsd:integer`$[<= 1822]$, with the proof set of the all five literals.

6.5.2 Mining frequent patterns with a fixed object

We will now concentrate on the patterns that can be mined from a set of triples having the same predicate and object. If it is asserted in an ontology that an individual s belongs to a class A, then in its RDF serialization the following triple appears: (s, `rdf:type`, A). If there are many such individuals, there will be many triples with the predicate `rdf:type` and the object A, but with varying subject. Informally, we can express it as a triple pattern ($\cdot$, `rdf:type`, A). Recall that the three-level index $\mathfrak{I}$, discussed in Section 6.3, has in the first level the predicates, in the second level the objects and the subjects are in the third level. For a fixed A, to find all the subjects matching the triple pattern, it is enough to retrieve the appropriate part of the third-level: $\mathfrak{I}$[`rdf:type`][A].

Algorithm 6.3 presents a function to mine frequent patterns consisting of a single class name A. In lines 2–11, it iterates over all the objects A in the second level of the index for the predicate `rdf:type`. For each of them, a temporary variable σ to compute the support is initialized to 0 and then, in lines 4–10, the algorithm iterates over all the subjects for the predicate `rdf:type` and the object A, that are gathered in the index. In line 5, the weight of the currently considered subject is added to the support. As soon as the support reaches the minimal support threshold θ_σ in line 5, the pattern A along with its proof set is returned in line 7 and processing of the object A finishes in line 8. The function either continues with the next value in the second level of the index in line 2 or, if such a value can not be found, it quits.

Recall the index from Figure 6.2 and the weighting function w_1, assigning each of the five locomotives an equal weight of $\frac{1}{5}$. During an execution of the function `MineType` with these parameters, A becomes `dbo:MeanOfTransporation` and `dbo:Locomotive`. In both cases the corresponding third level contains all five locomotives and thus $\sigma = 1$ exceeds the minimal support threshold θ_σ and the function yields the patterns `dbo:MeanOfTransporation` and `dbo:Locomotive`, both with the proof sets equal to $\mathcal{I}^{ex}$. The execution of the loop in lines 4–10 is, in both cases, finished after reaching the third locomotive in the index.

The described idea can be generalized to an arbitrary predicate p, to generate patterns of the form p VALUE a and r VALUE l, where a stands for an individual and l for a literal. Consider the function presented in Algorithm 6.4. It receives the predicate p as a parameter and then in lines 2–11 it iterates over every value v in the second level of the index for the predicate p. For each of the values v, a temporary variable σ for the support is initialized in line 3 and the function iterates in lines 4–10 over all values s in the third level of the index for the predicate p and the object v.

```
1  function MineDatatype(ℒ, w)
2      σ ← an empty map for supports, with 0 as the default value
3      S ← an empty map for proof sets, with ∅ as the default value
4      m ← an empty map for minimal values, with ∞ as the default value
5      M ← an empty map for maximal values, with −∞ as the default value
6      foreach l ∈ ℒ do
7          if l has an explicitly defined type DT then
8              D ← {DT}
9          end
10         else if l has a language tag then
11             D ← {rdf:PlainLiteral}
12         end
13         else
14             D ← ∅
15             foreach supported datatype DT do
16                 if l is in the lexical space of DT then
17                     D ← D ∪ {DT}
18                 end
19             end
20         end
21         foreach DT ∈ D do
22             σ[DT] ← σ[DT] + w[l]
23             S[DT] ← S[DT] ∪ {l}
24             if DT ∈ {xsd:nonNegativeInteger, owl:real, owl:rational, xsd:decimal, xsd:integer,
                xsd:dateTime, xsd:dateTimeStamp} then
25                 m[DT] ← min{m[DT], l^LT}
26                 if DT ≠ xsd:nonNegativeInteger then
27                     M[DT] ← max{M[DT], l^LT}
28                 end
29             end
30         end
31     end
32     foreach DT : σ[DT] ≥ θ_σ do
33         min ← m[DT]
34         max ← M[DT]
35         if max > −∞ then
36             assert min < ∞
37             yield DT AND DT[<= max] AND DT[>= min], S[DT]
38         end
39         else if min < ∞ then
40             yield DT AND DT[>= min], S[DT]
41         end
42         else
43             yield DT, S[DT]
44         end
45     end
46 end
```

Algorithm 6.2: A function to mine frequent patterns of forms D, $D[<= M]$ and $D[>= m]$, where D is a datatype. Its parameters are a set of literals $\mathcal{L}$ and a weighting function w.

```
1  function MineType(ℑ, w)
2      foreach A ∈ ℑ[rdf:type] do
3          σ ← 0
4          foreach s ∈ ℑ[rdf:type][A] do
5              σ ← σ + w[s]
6              if σ ≥ θ_σ then
7                  yield A, ℑ[rdf:type][A]
8                  break
9              end
10         end
11     end
12 end
```

Algorithm 6.3: A function to mine frequent patterns of form A, where A is a class. Its parameters are a three-level index $\mathfrak{I}$ and a weighting function w.

The weight of every value s is added to the support in line 5 and once the support reaches at least the minimal support threshold θ_σ in line 6, the pattern $p \,\text{VALUE}\, v$ is generated and returned along with its proof set in line 7. The computation for the current value of v is finished in line 8 and the function continues, if possible, with the next value of v from the second level of the index in line 2. While the function is capable of generating two types of patterns, one for object properties and the other one for data properties, we do not need to distinguish them, as in the Manchester syntax they are syntactically identical and the distinction is based on the used property.

```
1  function MineValue(ℑ, w, p)
2      foreach v ∈ ℑ[p] do
3          σ ← 0
4          foreach s ∈ ℑ[p][v] do
5              σ ← σ + w[s]
6              if σ ≥ θ_σ then
7                  yield p VALUE v, ℑ[p][v]
8                  break
9              end
10         end
11     end
12 end
```

Algorithm 6.4: A function to mine frequent patterns $p \,\text{VALUE}\, a$ for p being an object property and a an individual, and $r \,\text{VALUE}\, l$, where r is a data property and l a literal. Its parameters are a three-level index $\mathfrak{I}$, a weighting function w and a property p.

Recall the index from Figure 6.2 and the weighting function w_1, assigning each of the five locomotives an equal weight of $\frac{1}{5}$ and assume $\theta_\sigma = 0.6$. We first discuss an execution of the function `MineValue` for the parameter p equal to `dbo:builder`. The variable v in line 2 iterates over the second level of the index, i.e. over the following three values: `dbr:Pafawag`, `dbr:H._Cegielski_-_Poznań_S.A.` and `dbr:Fablok`. For the first IRI, there are only two IRIs in the third level of the index, so σ never reaches θ_σ and no pattern is generated. The same goes for the third IRI, with a single IRI in the third level. For `dbr:H._Cegielski_-_Poznań_S.A.`, the situation is different: there are three IRIs in the third level, so σ reaches 0.6 and the pattern `dbo:builder` VALUE `dbr:H._Cegielski_-_Poznań_S.A.` is generated with the proof set consisting of `dbr:PKP_class_EU07`, `dbr:PKP_class_OKl27` and `dbr:PKP_class_SU46`. Assume now that p is `dbo:numberBuilt`. The variable v in line 2 iterates over five different values, but neither has

enough IRIs in the third level to reach the minimal support threshold and thus no pattern is generated.

The very same idea can be applied to yet another form of patterns, i.e. $p\,\textsc{self}$. For a fixed predicate p, we now look for triples of the form (s, p, s), i.e. having the same value in the subject and object positions. For this, we use a function presented in Algorithm 6.5. The function uses two temporary variables, σ initialized to 0 in line 2 to compute the support and S initialized to $\emptyset$ in line 3 to store the proof set. The main part of the function is the iteration over all values s in the second level of the index for the predicate p in lines 4–9. The function then checks in line 5 if the same value s occurs in the third level of the index for the predicate p and the object s. If it does, the support is increased by the weight of s in line 6 and s is stored in the proof set in line 7. After the iteration concludes, the function checks in line 10 if the support reached the minimal support threshold θ_σ, and if yes, it returns in line 11 the pattern $p\,\textsc{self}$ along with the proof set S. Otherwise no pattern is returned. We can not optimize this function by moving the if clause from line 10 up to the inside of the loop, because then it might return an incomplete proof set.

```
1  function MineSelf(ℑ, w, p)
2      σ ← 0
3      S ← ∅
4      foreach s ∈ ℑ[p] do
5          if s ∈ ℑ[p][s] then
6              σ ← σ + w[s]
7              S ← S ∪ {s}
8          end
9      end
10     if σ ≥ θ_σ then
11         yield p SELF, S
12     end
13 end
```

Algorithm 6.5: A function to mine frequent patterns of form $p\,\textsc{self}$ for p being an object property. Its parameters are a three-level index $\mathfrak{I}$, a weighting function w and the object property p.

Finally, we will consider an enumeration of individuals $\{a\}$ and an enumeration of literals $\{l\}$. Presence of these patterns is determined only by the weighting function w and does not depend on the index itself, as the pattern $\{a\}$ is frequent if, and only if, $w[a] \geq \theta_\sigma$. The function presented in Algorithm 6.6 uses this observation: in lines 2–6 it iterates over all currently considered IRIs and literals and it checks in line 3 if any of them has high enough weight. For such, it returns in line 4 the pattern $\{s\}$, along with the corresponding proof set consisting only of the considered value.

```
1  function MineEnum(ℐ, ℒ, w)
2      foreach s ∈ ℐ ∪ ℒ do
3          if w[s] ≥ θ_σ then
4              yield {s}, {s}
5          end
6      end
7  end
```

Algorithm 6.6: A function to mine frequent patterns of form $\{a\}$ for a being an individual and $\{l\}$ for l being a literal. Its parameters are a set of IRIs $\mathcal{I}$, a set of literals $\mathcal{L}$ and a weighting function w.

For example, consider the set $\mathcal{I}^{ex}$ and assume that $\theta_\sigma = 0.6$ and the weighting function w_3

such that

$$w_3(\texttt{dbr:PKP\_class\_OKl27}) = \frac{8}{12} \tag{6.59}$$

$$w_3(a) = \frac{1}{12} \qquad \text{for the four other IRIs } a \tag{6.60}$$

An execution of the function `MineEnum` for the set $\mathcal{I}^{ex}$ and the function w_3 finds that

$$w_3[\texttt{dbr:PKP\_class\_OKl27}] \geq \theta_\sigma \tag{6.61}$$

and thus yields the pattern $\{\texttt{dbr:PKP\_class\_OKl27}\}$ with the proof set $\{\texttt{dbr:PKP\_class\_OKl27}\}$.

6.5.3 Mining intersections

After mining a set of patterns, it is useful to combine them into longer patterns using the AND operator. On one hand, this helps to present the mined patterns in a more condensed manner in the user interface and later in the ontology. On the other hand, it enables more expressiveness, as some patterns containing intersections can not be represented as a set of patterns, e.g. $p\,\text{SOME}(C\,\text{AND}\,D)$ requires a filler for p to belong at the same time to classes C and D, opposed to a set of two patterns $p\,\text{SOME}\,C$ and $p\,\text{SOME}\,D$, which require a filler to belong to C and a possibly different filler to belong to D.

To achieve these properties, SLDM combines a set of patterns mined over a given set of IRIs into closed intersections, that is intersections such that the proof set and the support of every operand are the same. The patterns used to build the intersections can then be safely removed from the set of patterns without any loss of information, as the intersection retains all the information about the support and the proof set. Algorithm 6.7 presents a function, which first constructs a mapping from a proof set to a set of patterns sharing the proof set, and then for every distinct proof set, a separate intersection is generated, comprising of all the patterns sharing this proof set. The obtained set of intersections is intended to replace the original set of patterns, as it contains the same information using not greater (and, hopefully, strictly less) number of patterns.

```
1 function MineClosedIntersections(𝒫)
2     G ← an empty map, with the default value ∅
3     foreach P, S ∈ 𝒫 do
4         G[S] ← G[S] ∪ {P}              // Add P to the set keyed by S in the map G
5     end
6     foreach key S in the map G do
7         yield G[S][0] AND G[S][1] AND G[S][2] ..., S
8     end
9 end
```

Algorithm 6.7: A function to mine closed intersections from the set $\mathcal{P}$ of pairs: a pattern P and its proof set S.

Recall the index from Figure 6.2 and the weighting function w_1, assigning each of the five locomotives an equal weight of $\frac{1}{5}$. In Subsection 6.5.2, we mined patterns `dbo:MeanOfTransporation` and `dbo:Locomotive`, both having all five locomotives in the proof sets. Using Algorithm 6.7, both patterns are combined into the intersection `dbo:MeanOfTransporation` AND `dbo:Locomotive` with the proof set consisting of all the five locomotives.

6.5.4 The pattern mining algorithm

So far, we discussed a set of functions, each designed to mine a specific class of patterns. We also showed how to retrieve and index additional knowledge form the remote RDF graph. We will now

combine them into *Swift Linked Data Miner* (SLDM), filling the missing parts in the process. An outline of the algorithm is presented as function `SLDM` in Algorithm 6.8.

```
1  function SLDM(I, L, w, d)
2      P ←MineDatatype(L, w)∪MineEnum(I, L, w)
3      T ← download triples having subjects in the set I, as described in Section 6.3
4      J ←BuildIndex(T)
5      J ←PruneIndex(J, w)
6      foreach p ∈ J do
7          if p == rdf:type then
8              P_p ← MineType(J, w)
9          end
10         else
11             P_p ← MineValue(J, w, p) ∪ MineSelf(J, w, p)
12         end
13         if P_p == ∅ ∧ d < θ_l then
14             P_p ← MineSome(J, w, p, d)
15         end
16         P ← P ∪ P_p
17     end
18     return MineClosedIntersections(P)
19 end
```

Algorithm 6.8: A function which combines retrieving, indexing and recursive mining into the Swift Linked Data Miner. Its parameters are a set of IRIs $\mathcal{I}$, a set of literals $\mathcal{L}$, a weighting function w and the current pattern length d.

The algorithm starts by mining patterns that do not depend on new information retrieved from the remote RDF graph, using the functions from Algorithm 6.2 and Algorithm 6.6. Then, the current set of IRIs $\mathcal{I}$ is used to retrieve new triples from the remote graph and an index is build. In order to save memory, the obtained index is pruned w.r.t. the weighting function w, using the function presented in Algorithm 6.9, that removes parts of the index that correspond to predicates that are not frequent. Precisely, it iterates over the first level of the index, builds an union S of all the third levels of the index. Then, for every such an union, it computes its support, i.e., sums up the weights of IRIs in the set S. If the obtained sum is less than the minimal support threshold θ_σ, the predicate p is not frequent and can not be a part of any frequent pattern, and thus the whole branch of the index corresponding to it can be removed.

Once the index is pruned, the algorithm iterates over all the predicates retained in it. If the predicate is `rdf:type`, the function `MineType` from Algorithm 6.3 is called, otherwise the functions `MineValue` from Algorithm 6.4 and `MineSelf` from Algorithm 6.5 are called. These functions construct a temporary set of patterns $\mathcal{P}_p$. If this set is empty, i.e. SLDM was unable to discover any patterns for the given parameters of the desired pattern length d, and the maximal length θ_l is not exceeded, the function `MineSome` from Algorithm 6.10 is used to discover patterns of the form $p\,\text{SOME}\ldots$.

First, in lines 2–15, the function constructs three temporary structures: sets $\mathcal{I}_{new}$ and $\mathcal{L}_{new}$ of all IRIs (resp. literals) occurring in the second level of the index for the predicate p and a counter *den*, which, for every subject occurring in the third level of the index for the predicate p, contains the number of different values in the second level. In other words, for a given IRI s and a predicate p, *den* contains a number of triples of the form $(s, p, \cdot)$ present in the index. Recall the index from Figure 6.2 and the weighting function w_1, assigning each of the five locomotives an equal weight of $\frac{1}{5}$. Consider p to be `dct:subject`, then $\mathcal{I}_{new}$ consists of the following four values: `dbc:3000_V_DC_locomotives`, `dbc:2-6-2T_locomotives`, `dbc:Bo-Bo_locomotives` and

```
1  function PruneIndex(ℑ, w)
2      foreach p ∈ ℑ do
3          S ← ∅
4          foreach o ∈ ℑ[p] do
5              S ← S ∪ ℑ[p][o]
6          end
7          σ ← 0
8          foreach s ∈ S do
9              σ ← σ + w[s]
10         end
11         if σ < θ_σ then
12             delete ℑ[p]
13         end
14     end
15 end
```

Algorithm 6.9: A function to prune the index $\mathfrak{I}$ w.r.t. the weighting function w. It removes whole parts of the index, which are not frequent according to the function w.

```
1  function MineSome(ℑ, w, p, d)
2      I_new ← ∅
3      L_new ← ∅
4      den ← an empty map, with the default value 0
5      foreach o ∈ ℑ[p] do
6          if o is an IRI then
7              I_new ← I_new ∪ {o}
8          end
9          else
10             L_new ← L_new ∪ {o}
11         end
12         foreach s ∈ ℑ[p][o] do
13             den[s] ← den[s] + 1
14         end
15     end
16     w_new ← an empty map, with the default value 0
17     foreach o ∈ ℑ[p] do
18         foreach s ∈ ℑ[p][o] do
19             w_new[o] ← w_new[o] + w[s]/den[s]
20         end
21     end
22     P ← SLDM(I_new, L_new, w_new, d + 1)
23     foreach C_new, S_new ∈ P do
24         S ← ∅
25         foreach s ∈ S_new do
26             S ← S ∪ ℑ[p][s]
27         end
28         C ← p SOME C_new
29         yield C, S
30     end
31 end
```

Algorithm 6.10: A function to recursively mine patterns of form $p\,\text{SOME}\ldots$. Its parameters are an index $\mathfrak{I}$, a weighting function w, a frequent predicate p and the pattern length d.

`dbc:Co-Co_locomotives`, while $\mathcal{L}_{new} = \emptyset$. The counter *den* equals to 2 for `dbr:PKP_class_ET22`, as it occurs twice in the third level of the index in the branch corresponding to `dct:subject` and 1 for the remaining four locomotives.

Then, a new weighting function w_{new} is computed in lines 16–21. Every object o present in the second level of the index for the predicate p is assigned a new weight, which is the sum over all of the subjects s in the third level, each incorporating $\frac{1}{den[s]}$ of its weight, i.e. an amount inversely proportional to the number of different objects o occurring with the predicate p and the subject s in the index. Continuing the previous example, the new weight assigned in w_{new} to `dbc:3000_V_DC_locomotives` is $\frac{3}{10}$, as the sum of the whole weight $\frac{1}{5}$ of `dbr:PKP_class_EU07` and half of the weight of `dbr:PKP_class_ET22`. `dbc:Co-Co_locomotives` receives the same value, having $\frac{1}{5}$ from `dbr:PKP_class_SU46` and the other half of the weight of `dbr:PKP_class_ET22`. The remaining two IRIs, `dbc:2-6-2T_locomotives`, `dbc:Bo-Bo_locomotives`, receive weight $\frac{1}{5}$ each, that is, respectively, the weight of `dbr:PKP_class_OKl27` or `dbr:PKP_class_SM42`.

Using the obtained three temporary structures, SLDM is recursively called in line 22 with an increased pattern length $d+1$, the obtained patterns, along with their proof sets, are stored in the set $\mathcal{P}$. Once the recursive call finishes, the mined patterns in $\mathcal{P}$ must be adjusted in lines 22–30, by prefixing them with p SOME and constructing their corresponding proof sets from the proof sets from the recursive call. Precisely, for every pattern, the new proof set is constructed by computing the union of all of the third levels of the index $\mathfrak{I}$ corresponding to the predicate p and every value s present in the proof set returned by the recursive call. Continuing the previous example recall the RDF graph presented in Table 6.1. The recursive call of SLDM downloads the four triples depicted in the bottom of the table and yields the pattern `skos:Concept` with the proof set consisting of `dbc:3000_V_DC_locomotives`, `dbc:2-6-2T_locomotives`, `dbc:Bo-Bo_locomotives` and `dbc:Co-Co_locomotives`. After the recursive call returns, the pattern is prefixed yielding `dct:subject` SOME `skos:Concept`, and the proof set is transformed to the set of all five locomotives.

6.6 Plugin to *Protégé*

Swift Linked Data Miner has been implemented in *Java* using the *OWL API*[1] [37] to represent the patterns and *Apache Jena*[2] [58] to interact with a remote SPARQL endpoint. It is available as a JAR library, with the building process controlled by Apache Maven[3]. It is open-source and available in a *Git*[4] repository at `http://bitbucket.com/jpotoniec/sldm`.

Such a choice of technologies enabled us to integrate SLDM as a plugin to *Protégé*, a very popular tool for ontology development [64, 67, 48]. The plugin adds a new tab to the *Protégé* interface, that contains the class hierarchy, a toolbar to control the execution of the algorithm and two tabs:

Basic depicted in Figure 6.3, with the address of the SPARQL endpoint and the mined axioms;

Expert with detailed settings of the algorithm, depicted in Figure 6.4.

The simplistic design of the *Basic* tab enables even an inexperienced user to mine axioms with SLDM using the default settings in literally few mouse clicks, while the detailed view of the *Expert* tab allows for fine-tuning the parameters if the situation dictates.

[1] `http://owlapi.sourceforge.net/`
[2] `https://jena.apache.org/`
[3] `https://maven.apache.org/`
[4] `https://git-scm.com/`

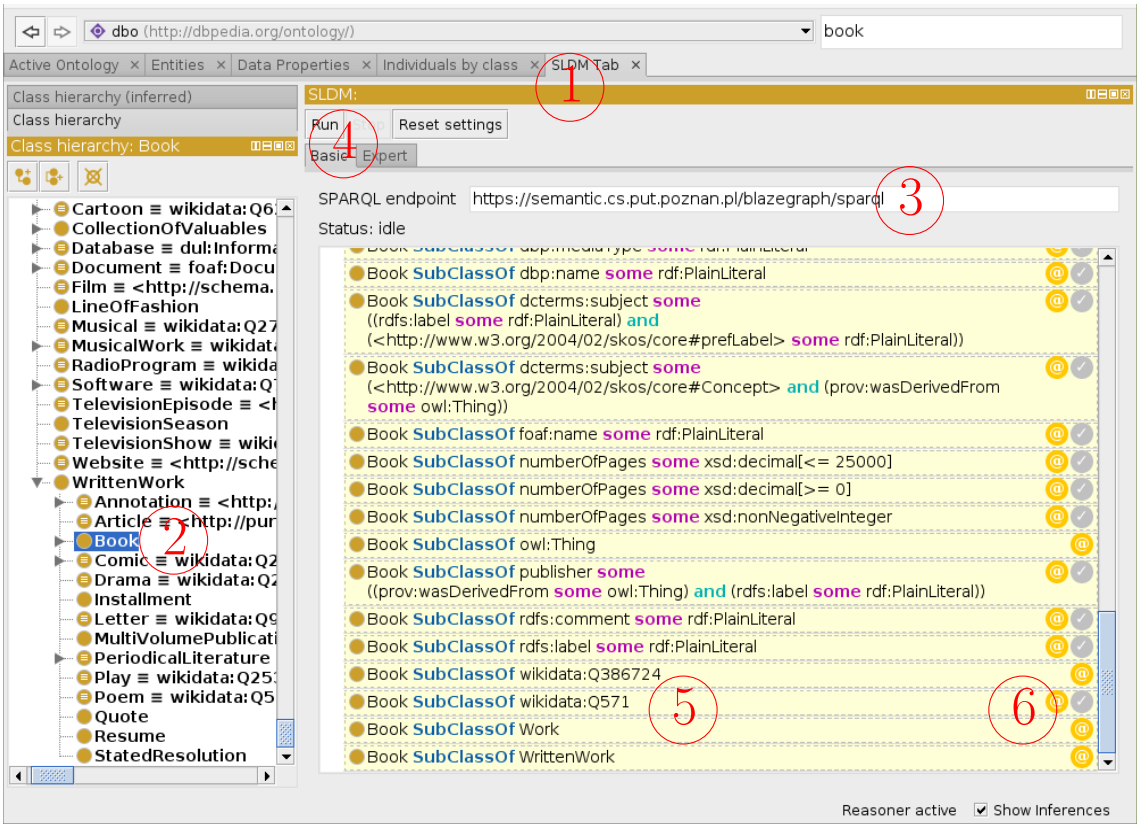


Figure 6.3: The basic interface of the SLDM plugin for *Protégé*. The user starts by selecting the SLDM Tab (1) and choosing the class of interest in the Class hierarchy (2). Then the user enters the address of the SPARQL endpoint (3) and clicks Run (4). The mined axioms are displayed right after they are mined in the main part of the window (5). Each axiom has one or two buttons attached (6): detailed information about support of the axiom can be retrieved by clicking the button labeled @; if the axiom does not logically follow from the ontology, it can be asserted to it by clicking the ✓ button (otherwise the button is not displayed).

SLDM has been also integrated with *WebProtégé*, a simpler version of *Protégé*, operating solely in a web browser [91, 39]. The plugin has been developed in the course of a bachelor thesis [86] and later presented at a conference [87]. The main goal was to move as much computation as possible to the client-side to avoid overloading the server hosting *WebProtégé* and enable access to a SPARQL endpoint, which is accessible by the client, but may be inaccessible to the server (e.g., being an internal endpoint of a company). The interface of the SLDM plugin for *WebProtégé* is presented in Figure 6.5.

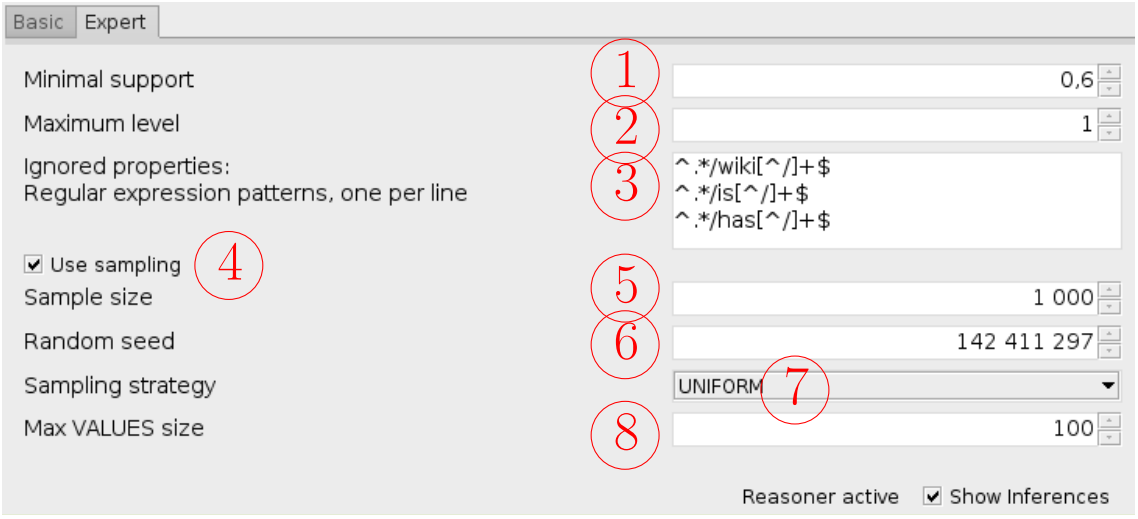


Figure 6.4: The expert tab of the SLDM plugin for *Protégé*, enabling the user to configure parameters of the algorithm. The user may set up the minimal support threshold (1) and the maximal length of mined patterns (2). It is possible to configure a set of regular expressions to ignore some predicates during mining (3), a useful option e.g. when the RDF graph contains information not directly related to the ontology at hand. If the user enables sampling (4), it is also possible to enter the sample size (5) and the seed used to initialize the random numbers generator (6). Moreover, it is possible to select the sampling strategy (7) and to enter the maximal number of IRIs used in a single SPARQL query in the VALUES clause (8).

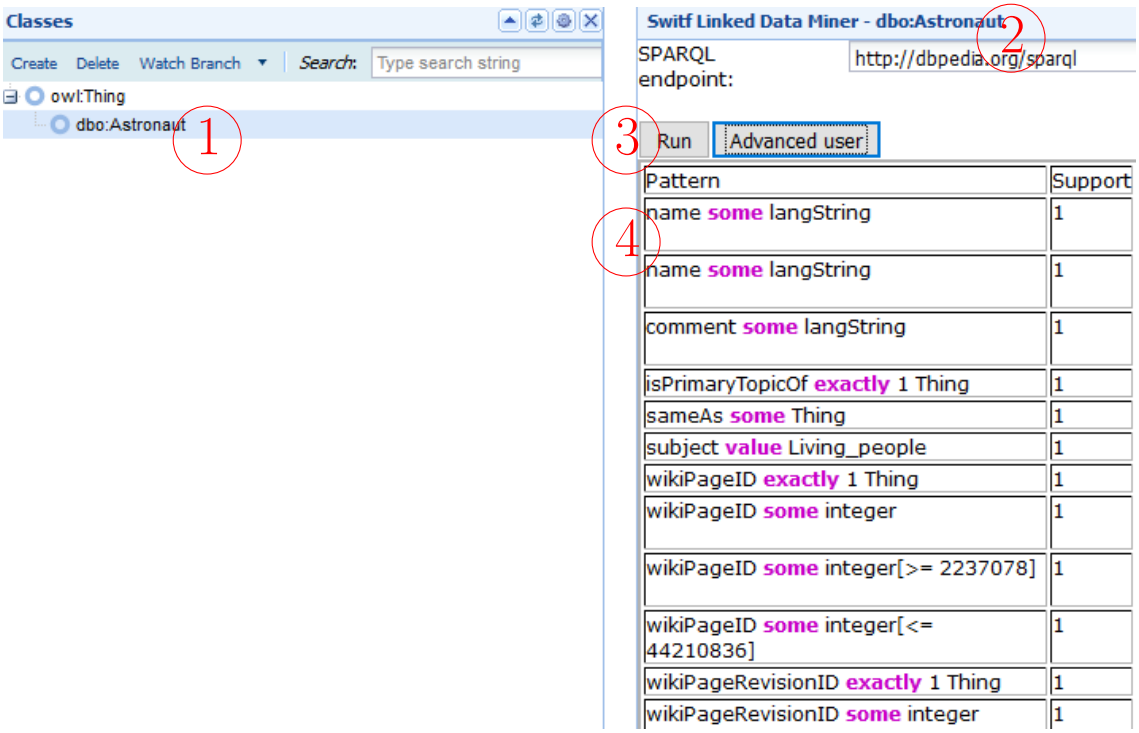


Figure 6.5: The interface of the SLDM plugin for *WebProtégé*. The user selects the class of interest (1), enters the address of the SPARQL endpoint (2) and clicks *Run* (3). The mined axioms, along with their support, are displayed in the right-hand side of the interface (4). The picture reprinted from [87].

6.7 Experimental evaluation

In order to evaluate the algorithm, we decided to perform a crowdsourcing experiment to verify whether the axioms mined by SLDM are valid and meaningful new knowledge, which could be added to an ontology. To prepare the experiment, we had to address two general issues first:

1. Which RDF graph is to be used for mining?
2. How to present the mined axioms to the crowd?

The answers to both questions must take into account the limitations of the crowd. In particular, one can not expect presence of domain experts on any particular subject in the crowd, i.e. the tasks for the crowd must require only general knowledge and can not require knowledge on any particular knowledge representation formalism. Taking these into account, we decided to use DBpedia as the RDF graph and translate each mined axiom to a set of sentences in English. The framework for the preparation and execution of the experiment has been presented as a master thesis [41] and below we present its core points.

We used DBpedia 2015-04 and started by performing an exploratory data analysis to gather basic statistics about the graph. For each of the classes present in the graph, we counted the number of instances asserted to the class, the average number of different triples having an instance of the class as a subject, the average number of different predicates in these triples and the depth of the class in the subsumption hierarchy in the DBpedia ontology. Analyzing the results, we decided on two sets of criterion, which must be all jointly fulfilled by a class to be selected for the experiment. The first set was to ensure good statistical support for the mined axioms and wide coverage of the different parts of the DBpedia:

- at least 100 instances;
- the average number of triples at least 50;
- the average number of predicates at least 20;
- pairwise different parent in the subsumption hierarchy;
- at least three different grandparents in the subsumption hierarchy;
- depth 3 in the subsumption hierarchy.

The other set was to limit the number of mined axioms, to keep the costs of the crowdsourcing verification under control. It consisted of only one criterion: the average number of predicates at most 35. Using these criterion, we selected 5 classes along with their parents and grandparents. The obtained taxonomy of 14 classes is presented in Figure 6.6.

On each of these classes, we run SLDM twice: the first execution was with the minimal support threshold $\theta_\sigma = 0.5$ and the other one with the minimal support threshold $\theta_\sigma = 0.8$. In both cases we used sample size of 50,000 and the maximal length $\theta_l = 2$. This way we obatained 14 sets of axioms for each for the two SLDM executions. To decide which of them were merely rediscovered facts already present in the ontology and which of them represented new knowledge, we used HermitT reasoner[5] [63] to see, for each mined axiom, two things: (1) does it logically follow from the DBpedia ontology, (2) does it logically follow from the DBpedia ontology enriched with all the axioms mined for the superclass of the considered class. The summary of the obtained results is presented in Table 6.2.

[5] `http://www.hermit-reasoner.com/`

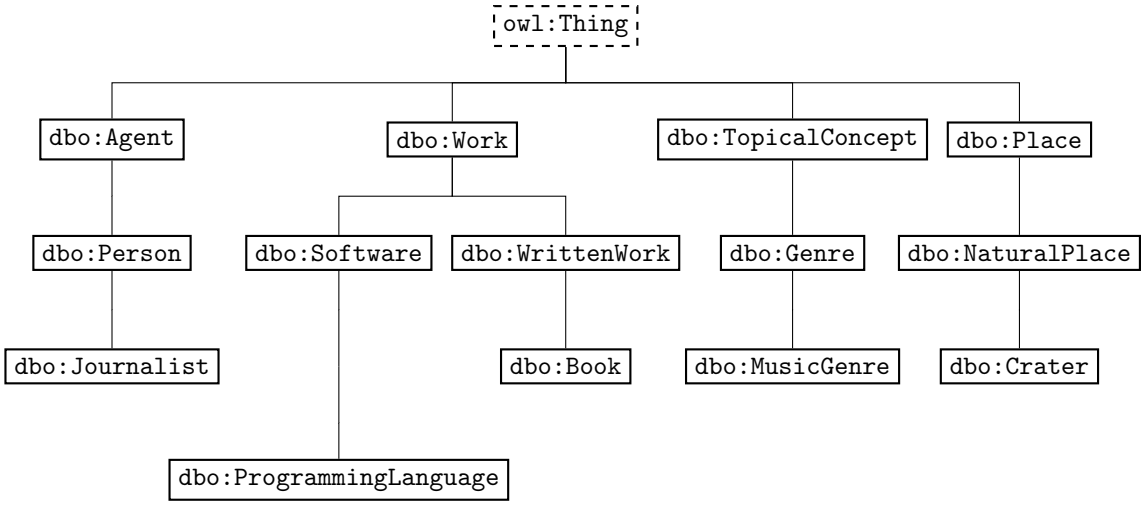


Figure 6.6: The taxonomy of classes selected from the DBpedia ontology to perform the crowdsourcing experiment. The class `owl:Thing` is depicted to show the position of the classes in the subsumption hierarchy, but was not used in the experiment.

Table 6.2: The numbers of axioms mined by SLDM for each of the two minimal support thresholds θ_σ and for each of the 14 classes presented in Figure 6.6. The column *overall* is the overall number of axioms mined for a given class with a given threshold, the column *in the ontology* is the number of the axioms that logically follow from the ontology and the column *in the superclass* is the number of the axioms that logically follow from the ontology with all the axioms mined for the superclass asserted. The last value can not be computed for the classes from the top level of the hierarchy.

class	number of mined axioms for $\theta_\sigma = 0.8$			number of mined axioms for $\theta_\sigma = 0.5$		
	overall	in the ontology	in the superclass	overall	in the ontology	in the superclass
Agent	9	3	-	22	3	-
Person	9	7	9	22	7	20
Journalist	109	8	10	131	8	21
Work	11	3	-	14	3	-
Software	17	4	12	45	4	14
ProgrammingLanguage	16	6	12	28	6	13
WrittenWork	14	4	12	63	4	15
Book	35	7	15	108	7	29
TopicalConcept	10	3	-	13	3	-
Genre	13	4	11	27	4	13
MusicGenre	13	5	13	27	5	27
Place	21	3	-	186	5	-
NaturalPlace	13	4	12	24	4	24
Crater	12	5	11	25	5	12

The obtained axioms were then translated to English using a tool based on Attempto Controlled English, a well-founded variant of English suitable to express formal knowledge without any loss of information [81]. The idea of the translation tool was based on [44], but the tool has been created from scratch as the preexisting tools were too limited for our use case. The translation used labels represented in the ontology using the `rdfs:label` predicate. If a label was not present in the ontology, the local part of the IRI (i.e., the part after the last `/`) was heuristically transformed and used, e.g. camel case `hasPhotoCollection` was replaced with `has photo collection`. During the translation, axioms referring to vocabulary originating from other Linked Data datasets, e.g., *Wikidata* [96, 97], were removed.

The main motivation for a worker in a crowdsourcing experiment is money, so the experiment must be carefully designed and conducted in order to avoid cheating workers. This is achieved by providing a gold standard, a set of tasks for the workers that have a single correct answer. The worker is periodically provided with such a task, but without revealing that there exists the correct answer. The performance of the worker is judged by the crowdsourcing platform based on the results on the gold standard. If the performance of a worker is too low, the answers given by the worker are discarded.

To provide such a gold standard, we used the axioms that logically follow from the ontology, as they are known to represent correct knowledge. An answer to any of the questions corresponding to these axioms must always be positive. Further, to prevent answering positively to every displayed question, we negated some of these axioms to provide questions that must be answered negatively. The remaining questions constituted a set of payable questions, i.e., the questions that the workers were paid to answer.

From the set of axioms mined with the threshold $\theta_\sigma = 0.8$ we obtained 56 questions in the gold standard and 168 payable questions, while for the other set we obtained, respectively, 61 and 425 questions. We posed the questions to *CrowdFlower*[6], offering a pay of 0.03 USD for answering a set of 10 questions. We requested answers from at least 20 different workers for every payable question. We also required that they respond correctly to at least 70% of the gold standard questions.

Before answering the first question, a worker was displayed an extensive instruction[7], explaining the purpose and giving examples for both positive and negative answers. For each of the questions, a worker could answer one of the three questions: *Yes* (i.e., the corresponding axiom represents correct knowledge), *No* (i.e., the corresponding axiom does not represent correct knowledge), *I don't know*. If the worker answered *No*, the worker was asked to provide a short, textual explanation.

The *CrowdFlower* job was finished within few hours and costed us around 35 USD. To each of the questions, it assigned three numbers: the number of workers that answered, respectively, *Yes*, *No* and *I don't know* to the questions. These three numbers always sum to 20, but apart from that they can be any integer value from 0 to 20. To present the results, we aggregated them into bins, containing a similar number of answers, i.e. 0–5, 6–10, 11–15, 16–20 answers. We then created groups by considering jointly bins for each of the answers and counted number of questions in each category. The aggregated results are presented in Table 6.3.

The obtained results show that most of the mined axioms were labeled as correct by most of the crowd, i.e. they would be a valuable suggestion for an ontology engineer.

[6] `https://www.crowdflower.com/`

[7] `https://bitbucket.org/jpotoniec/sldm/raw/aa87e060a8621a50fcf16eaaa04c543aaa42d6aa/CF_source/instructions.txt`

Table 6.3: The summary of results obtained from the crowdsourcing experiment. Each question was posed to 20 workers, and then assigned to a category depending on the number of each answers.

Yes	*No*	*I don't know*	$\theta_\sigma = 0.5$		$\theta_\sigma = 0.8$	
16–20	0–5	0–5	295	69.41%	65	38.69%
11–15	6–10	0–5	40	9.41%	41	24.40%
11–15	0–5	0–5	78	18.35%	45	26.79%
6–10	11–15	0–5	3	0.71%	3	1.79%
6–10	6–10	0–5	5	1.18%	13	7.74%
0–5	16–20	0–5	2	0.47%	0	0.00%
0–5	11–15	0–5	2	0.47%	1	0.60%
overall			425	100.00%	168	100.00%

Table 6.4: The summary of the stability analysis of the algorithm. The column *distinct axioms* represent the number of character-wise distinct axioms mined in any of the ten repeats, while the column *all axioms* is the overall number of mined axioms. The columns 1–10 present the number of distinct axioms entailed by the number of sets given in the header.

class	distinct axioms	all axioms	# of axioms entailed by given # of the sets									
			1	2	3	4	5	6	7	8	9	10
`dbo:Book`	18	160	0	0	0	0	0	0	0	2	0	16
`dbo:Crater`	13	120	0	0	0	1	0	0	0	0	0	12
`dbo:Journalist`	16	150	0	0	0	0	0	0	0	5	0	11
`dbo:MusicGenre`	16	124	0	0	1	1	0	1	0	0	0	13
`dbo:ProgrammingLanguage`	15	150	0	0	0	0	0	0	0	0	0	15

6.8 Computational characteristics of SLDM

6.8.1 Stability of the obtained results

Inevitably, using sampling can introduce mistakes in the obtained results. To see how well does SLDM behave depending on the choice of the initial set of IRIs, we designed the following experiment. We set the sample size to 250 and the minimal support threshold θ_σ to 0.95. Using `random.org` we selected ten different random seeds and run SLDM using each of these random seeds for each of the five classes from the bottom of the hierarchy of classes presented in Figure 6.6. In this way, for each of the classes, we obtained ten sets of axioms: $S_1, S_2, \ldots, S_{10}$. First, we computed a union of these sets and counted the overall number of mined axioms $|S_1| + \ldots + |S_{10}|$ and the number of distinct axioms $|S_1 \cup \ldots \cup S_{10}|$. Then, each of the sets was enriched with the axioms of DBpedia ontology $\mathcal{O}$: $S_1 \cup \mathcal{O}, \ldots, S_{10} \cup \mathcal{O}$. For each of the distinct mined axioms $\alpha \in S_1 \cup \ldots \cup S_{10}$, we computed how many of the enriched sets entail the axiom:

$$|\{S_i \colon S_i \cup \mathcal{O} \models \alpha\}|$$

In Table 6.4, we present a summary of the obtained numbers. In every case, at least 68% of the axioms were entailed by all the sets and at least 87.5% by at least half of the sets. For `dbo:ProgrammingLanguage` every time exactly the same set of axioms was mined. These observations confirm our hypothesis that, even though sampling may introduce mistakes, the obtained results generally do not depend on the choice of the sample.

6.8.2 Analysis of the sampling strategies

In order to see how the different sampling strategies influence the obtained results, we performed the following experiment. We used the same five classes from the bottom of the hi-

erarchy from Figure 6.6, the minimal support threshold θ_σ was selected from the set of six different values $\{0.5, 0.6, 0.7, 0.8, 0.9, 1\}$ and the sample size from the set of five different values $\{50, 100, 200, 500, 1000\}$. We also considered all three sampling strategies. For each of the combinations of the parameters we run SLDM and thus computed $5 \cdot 6 \cdot 5 \cdot 3 = 450$ sets of axioms. From these sets we removed the axioms that were logically entailed by the DBpedia ontology. We then counted the number of axioms in each set and report the results in Table 6.5.

In 99/150, that is 66% of the cases, the predicates counting strategy yielded the highest (including ties) number of mined axioms. This is an expected observation, since the predicates counting strategy prefers IRIs that are described with a large number of predicates, while SLDM computes support relatively to triples sharing the predicate.

In order to perform further comparison, within each of 150 various parameters settings except the strategy, i.e. the class, the minimal support threshold and the sample size, we posed the following question: is any of the three sets of axioms fully entailed by any other one. For each pair of strategies we counted the number of such situations and report the overall numbers in Table 6.6. Axioms mined with both counting strategies entail axioms mined with the uniform strategy in 81% of the cases, while the contrary is true in less than 50% of the cases. This hints that the counting strategies yield more general results than the uniform strategy, which is consistent with the fact that the counting strategies use more information during the sample selection. Furthermore, the results of the predicates counting strategy and the triples counting strategy are very similar. As the query template used in the predicates counting strategy is simpler, this is probably the preferred of the two counting strategies.

6.8.3 Runtime performance of SLDM

In order to evaluate runtime performance of SLDM, we selected the class `dbo:Book` and performed the following experiment. We created all 300 possible combinations of the following parameters values: 10 different random seeds; 6 minimal support thresholds θ_σ $\{0.5, 0.6, 0.7, 0.8, 0.9, 1\}$; 5 sample sizes $\{50, 100, 200, 500, 1000\}$. We fixed the sampling strategy to the uniform strategy. For each of the combinations of the parameters values, we executed SLDM 10 times, to account for natural variability of the computer system. Overall, during this experiment, we executed SLDM 3000 times over a timespan of 15 hours. The computation took place on a server operated by *Debian Linux* and equipped with two *Intel Xeon E5-2630 v3* CPU running 2.4GHz each and 256 GB of random-access memory (RAM).

During each execution, we measured overall CPU time consumed by SLDM, number of SPARQL queries posed to the SPARQL endpoint and the memory consumption. Each measured CPU time includes the time required to start *Java Virtual Machine* (JVM), and the memory consumption was also measured for the whole JVM. We then averaged CPU times and numbers of posed queries over the repeats and random seeds. The obtained results were then plotted as box plots and presented in Figure 6.7 and Figure 6.8. Maximal memory consumption for each of the six different minimal support thresholds θ_σ is presented in Table 6.7.

The CPU time of the execution never exceeded 120 seconds, even for the lowest minimal support threshold and highest sample size and the number of posed queries never exceeded 400. Both metrics are correlated due to the design of the algorithm and the plots clearly show this. The maximal amount of required RAM was less than 10GB. Neither CPU time nor the required amount of memory forbid SLDM to be used on a contemporary personal computer in an on-line manner.

Table 6.5: The number in a cell is the number of axioms mined by SLDM for the class given in the header, using the values given in the header: the sampling strategy, the sample size n and the minimal support threshold θ_σ. The axioms logically entailed by the DBpedia ontology were not counted. For sampling strategies, *uni.* stands for uniform, *pc* for predicates counting, *tc* for triples counting. Reprinted from [72].

		dbo:Book			dbo:Crater			dbo:Journalist			dbo:MusicGenre			dbo:ProgrammingLanguage		
θ_σ	n	uni.	pc	tc	uni.	pc	tc	uni.	pc	tc	uni.	pc	tc	uni.	pc	tc
0.5	50	54	88	85	19	23	20	50	19	66	18	40	63	49	34	61
	100	87	102	87	22	23	21	69	19	64	13	35	50	32	57	51
	200	81	109	99	22	23	23	69	96	76	20	40	59	25	45	41
	500	96	106	102	20	20	20	96	77	96	20	34	47	22	37	38
	1000	96	111	113	20	20	20	113	125	113	22	30	31	22	22	22
0.6	50	58	61	60	16	21	20	38	79	57	7	25	30	23	31	28
	100	46	67	54	16	17	17	62	63	56	7	24	38	15	27	24
	200	51	66	61	16	16	17	61	65	60	9	29	34	13	20	23
	500	55	68	68	16	16	16	83	68	64	11	27	34	13	19	19
	1000	60	69	75	16	16	16	116	110	110	14	18	18	13	13	13
0.7	50	26	37	29	10	17	16	36	55	49	7	21	28	11	11	19
	100	39	42	41	7	16	10	53	55	58	7	17	29	11	16	15
	200	31	49	54	10	16	10	53	61	56	7	17	29	11	13	17
	500	41	52	55	10	10	10	85	83	71	7	13	23	11	11	13
	1000	40	53	54	10	10	10	105	105	91	8	9	10	11	11	11
0.8	50	14	29	17	7	10	7	34	61	16	7	11	17	9	11	13
	100	19	28	25	7	7	10	16	16	16	7	8	19	9	11	11
	200	21	26	25	7	7	7	16	16	16	7	8	18	9	11	11
	500	26	27	31	7	7	7	41	26	17	7	7	16	11	11	11
	1000	24	36	27	7	7	7	85	78	32	8	8	8	10	10	10
0.9	50	10	14	9	7	7	7	15	15	15	7	7	15	9	11	11
	100	9	17	15	7	7	7	13	15	15	7	8	11	9	11	9
	200	11	12	10	7	7	7	13	15	15	7	7	13	9	11	11
	500	9	12	14	7	7	7	15	15	15	8	8	8	9	10	9
	1000	10	12	12	7	7	7	15	15	15	8	8	8	9	9	9
1.0	50	7	9	9	7	7	7	3	14	13	6	6	6	8	5	6
	100	4	5	6	7	5	6	3	11	3	6	5	5	7	7	5
	200	2	6	4	5	5	5	3	3	3	1	3	5	5	7	5
	500	2	2	2	1	1	1	3	3	3	1	5	5	5	5	5
	1000	2	2	2	1	1	1	3	3	3	1	3	3	0	0	0

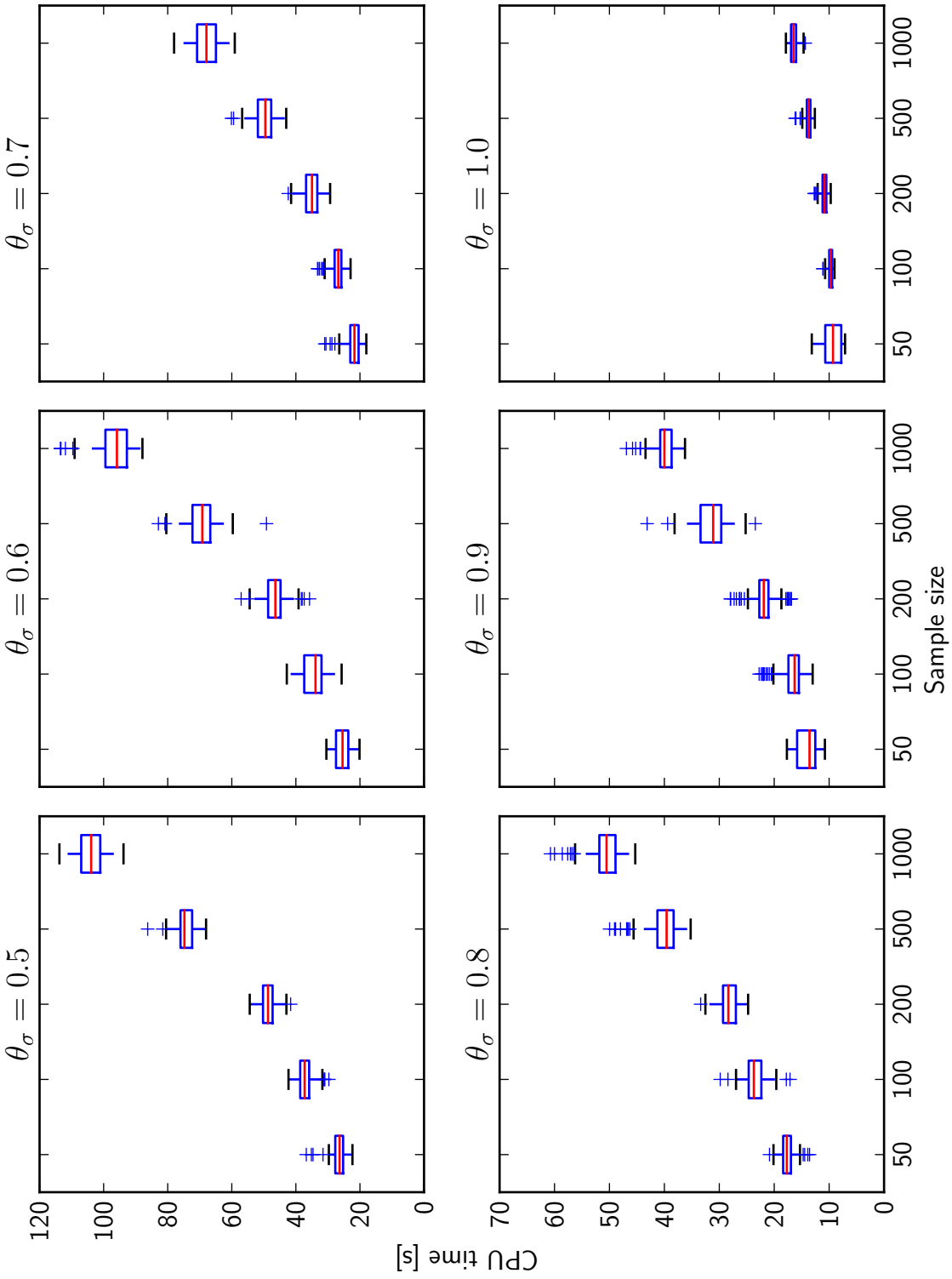


Figure 6.7: CPU time in seconds of SLDM execution during mining axioms for class `dbo:Book`. The times are averaged over 10 random seeds and 10 repeats and include starting and initalization of JVM. Each chart corresponds to a different value of the minimal support threshold θ_σ. The charts in the same row share the vertical axis and the charts in the same column share the horizontal axis. Reprinted from [72].

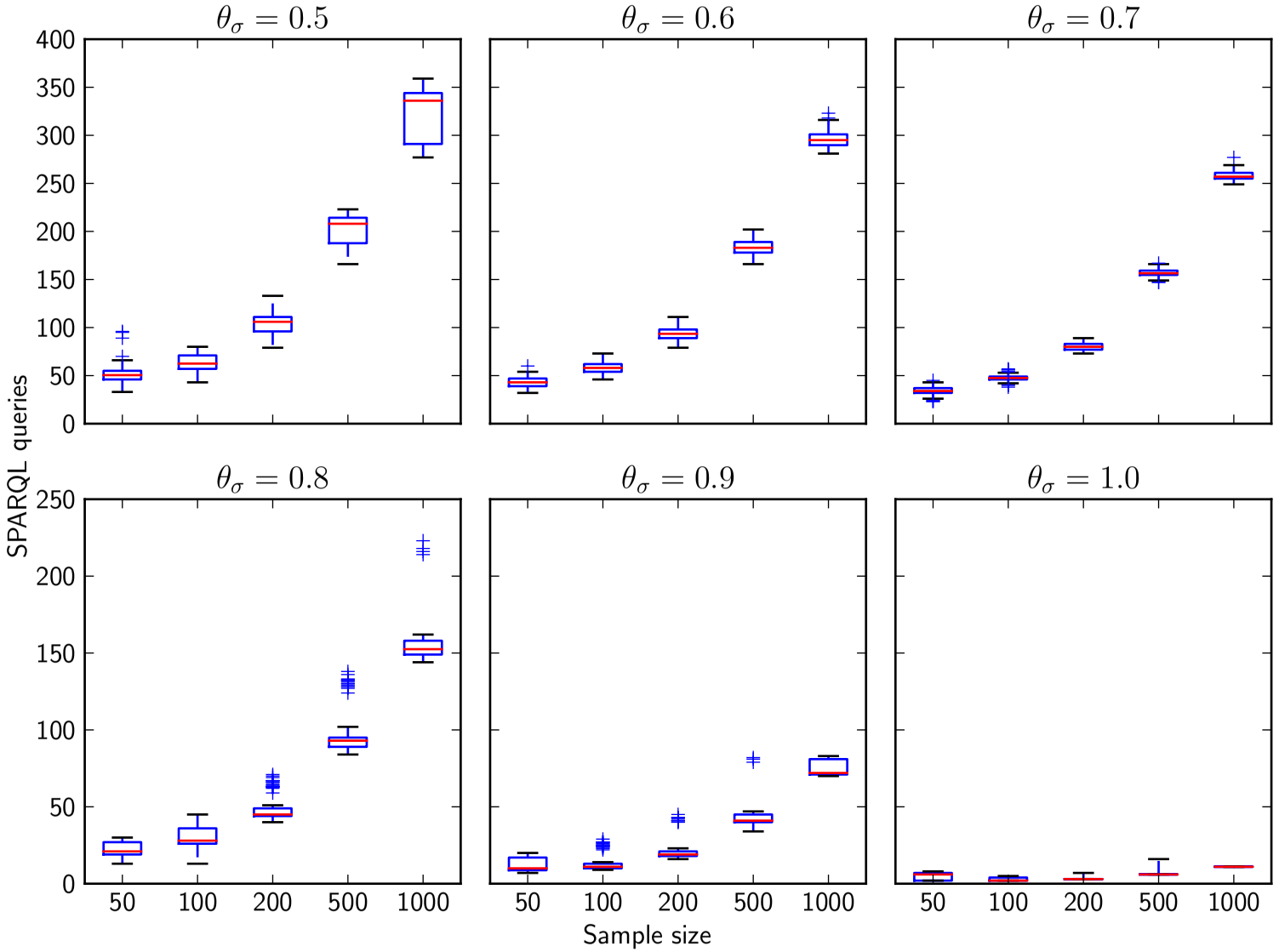


Figure 6.8: Number of SPARQL queries posed by SLDM to the SPARQL endpoint during mining axioms for class `dbo:Book`. The numbers are averaged over 10 random seeds and 10 repeats. Each chart corresponds to a different value of the minimal support threshold θ_σ. The charts in the same row share the vertical axis and the charts in the same column share the horizontal axis. Reprinted from [72].

Table 6.6: For each pair of the strategies, we counted the number of situations, where all the axioms mined by the strategy named in the row were entailed by the axioms mined by the strategy in the column, while keeping the rest of the parameters the same. For example, in 63 out of 150 different settings, all of the axioms mined with the predicates counting strategy were entailed by respective sets mined with the uniform strategy. Reprinted from [72].

entailed \ entailed by	uniform	predicates counting	triples counting
uniform	–	122/150=81%	121/150=81%
predicates counting	63/150=42%	–	100/150=67%
triples counting	69/150=46%	99/150=66%	–

Table 6.7: The maximal amount of RAM required by JVM during the execution of SLDM for class Book, aggregated over different executions, random seeds and sample sizes. Reprinted from [72].

The minimal support threshold θ_σ	0.5	0.6	0.7	0.8	0.9	1.0
Required amount of RAM [GB]	9.82	9.81	5.44	4.85	2.64	0.79

Chapter 7

Conclusions

The main contributions of the thesis consists in developing a set of methods for automatic enrichment of ontologies from Linked Data. We presented three methods for learning ontologies from Linked Data, each approaching the problem from a different perspective and thus solving different specific problems.

The first method uses mathematical modelling to make an optimal selection of a subset of patterns provided by a pattern generator. Usage of a mathematical model provides a formal definition of the problem as an optimization task. It uses a general solver to provide an anytime solution to the problem: the longer the solver is allowed to work, the better the final solution is obtained. If it is interrupted before the computation completes, it provides an heuristic solution to the problem. The method yields class expressions in OWL 2 EL, that can be named and used as new, named classes in an ontology. It is thus suitable to extend an existing class hierarchy. We provide an implementation of the proposed method as an operator within *RMonto*, a plugin to data mining environment *RapidMiner*. This method is based on our earlier work [74], its details are described in Chapter 4.

The second method allows completion of a class hierarchy of an ontology with missing subsumptions. The approach is based on Formal Concept Analysis (FCA), a well-founded framework for analysis of dependencies in a lattice of concepts. It integrates attribute exploration algorithm with a classifier, to help the user by answering some of the questions posed by the algorithm. The features used by the classifier are retrieved from Linked Data and the answers provided by the user are used as labels in the classification task. This approach is suitable to refine an existing ontology with additional class inclusion axioms within the class hierarchy. This method was implemented as a standalone Java application. We introduced this approach in [71, 76] and described the details in Chapter 5.

The third approach, called Swift Linked Data Miner (SLDM), is a frequent pattern mining algorithm with pattern space covering all class expressions allowed in OWL 2 EL as a superclass part of a class inclusion axiom. The algorithm does not follow a common generate-and-test strategy, where one first generates the candidate patterns and then tests whether they are frequent or not, rather SLDM is based on a clever organization of RDF triples in memory, which enables a very efficient frequent pattern discovery. SLDM is coupled with an approach to iteratively retrieve relevant parts of a Linked Data dataset during mining, taking into account the necessity to optimize the number and complexity of SPARQL queries posed to the SPARQL endpoint, as well as to save resources by using sampling. We showed that SLDM is a viable choice to add new axioms describing already existing classes in an ontology. We provided a Java implementation of SLDM, which can be used as a library or directly from popular ontology engineering tools *Protégé*

and *WebProtégé*. SLDM was presented earlier in [72, 75] and it is described in Chapter 6.

From the three presented methods, SLDM seems to be the most user-friendly one. The method based on mathematical modeling generates class expressions, that are equivalent to new classes, but can not generate meaningful descriptions to them. The user is thus forced to analyze the resulting class expression and decide on their natural language semantics. The attribute exploration algorithm requires lots of attention from the user even when coupled with a classifier. The algorithm forces the user to answer questions, some of which are very hard to understand and the process is tedious and error-prone. In comparison, SLDM is almost a maintenance-free algorithm: in its simplest form the user specifies a class of interest and a SPARQL endpoint and then just accepts or rejects the proposed axioms.

When comparing for the required computational time, SLDM is also taking the lead. In case of the first method, binary linear programming is an NP-complete problem and solving it may take long time, slowing down the whole process. The goal of equipping attribute exploration with a classifier was to shorten the time required to complete the process. While the goal was achieved, the required amount of time is still considerable, as the classifier must be trained first, and only then it can take care of some of the questions posed by the algorithm. SLDM is optimized for required time by using a three-level index, splitting queries posed to a SPARQL endpoint and sampling.

While these properties are important, ultimately, each of the methods serves a slightly different purpose and this should be the main criterion when selecting which one to use in the problem at hand: the first one introduces new classes, the second one completes the subsumption relation while the third one extends axiomatization of the ontology.

In comparison with the preexisting methods, the main advantage of the proposed methods is the ability to work using only an RDF graph available in an online SPARQL endpoint. There is no need to collect the whole graph beforehand and then process it offline. Moreover, none of the proposed methods uses any background knowledge about what is common or popular in ontology modelling, like the work of Bühmann and Lehmann [15]. This enables easier adaptation to the ever-changing landscape of the Linked Data. Finally, SLDM may be run in a mode that conserves the resources of the SPARQL endpoint, by limiting the complexity and number of posed queries. This is also a new feature, as the previously proposed methods tend to use complex SPARQL 1.1 queries with COUNT and GROUP BY clauses.

The proposed methods may find several applications. For example, an RDF graph may be build by a community, that introduces high amount of noise and disagreement, as it is in the case of *DBpedia*. It could be useful to extract and formalize the knowledge, and then to use it to guide the further growth of the graph. Another application is related to knowledge extraction from unstructured text: an algorithm produces an RDF graph without any ontology from the body of the text, the ontology is then automatically discovered and used to guide further extraction. In this way the graph is data-driven, yet equipped with taxonomical knowledge and consistent usage of vocabulary.

Each of the methods can be extended further and currently we propose the following research directions. The method based on mathematical modelling requires first performing a materialization of the patterns found in the data and only then the model can be generated and solved. An interesting problem is how to avoid the materialization and generate the model directly from the RDF graph. The generated class expressions lack descriptions and thus can be very hard to digest for the user. Providing a textual description, that would be related to the descriptions of the vocabulary used in a class expression on one hand, but not being merely an artificial concatenation using some controlled language, is certainly an interesting challenge. Finally, analyzing

and comparing different possible formulation of the optimization goal could be of interest. It may be the case that ontology engineering is not an "one size fits all" type of problem and different qualities must be taken into account in different contexts.

The main drawback of the second method is the time required to train a classifier. Replacing one classifier with another one, requiring a smaller number of examples, would only diminish the problem and not remove it completely. Ultimately, solving the problem would require transfer learning, i.e. some form of transferring knowledge obtained during one execution to the next one. This either requires classification features that are ontology-independent or an approach to translate the features between ontologies. Probably the best scenario would be to develop the features that are both ontology-independent and user-independent, and provide the users with a pretrained classifier, which is only fine-tuned during the execution, similarly as it is done now with artificial neural networks for image processing.

An obvious limitation of Swift Linked Data Miner is the pattern space covering only OWL 2 EL. While this is an OWL 2 profile of importance, extending it to cover the whole specification of OWL 2 would allow the user to make a choice about the constructors used in the mined axioms, and thus their expressivity and reasoning complexity.

Acknowledgements

The author acknowledges the support from the Polish National Science Center (Grant No 2013/11/N/ST6/03065), from the PARENT-BRIDGE program of Foundation for Polish Science, co-financed by the European Union, Regional Development Fund (Grant No POMOST/2013-7/8) and from grant 09/91/DSPB/0627.

Appendix A

OWL 2 EL

Below, we present OWL 2 EL using two syntaxes: from DL perspective in Manchester syntax [38] and from RDF perspective in Turtle syntax [18]. The latter immediately gives us bridge between RDF and OWL 2. We point out here that the Manchester syntax is incomplete, in the sense that some valid OWL 2 expressions can not be expressed in this syntax. However, due to its conciseness and clarity, we find it useful and practical. Moreover, we present, whenever possible, DL expressions corresponding to the presented OWL 2 expressions and, finally, their formal semantics according to [29]. We begin with the definition of data ranges, then we define class expressions and, finally, we define OWL 2 EL axioms. We use the following schema:

1. **name of the expression** a textual rendering with the arguments denoted using monotype font

   DL a corresponding DL expression, present only if such a correspondence exists

   semantics formal semantics based on the interpretation tuple I

   Manchester rendering in Manchester syntax

   Turtle rendering of corresponding RDF triples (or parts of them) in Turtle

A.1 Data ranges

An OWL 2 EL data range is one of the following:

1. **datatype** datatype `D` (This point is only to indicate that any valid datatype name is a valid data range.)

   semantics $\texttt{D}^{DT}$

   Manchester `D`

   Turtle `D`

2. **intersection of data ranges** an intersection of a data range `D` and a data range `E`

   semantics $\texttt{D}^{DT} \cap \texttt{E}^{DT}$

   Manchester `D and E`

   Turtle

```
_:x a rdfs:Datatype ;
    owl:intersectionOf (D E) .
```

3. **enumeration of literals** an enumeration containing a literal `l`

   semantics $\{l^{LT}\}$

   Manchester {`l`}

   Turtle

```
_:x a rdfs:Datatype ;
    owl:oneOf (l) .
```

A.2 Class expressions

An OWL 2 EL class expression is one of the following:

1. **class** class `C` (This point is only to indicate that any valid class name is a valid class expression.)

   DL `C`

   semantics $\mathtt{C}^C$

   Manchester `C`

   Turtle `C`

2. **intersection of class expressions** an intersection of a class expression `C` and a class expression `D`

   DL $\mathtt{C} \sqcap \mathtt{D}$

   semantics $\mathtt{C}^C \cap \mathtt{D}^C$

   Manchester `C and D`

   Turtle

```
_:x a owl:Class ;
    owl:intersectionOf (C D) .
```

3. **enumeration of individuals** an enumeration containing an individual `a`

   DL {`a`}

   semantics $(\{\mathtt{a}\})^C = \{\mathtt{a}^I\}$

   Manchester {`a`}

   Turtle

```
_:x a owl:Class .
    owl:oneOf (a) .
```

4. **existential quantification** an object is related with an object property (resp. a data property) `P` to an individual of a class expression (resp. a literal of a data range) `C`

   DL $\exists \mathtt{P}.\mathtt{C}$

   semantics (object property) $\{x\colon \exists y\colon \left[(x,y) \in \mathtt{P}^{OP} \wedge y \in \mathtt{C}^C\right]\}$

   semantics (data property) $\{x\colon \exists y\colon \left[(x,y) \in \mathtt{P}^{DP} \wedge y \in \mathtt{C}^{DT}\right]\}$

   Manchester `P some C`

   Turtle

```
_:x a owl:Restriction ;
    owl:onProperty P ;
    owl:someValuesFrom C .
```

5. **value restriction** an object is related with an object property (resp. a data property) `P` to an individual (resp. a literal) `a`

   DL $\exists \mathtt{P}.\{\mathtt{a}\}$

   semantics (object property) $\{x\colon (x, \mathtt{a}^I) \in \mathtt{P}^{OP}\}$

   semantics (data property) $\{x\colon (x, \mathtt{a}^{LT}) \in \mathtt{P}^{DP}\}$

   Manchester `P value a`

   Turtle

   ```
   _:x a owl:Restriction ;
       owl:onProperty P ;
       owl:hasValue a .
   ```

6. **self-restriction** an object is related with an object property `P` with itself

   semantics $\{x\colon (x, x) \in \mathtt{P}^{OP}\}$

   Manchester `P Self`

   Turtle

   ```
   _:x rdf:type owl:Restriction ;
       owl:onProperty P ;
       owl:hasSelf "true"^^xsd:boolean .
   ```

   (While the datatype `xsd:boolean` is forbidden in OWL 2 EL, its usage here is merely due to the RDF serialization and does not increase the reasoning complexity.)

A.3 Axioms

An OWL 2 EL axiom is one of the following:

1. **class inclusion** a class expression `C` is a subclass of a class expression `D`

   DL $\mathtt{C} \sqsubseteq \mathtt{D}$

   semantics $\mathtt{C}^C \subseteq \mathtt{D}^C$

   Manchester

   ```
   Class: C
   SubClassOf: D
   ```

   (`C` must be a class.)

   Turtle `C rdfs:subClassOf D .`

2. **class equivalence** a class expression `C` is equivalent to a class expression `D`

   DL $\mathtt{C} \equiv \mathtt{D}$

   semantics $\mathtt{C}^C = \mathtt{D}^C$

   Manchester

```
Class: C
EquivalentTo: D
```

(C must be a class.)

Turtle `C owl:equivalentClass D .`

3. **class disjointness** a class expression C is disjoint with a class expression D

DL $\mathtt{C} \sqcap \mathtt{D} \equiv \bot$

semantics $\mathtt{C}^C \cap \mathtt{D}^C = \emptyset$

Manchester

```
Class: C
DisjointWith: D
```

(C must be a class.)

Turtle `C owl:disjointWith D .`

4. **object property inclusion** an object property P is a subproperty of an object property Q

DL $\mathtt{P} \sqsubseteq \mathtt{Q}$

semantics $\mathtt{P}^{OP} \sqsubseteq \mathtt{Q}^{OP}$

Manchester

```
ObjectProperty: P
SubPropertyOf: Q
```

Turtle `P rdfs:subPropertyOf Q .`

5. **object property inclusion with a property chain** composition of object properties $\mathtt{P}_1$, $\mathtt{P}_2$, ..., $\mathtt{P}_n$ is a subproperty of an object property Q

DL $\mathtt{P}_1 \circ \mathtt{P}_2 \circ \ldots \circ \mathtt{P}_n \sqsubseteq \mathtt{Q}$

semantics $\forall y_0, \ldots, y_n \colon \left[\left((y_0, y_1) \in \mathtt{P}_1^{OP} \wedge (y_1, y_2) \in \mathtt{P}_2^{OP} \wedge \ldots \wedge (y_{n-1}, y_n) \in \mathtt{P}_n^{OP}\right) \rightarrow (y_0, y_n) \in \mathtt{Q}^{OP}\right]$

Manchester

```
ObjectProperty: Q
SubPropertyChain: P1 o P2 o ... o Pn
```

Turtle `Q owl:propertyChainAxiom (P1 P2 ... Pn) .`

6. **data property inclusion** a data property P is a subproperty of a data property Q

DL $\mathtt{P} \sqsubseteq \mathtt{Q}$

semantics $\mathtt{P}^{DP} \sqsubseteq \mathtt{Q}^{DP}$

Manchester

```
DataProperty: P
SubPropertyOf: Q
```

Turtle `P rdfs:subPropertyOf Q .`

7. **object property equivalence** an object property P is equivalent to an object property Q

DL $\mathtt{P} \equiv \mathtt{Q}$

semantics $\mathtt{P}^{OP} \equiv \mathtt{Q}^{OP}$

Manchester

```
ObjectProperty: P
EquivalentTo: Q
```

Turtle `P owl:equivalentProperty Q .`

8. **data property equivalence** a data property `P` is equivalent to a data property `Q`

DL $\mathtt{P} \equiv \mathtt{Q}$

semantics $\mathtt{P}^{OP} \equiv \mathtt{Q}^{OP}$

Manchester

```
DataProperty: P
EquivalentTo: Q
```

Turtle `P owl:equivalentProperty Q .`

9. **transitive object properties** an object property `P` is transitive

DL $\mathtt{P} \circ \mathtt{P} \sqsubseteq \mathtt{P}$

semantics $\forall x, y, z\colon \left[\left((x, y) \in \mathtt{P}^{OP} \wedge (y, z) \in \mathtt{P}^{OP}\right) \rightarrow (x, z) \in \mathtt{P}^{OP}\right]$

Manchester

```
ObjectProperty: P
Characteristics: Transitive
```

Turtle `P a owl:TransitiveProperty .`

10. **reflexive object properties** an object property `P` is reflexive

semantics $\forall x\colon \left(x \in \Delta_I \rightarrow (x, x) \in \mathtt{P}^{OP}\right)$

Manchester

```
ObjectProperty: P
Characteristics: Reflexive
```

Turtle `P a owl:ReflexiveProperty .`

11. **domain restriction** a class expression `C` is the *domain* of a *property* `P`

DL $dom(\mathtt{P}) \sqsubseteq \mathtt{C}$

semantics (object property) $\forall x, y\colon \left((x, y) \in \mathtt{P}^{OP} \rightarrow x \in \mathtt{C}^{C}\right)$

semantics (data property) $\forall x, y\colon \left((x, y) \in \mathtt{P}^{DP} \rightarrow x \in \mathtt{C}^{C}\right)$

Manchester (object property)

```
ObjectProperty: P
Domain: C
```

Manchester (data property)

```
DataProperty: P
Domain: C
```

Turtle `P rdfs:domain C .`

12. **range restriction** an OWL 2 EL class expression (resp. a data range) `C` is the range of an object property (resp. a data property) `P`

    DL $ran(\mathtt{P}) \sqsubseteq \mathtt{C}$

    semantics (object property) $\forall x, y\colon \big((x,y) \in \mathtt{P}^{OP} \rightarrow y \in \mathtt{C}^{C}\big)$

    semantics (data property) $\forall x, y\colon \big((x,y) \in \mathtt{P}^{DP} \rightarrow y \in \mathtt{C}^{DT}\big)$

    Manchester (object property)

    ```
    ObjectProperty: P
    Range: C
    ```

    Manchester (data property)

    ```
    DataProperty: P
    Range: C
    ```

    Turtle `P rdfs:range C .`

13. **same individuals** an individual `a` is the same as an individual `b`

    DL $\{\mathtt{a}\} \equiv \{\mathtt{b}\}$

    semantics $\mathtt{a}^I = \mathtt{b}^I$

    Manchester

    ```
    Individual: a
    SameAs: b
    ```

    Turtle `a owl:sameAs b .`

14. **different individuals** an individual `a` is different from an individual `b`

    DL $\{\mathtt{a}\} \sqcap \{\mathtt{b}\} \sqsubseteq \bot$

    semantics $\mathtt{a}^I \neq \mathtt{b}^I$

    Manchester

    ```
    Individual: a
    DifferentFrom: b
    ```

    Turtle `a owl:differentFrom b .`

15. **class assertion** an individual `b` has type `C` (where `C` is a class expression)

    DL $\mathtt{C}(\mathtt{b})$

    semantics $\mathtt{b}^I \in \mathtt{C}^C$

    Manchester

    ```
    Individual: b
    Types: C
    ```

    Turtle `b rdf:type C .`

16. **property assertion** an individual `b` is in relation `P` with an individual (resp. a literal) `c`, where `P` is an object property (resp. a data property)

    DL $\mathtt{P}(\mathtt{b}, \mathtt{c})$

semantics (object property) $(\mathtt{b}^I, \mathtt{c}^I) \in \mathtt{P}^{OP}$

semantics (data property) $(\mathtt{b}^I, \mathtt{c}^{DT}) \in \mathtt{P}^{DP}$

Manchester

```
Individual: b
Facts: P c
```

Turtle `b P c .`

17. **negative property assertion** an individual `b` is not in relation `P` with (resp. a literal) `c`, where `P` is an object property (resp. a data property)

semantics (object property) $(\mathtt{b}^I, \mathtt{c}^I) \notin \mathtt{P}^{OP}$

semantics (data property) $(\mathtt{b}^I, \mathtt{c}^{DT}) \notin \mathtt{P}^{DP}$

Manchester

```
Individual: b
Facts: not P c
```

Turtle

```
_:x a owl:NegativePropertyAssertion ;
    owl:sourceIndividual b ;
    owl:assertionProperty P ;
    owl:targetIndividual c .
```

18. **functional data properties** an object property `P` is functional

semantics $\forall x, y, z\colon \left[\left((x, y) \in \mathtt{P}^{DP} \wedge (x, z) \in \mathtt{P}^{DP}\right) \rightarrow y = z\right]$

Manchester

```
DataProperty: P
Characteristics: Functional
```

Turtle `P a owl:FunctionalProperty .`

19. **keys** a sequence of object properties $\mathtt{P_1}, \mathtt{P_2}, \ldots, \mathtt{P_k}$ together with a sequence of data properties $\mathtt{P_{k+1}}, \ldots, \mathtt{P_n}$ are a primary key for a class expression `C`

semantics

$$\forall x, y, z_1, \ldots, z_n\colon \big(\big[x \in \mathtt{C}^C \wedge x \in NAMED \wedge y \in \mathtt{C}^C \wedge y \in NAMED \wedge (x, z_1) \in \mathtt{P}_1^{OP} \wedge (y, z_1) \in \mathtt{P}_1^{OP} \wedge \ldots (x, z_k) \in \mathtt{P}_k^{OP} \wedge (y, z_k) \in \mathtt{P}_k^{OP} \wedge (x, z_{k+1}) \in \mathtt{P}_{k+1}^{DP} \wedge (y, z_{k+1}) \in \mathtt{P}_1^{DP} \wedge \ldots (x, z_n) \in \mathtt{P}_n^{DP} \wedge (y, z_n) \in \mathtt{P}_1^{DP}\big] \rightarrow x = y\big)$$

$NAMED$ is the subset of domain corresponding to all the named individuals.

Manchester

```
Class: C
HasKey: P1 P2 ... Pn
```

Turtle `C owl:hasKey (P1 P2 ... Pn) .`

Index

Bibliography

[1] Logistic regression. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, pages 631–631. Springer US, Boston, MA, 2010.

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. DBpedia: A nucleus for a web of open data. In Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007.*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2007.

[3] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 364–369. Professional Book Center, 2005.

[4] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York, NY, USA, 2003.

[5] Franz Baader, Bernhard Ganter, Baris Sertkaya, and Ulrike Sattler. Completing description logic knowledge bases using formal concept analysis. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 230–235, 2007.

[6] Franz Baader, Carsten Lutz, and Sebastian Brandt. Pushing the EL envelope further. In Kendall Clark and Peter F. Patel-Schneider, editors, *Proceedings of the Fourth OWLED Workshop on OWL: Experiences and Directions, Washington, DC, USA, 1-2 April 2008.*, volume 496 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[7] Tim Berners-Lee. Linked data. Retrieved at 2016-09-26 from `https://www.w3.org/DesignIssues/LinkedData.html`.

[8] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[9] Mark Birbeck, Ben Adida, Ivan Herman, and Manu Sporny. RDFa 1.1 primer - third edition. W3C note, W3C, March 2015. `http://www.w3.org/TR/2015/NOTE-rdfa-primer-20150317/`.

[10] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[11] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - A crystallization point for the web of data. *J. Web Sem.*, 7(3):154–165, 2009.

[12] Yannick Le Bras, Philippe Lenca, and Stéphane Lallich. Optimonotone measures for optimal rule discovery. *Computational Intelligence*, 28(4):475–504, 2012.

[13] Jeen Broekstra and Arjohn Kampman. An RDF query and transformation language. In Steffen Staab and Heiner Stuckenschmidt, editors, *Semantic Web and Peer-to-Peer: Decentralized Management and Exchange of Knowledge and Information*, pages 23–39. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[14] Lorenz Bühmann, Daniel Fleischhacker, Jens Lehmann, André Melo, and Johanna Völker. Inductive lexical learning of class expressions. In Krzysztof Janowicz, Stefan Schlobach, Patrick Lambrix, and Eero Hyvönen, editors, *Knowledge Engineering and Knowledge Management - 19th International Conference, EKAW 2014, Linköping, Sweden, November 24-28, 2014. Proceedings*, volume 8876 of *Lecture Notes in Computer Science*, pages 42–53. Springer, 2014.

[15] Lorenz Bühmann and Jens Lehmann. Pattern based knowledge base enrichment. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul T. Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web - ISWC 2013*, volume 8218 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2013.

[16] Lorenz Bühmann, Jens Lehmann, and Patrick Westphal. DL-Learner - A framework for inductive learning on the semantic web. *J. Web Sem.*, 39:15–24, 2016.

[17] Peter Burmeister and Richard Holzer. Treating incomplete knowledge in formal concept analysis. In Bernhard Ganter, Gerd Stumme, and Rudolf Wille, editors, *Formal Concept Analysis: Foundations and Applications*, pages 114–126. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[18] Gavin Carothers and Eric Prud'hommeaux. RDF 1.1 turtle. W3C recommendation, W3C, February 2014. `http://www.w3.org/TR/2014/REC-turtle-20140225/`.

[19] Bonaventura Coppola, Aldo Gangemi, Alfio Massimiliano Gliozzo, Davide Picca, and Valentina Presutti. Frame detection over the semantic web. In Lora Aroyo, Paolo Traverso, Fabio Ciravegna, Philipp Cimiano, Tom Heath, Eero Hyvönen, Riichiro Mizoguchi, Eyal Oren, Marta Sabou, and Elena Paslaru Bontas Simperl, editors, *The Semantic Web: Research and Applications, 6th European Semantic Web Conference, ESWC 2009, Heraklion, Crete, Greece, May 31-June 4, 2009, Proceedings*, volume 5554 of *Lecture Notes in Computer Science*, pages 126–142. Springer, 2009.

[20] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[21] M. Duerst and M. Suignard. Internationalized resource identifiers (iris). RFC 3987, RFC Editor, January 2005. `http://www.rfc-editor.org/rfc/rfc3987.txt`.

[22] F. Baader, S. Brandt, and C. Lutz. Pushing the el envelope. LTCS-Report LTCS-05-01, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2005. `http://lat.inf.tu-dresden.de/research/reports.html`.

[23] Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. DL-FOIL concept learning in description logics. In Filip Zelezný and Nada Lavrac, editors, *Inductive Logic Programming, 18th International Conference, ILP 2008, Prague, Czech Republic, September 10-12, 2008, Proceedings*, volume 5194 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2008.

[24] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, RFC Editor, June 1999. `http://www.rfc-editor.org/rfc/rfc2616.txt`.

[25] Daniel Fleischhacker and Johanna Völker. Inductive learning of disjointness axioms. In Robert Meersman, Tharam S. Dillon, Pilar Herrero, Akhil Kumar, Manfred Reichert, Li Qing, Beng Chin Ooi, Ernesto Damiani, Douglas C. Schmidt, Jules White, Manfred Hauswirth, Pascal Hitzler, and Mukesh K. Mohania, editors, *On the Move to Meaningful Internet Systems: OTM 2011*, volume 7045 of *Lecture Notes in Computer Science*, pages 680–697. Springer, 2011.

[26] Yongjian Fu and Jiawei Han. Meta-rule-guided mining of association rules in relational databases. In *KDOOD/TDOOD*, pages 39–46, 1995.

[27] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.*, 24(6):707–730, 2015.

[28] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In Daniel Schwabe, Virgílio A. F. Almeida, Hartmut Glaser, Ricardo A. Baeza-Yates, and Sue B. Moon, editors, *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 413–422. International World Wide Web Conferences Steering Committee / ACM, 2013.

[29] Bernardo Cuenca Grau, Peter Patel-Schneider, and Boris Motik. OWL 2 web ontology language direct semantics (second edition). W3C recommendation, W3C, December 2012. `http://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/`.

[30] Aryeh Gregor, Alex Russell, Anne van Kesteren, Robin Berjon, and Ms2ger. W3C DOM4. Last call WD, W3C, July 2014. `http://www.w3.org/TR/2014/WD-dom-20140710/`.

[31] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.*, pages 1–12. ACM, 2000.

[32] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004.

[33] Steven Harris and Andy Seaborne. SPARQL 1.1 query language. W3C recommendation, W3C, March 2013. `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/`.

[34] Sandro Hawke, Boris Motik, Jie Bao, Axel Polleres, and Peter Patel-Schneider. rdf:plainliteral: A datatype for RDF plain literals (second edition). W3C recommendation, W3C, December 2012. `http://www.w3.org/TR/2012/REC-rdf-plain-literal-20121211/`.

[35] Amac Herdagdelen and Marco Baroni. The concept game: Better commonsense knowledge extraction by combining text mining and a game with a purpose. In *Commonsense Knowledge, Papers from the 2010 AAAI Fall Symposium, Arlington, Virginia, USA, November 11-13, 2010*, volume FS-10-02 of *AAAI Technical Report*. AAAI, 2010.

[36] Johannes Hoffart, Fabian M. Suchanek, Klaus Berberich, and Gerhard Weikum. YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. *Artif. Intell.*, 194:28–61, 2013.

[37] Matthew Horridge and Sean Bechhofer. The OWL API: A java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.

[38] Matthew Horridge and Peter Patel-Schneider. OWL 2 web ontology language manchester syntax (second edition). W3C note, W3C, December 2012. `http://www.w3.org/TR/2012/NOTE-owl2-manchester-syntax-20121211/`.

[39] Matthew Horridge, Tania Tudorache, Csongor Nyulas, Jennifer Vendetti, Natalya Fridman Noy, and Mark A. Musen. Webprotégé: a collaborative web-based platform for editing biomedical ontologies. *Bioinformatics*, 30(16):2384–2385, 2014.

[40] Oana Inel, Khalid Khamkham, Tatiana Cristea, Anca Dumitrache, Arne Rutjes, Jelle van der Ploeg, Lukasz Romaszko, Lora Aroyo, and Robert-Jan Sips. Crowdtruth: Machine-human computation framework for harnessing disagreement in gathering annotated data. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig A. Knoblock, Denny Vrandecic, Paul T. Groth, Natasha F. Noy, Krzysztof Janowicz, and Carole A. Goble, editors, *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II*, volume 8797 of *Lecture Notes in Computer Science*, pages 486–504. Springer, 2014.

[41] Piotr Jakubowski. Ocena jakosci baz wiedzy na platformie crowdsourcingowej. Master's thesis, Poznan University of Technology, 2015.

[42] S. Josefsson. The base16, base32, and base64 data encodings. RFC 3548, RFC Editor, July 2003. `https://tools.ietf.org/html/rfc3548`.

[43] Sham M. Kakade, Adam Kalai, Varun Kanade, and Ohad Shamir. Efficient learning of generalized linear and single index models with isotonic regression. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 927–935, 2011.

[44] Kaarel Kaljurand and Norbert E. Fuchs. Verbalizing OWL in attempto controlled english. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions, Innsbruck, Austria, June 6-7, 2007*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[45] Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. RQL: a declarative query language for RDF. In David Lassner, Dave De Roure, and Arun Iyengar, editors, *Proceedings of the Eleventh International World Wide Web Conference, WWW 2002, May 7-11, 2002, Honolulu, Hawaii*, pages 592–603. ACM, 2002.

[46] Yevgeny Kazakov, Markus Krötzsch, and František Simančík. The incredible ELK. *Journal of Automated Reasoning*, 53(1):1–61, 2013.

[47] Rohit Khare and Tantek Çelik. Microformats: A pragmatic path to the semantic web. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 865–866, New York, NY, USA, 2006. ACM.

[48] Holger Knublauch, Ray W. Fergerson, Natalya Fridman Noy, and Mark A. Musen. The Protégé OWL plugin: An open development environment for semantic web applications. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *The Semantic Web - ISWC 2004: Third International Semantic Web Conference,Hiroshima, Japan, November 7-11, 2004. Proceedings*, volume 3298 of *Lecture Notes in Computer Science*, pages 229–243. Springer, 2004.

[49] Agnieszka Lawrynowicz and Jedrzej Potoniec. Fr-ont: An algorithm for frequent concept mining with formal ontologies. In Marzena Kryszkiewicz, Henryk Rybinski, Andrzej Skowron, and Zbigniew W. Ras, editors, *Foundations of Intelligent Systems - 19th International Symposium, ISMIS 2011, Warsaw, Poland, June 28-30, 2011. Proceedings*, volume 6804 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 2011.

[50] Agnieszka Lawrynowicz and Jedrzej Potoniec. Pattern based feature construction in semantic data mining. *Int. J. Semantic Web Inf. Syst.*, 10(1):27–65, 2014.

[51] Jens Lehmann. DL-Learner: Learning concepts in description logics. *Journal of Machine Learning Research*, 10:2639–2642, 2009.

[52] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[53] Huiying Li and Qiang Sima. Parallel mining of OWL 2 EL ontology from large linked datasets. *Knowl.-Based Syst.*, 84:10–17, 2015.

[54] Charles X. Ling and Victor S. Sheng. Cost-sensitive learning. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, pages 231–235. Springer, 2010.

[55] Francesca A. Lisi and Floriana Esposito. Learning SHIQ+log rules for ontology evolution. In Aldo Gangemi, Johannes Keizer, Valentina Presutti, and Heiko Stoermer, editors, *Proceedings of the 5th Workshop on Semantic Web Applications and Perspectives (SWAP2008), Rome, Italy, December 15-17, 2008*, volume 426 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[56] Birte Lönneker-Rodman and Collin F. Baker. The FrameNet model and its applications. *Natural Language Engineering*, 15(3):415–453, 2009.

[57] Despoina Magka, Yevgeny Kazakov, and Ian Horrocks. Tractable extensions of the description logic $\mathcal{EL}$ with numerical datatypes. *Journal of Automated Reasoning*, 47(4):427–450, 2011.

[58] Brian McBride. Jena: A semantic web toolkit. *IEEE Internet Computing*, 6(6):55–59, 2002.

[59] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. YALE: rapid prototyping for complex data mining tasks. In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven, and Dimitrios Gunopulos, editors, *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 935–940. ACM, 2006.

[60] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.

[61] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, 2015.

[62] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Achille Fokoue, and Zhe Wu. OWL 2 web ontology language profiles (second edition). W3C recommendation, W3C, December 2012. `http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/`.

[63] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau reasoning for description logics. *J. Artif. Intell. Res. (JAIR)*, 36:165–228, 2009.

[64] Mark A. Musen. The protégé project: a look back and a look forward. *AI Matters*, 1(4):4–12, 2015.

[65] Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. Scalable knowledge harvesting with high precision and high recall. In Irwin King, Wolfgang Nejdl, and Hang Li, editors, *Proceedings of the Forth International Conference on Web Search and Web Data Mining, WSDM 2011, Hong Kong, China, February 9-12, 2011*, pages 227–236. ACM, 2011.

[66] Vít Novácek, Siegfried Handschuh, and Stefan Decker. Getting the meaning right: A complementary distributional layer for the web semantics. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Fridman Noy, and Eva Blomqvist, editors, *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, volume 7031 of *Lecture Notes in Computer Science*, pages 504–519. Springer, 2011.

[67] Natalya Fridman Noy, Michael Sintek, Stefan Decker, Monica Crubézy, Ray W. Fergerson, and Mark A. Musen. Creating semantic web contents with protégé-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.

[68] Peter Patel-Schneider, Bijan Parsia, and Boris Motik. OWL 2 web ontology language structural specification and functional-style syntax (second edition). W3C recommendation, W3C, December 2012. `http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/`.

[69] Heiko Paulheim and Christian Bizer. Type inference on noisy RDF data. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul T. Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, volume 8218 of *Lecture Notes in Computer Science*, pages 510–525. Springer, 2013.

[70] A. Phillips and M. Davis. Tags for identifying languages. BCP 47, RFC Editor, September 2009. https://tools.ietf.org/html/rfc5646.

[71] Jedrzej Potoniec. Towards ontology refinement by combination of machine learning and attribute exploration. In Patrick Lambrix, Eero Hyvönen, Eva Blomqvist, Valentina Presutti, Guilin Qi, Uli Sattler, Ying Ding, and Chiara Ghidini, editors, *Knowledge Engineering and Knowledge Management - EKAW 2014 Satellite Events, VISUAL, EKM1, and ARCOE-Logic, Linköping, Sweden, November 24-28, 2014. Revised Selected Papers.*, volume 8982 of *Lecture Notes in Computer Science*, pages 225–232. Springer, 2014.

[72] Jedrzej Potoniec, Piotr Jakubowski, and Agnieszka Ławrynowicz. Swift Linked Data Miner: Mining OWL 2 EL class expressions directly from online RDF datasets . *Joural of Web Semantics*. DOI: 10.1016/j.websem.2017.08.001.

[73] Jedrzej Potoniec and Agnieszka Lawrynowicz. RMonto: Ontological extension to RapidMiner. In *Poster and Demo Session of the ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany*, 2011.

[74] Jedrzej Potoniec and Agnieszka Lawrynowicz. Combining ontology class expression generation with mathematical modeling for ontology learning. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 4198–4199. AAAI Press, 2015.

[75] Jedrzej Potoniec and Agnieszka Lawrynowicz. A Protégé plugin with Swift Linked Data Miner. In Takahiro Kawamura and Heiko Paulheim, editors, *Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016.*, volume 1690 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[76] Jedrzej Potoniec, Sebastian Rudolph, and Agnieszka Lawrynowicz. Towards combining machine learning with attribute exploration for ontology refinement. In Matthew Horridge, Marco Rospocher, and Jacco van Ossenbruggen, editors, *Proceedings of the ISWC 2014 Posters & Demonstrations Track a track within the 13th International Semantic Web Conference, ISWC 2014, Riva del Garda, Italy, October 21, 2014.*, volume 1272 of *CEUR Workshop Proceedings*, pages 229–232. CEUR-WS.org, 2014.

[77] J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[78] Riccardo Rosati. Dl+log: Tight integration of description logics and disjunctive datalog. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, pages 68–78. AAAI Press, 2006.

[79] Viachaslau Sazonau, Uli Sattler, and Gavin Brown. General terminology induction in OWL. In Marcelo Arenas, Óscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul T. Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan, and Steffen Staab, editors, *The Semantic Web - ISWC 2015*, volume 9366 of *Lecture Notes in Computer Science*, pages 533–550. Springer, 2015.

[80] Max Schmachtenberg, Christian Bizer, Anja Jentzsch, and Richard Cyganiak. Linking open data cloud diagram 2014. Distributed under a CC-BY-SA licence, retrieved at 2016-09-26 from `http://lod-cloud.net/`.

[81] Rolf Schwitter and Norbert E. Fuchs. Attempto controlled english (ACE) A seemingly informal bridgehead in formal territory (poster abstract). In Michael J. Maher, editor, *Logic Programing, Proceedings of the 1996 Joint International Conference and Syposium on Logic Programming, Bonn, Germany, September 2-6, 1996*, page 536. MIT Press, 1996.

[82] Michael Sintek and Stefan Decker. TRIPLE - an RDF query, inference, and transformation language. In *Proceedings of the 14th International Conference on Applications of Prolog, INAP 2001, Univeristy of Tokyo, Tokyo, Japan, October 20-22, 2001*, pages 47–56. The Prolog Association of Japan, 2001.

[83] Katharina Siorpaes and Martin Hepp. Games with a purpose for the semantic web. *IEEE Intelligent Systems*, 23(3):50–60, 2008.

[84] Katharina Siorpaes and Martin Hepp. Ontogame: Weaving the semantic web by online games. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008, Proceedings*, volume 5021 of *Lecture Notes in Computer Science*, pages 751–766. Springer, 2008.

[85] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.

[86] Tomasz Sosnowski. Rozszerzenie WebProtégé o możliwość indukcji aksjomatów z powiazanych danych. Bachelor's thesis, Poznan University of Technology, 2016.

[87] Tomasz Sosnowski, Jedrzej Potoniec, and Agnieszka Ławrynowicz. Swift linked data miner-extension for webprotégé. In P. Ciancarini, F. Poggi, M. Horridge, J. Zhao, T. Groza, M. C. Suárez-Figueroa, M. d'Aquin, and V. Presutti, editors, *Knowledge Engineering and Knowledge Management - EKAW 2016 Satellite Events, EKM and Drift-a-LOD. Bologna, Italy, November 19–23, 2016. Revised Selected Papers*, volume 10180 of *Lecture Notes in Computer Science*. Springer, 2017.

[88] Michael Sperberg-McQueen, Eve Maler, Tim Bray, François Yergeau, Jean Paoli, and John Cowan. Extensible markup language (XML) 1.1 (second edition). W3C recommendation, W3C, August 2006. `http://www.w3.org/TR/2006/REC-xml11-20060816`.

[89] Michael Sperberg-McQueen, Ashok Malhotra, Paul V. Biron, Sandy Gao, Henry Thompson, and David Peterson. W3C xml schema definition language (XSD) 1.1 part 2: Datatypes. W3C recommendation, W3C, April 2012. `http://www.w3.org/TR/2012/REC-xmlschema11-2-20120405/`.

[90] Fabian M. Suchanek, Mauro Sozio, and Gerhard Weikum. SOFIE: a self-organizing framework for information extraction. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 631–640. ACM, 2009.

[91] Tania Tudorache, Csongor Nyulas, Natalya Fridman Noy, and Mark A. Musen. Webprotégé: A collaborative ontology editor and knowledge acquisition tool for the web. *Semantic Web*, 4(1):89–99, 2013.

[92] Paul E. Utgoff. Incremental learning. In Claude Sammut and Geoffrey I. Webb, editors, *Encyclopedia of Machine Learning*, pages 515–518. Springer US, Boston, MA, 2010.

[93] Johanna Völker, Daniel Fleischhacker, and Heiner Stuckenschmidt. Automatic acquisition of class disjointness. *J. Web Sem.*, 35:124–139, 2015.

[94] Johanna Völker and Mathias Niepert. Statistical schema induction. In Grigoris Antoniou, Marko Grobelnik, Elena Paslaru Bontas Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Z. Pan, editors, *The Semantic Web: Research and Applications*, volume 6643 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 2011.

[95] Luis von Ahn. Games with a purpose. *IEEE Computer*, 39(6):92–94, 2006.

[96] Denny Vrandecic. The rise of wikidata. *IEEE Intelligent Systems*, 28(4):90–95, 2013.

[97] Denny Vrandecic and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, 2014.

[98] Rudolf Wille. Formal concept analysis as mathematical theory of concepts and concept hierarchies. In Bernhard Ganter, Gerd Stumme, and Rudolf Wille, editors, *Formal Concept Analysis: Foundations and Applications*, pages 1–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[99] Gregory Williams, Elias Torres, Kendall Clark, and Lee Feigenbaum. SPARQL 1.1 protocol. W3C recommendation, W3C, March 2013. `http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/`.

[100] Gerhard Wohlgenannt, Marta Sabou, and Florian Hanika. Crowd-based ontology engineering with the ucomp protégé plugin. *Semantic Web*, 7(4):379–398, 2016.

[101] David Wood, Markus Lanthaler, and Richard Cyganiak. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, February 2014. `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`.

[102] Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005, 2004.

[103] Harry Zhang. The optimality of naive bayes. In Valerie Barr and Zdravko Markov, editors, *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, Miami Beach, Florida, USA*, pages 562–567. AAAI Press, 2004.