Increasing the Usability of Graph Databases by Learning SPARQL Queries and RDF Data

Gonzalo I. Diaz

Keble College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2018

A la emancipación del pueblo chileno.

Abstract

Semantic Web technologies and other open standards have the potential of allowing current open datasets and knowledge bases to become more interlinked and interoperable, providing interfaces for end-users to access data via powerful high-level query languages such as SPARQL. As a consequence, it becomes ever more important for these data to be efficiently and easily searchable by non-expert audiences. The challenge of increasing the usability of these database systems, therefore, becomes central. In this thesis, we approach the problem of increasing the usability of querying graph data, in the form of RDF knowledge bases, from different perspectives. We study the learning — more specifically, the reverse engineering — of SPARQL queries, including its theoretical and practical aspects. Along the way we find that a key limitation of these approaches is the completeness of the data, and therefore turn to learning — in this case, knowledge base completion — of RDF data.

Specifically, we begin by studying the definability problem for first-order logic, providing exact complexity bounds. We next provide a theoretical study of reverse engineering in the SPARQL context, formalising variants of the reverse engineering problem and giving bounds on their complexity. We develop algorithms for reverse engineering and perform experimental analyses, showing that they scale well. Additionally, we implement and present a proof-of-concept user application which demonstrates how reverse engineering is capable of guiding users who are unfamiliar with both the dataset and with SPARQL to desired queries and result sets based on a query by example paradigm. Finally, to address the issue of the completeness of data, we propose a scalable and ontology-aware graph embedding model which allows for fact inference in RDF datasets, providing a data learning approach that is complementary to query learning.

Acknowledgments

I am grateful to my D. Phil. supervisor, Prof. Michael Benedikt, and to my main collaborator, Prof. Marcelo Arenas, for providing excellent guidance and encouragement.

I would also like to thank my colleagues and collaborators. To Egor Kostylev, Achille Fokoue and Mohammad Sadoghi for sharing their experience and for enriching me with their advice and work ethic. To Ben Spencer and Max Schleich for endless conversations about work and life.

I thank the Comisión Nacional de Ciencias y Tecnología (CONICYT) of the Government of Chile for funding my education. I am excited to return to Chile and contribute to the development of my country, with the knowledge I have gained thanks to the collective effort of our people.

A huge shout out to all my friends, and especially to Rodrigo, Paula, Pame, Lore, and Juan Carlos. It has been my honour to share such awe inspiring times with them.

I thank my parents, Marco and Cecilia, and my brother Alejandro, for always supporting me.

Finally, I thank my wife, Macarena, for being my lifeline. I could not have done this without you.

Declaration

This thesis is based on original research and contains no material that has already been accepted, or is concurrently being submitted, for any degree or diploma or certificate or other qualification in this university or elsewhere.

To the best of my knowledge and belief this thesis contains no material previously published or written by another person, except where due reference is made in the text.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Currently, databases are quickly growing in size, in variety, and in quality, with a widening range of applications, from more classical use cases such as company databases, to publicly available government data [75, see government subcloud], encyclopaedic knowledge bases [18, 50], and newer applications such as the Internet of Things [78]. Semantic Web technologies [45] and other open standards have the potential of allowing these data to become interlinked and interoperable, as is the case with the linked open data cloud [24, 75], for example. However, it becomes ever more important for these data to be efficiently, and perhaps more importantly, *easily* searchable by a potential audience which no longer consists mainly of database administrators but ordinary users. The challenge of increasing the usability of database systems, therefore, becomes central.

Semantic Web technologies such as the Resource Description Framework (RDF) [73] and its query language, the SPARQL Protocol and Query Language (SPARQL is its recursive acronym) [112], offer the possibility of opening up the use of public datasets to a great variety of ordinary users. The relatively low-level data model used by RDF, namely triples (i.e. 3-tuples formed by objects extracted from a well-defined universe) which model basic subject-predicate-object statements, has allowed owners and producers of data to publish them with a common format. Thus, a cosmos of linked open data has grown from the interlinking of diverse datasets that range from high-level ontologies, which describe classes of concepts and the relations that may exist between them [73, 102], to very specific datasets such as scientific knowledge bases [75, life sciences subcloud], to open government data (usually available on open government websites), to large knowledge bases of varied facts, such as DBpedia [69] or Wikidata [111].

However, some of the key obstacles to the consumption of open data include the ease of querying, the interoperability of different databases, the availability of expressive ontologies which permit efficient calculation of inferred facts, and the incompleteness of data, among other issues. The unfamiliarity of users with the structure of data, as well as their unfamiliarity with the precise query languages they must use to access those data, however, will be our main focus in this thesis. This problem resonates with the broader issue of increasing the usability of database systems [63].

While increasing the usability of database systems can be achieved in many ways [63], including designing user-friendly or otherwise enhanced querying interfaces [42, 12, 66] and providing keyword-based or natural language querying [67, 90], in this thesis we focus on the study of extensions — or outright alternatives — to the classical querying paradigm wherein a user with knowledge of the database must formulate precise queries in a specific query language in order to obtain results.

The general goal of upending this classical querying paradigm (formulate query; execute query; obtain answers) has produced an array of related concepts and sub fields of research which are not immediately easy to distinguish, such as *query-by-example*, *query-by-output*, *query reverse engineering*, among others. What they do have in common, however, is the intention of relieving the user of the burden of having to formulate a query in a precise query language in order to obtain results.

Specifically, we approach the problem of increasing the usability of querying graph data, in the form of RDF knowledge bases, from different perspectives, ranging from the theoretical (e.g. studying the computational complexity of related decision problems) to the practical (implementing a proof-of-concept user interface for querying SPARQL endpoints by example). Along the way we will find that a key limitation of these *query-by-example* systems is the completeness of the data, and therefore turn to machine-learning approaches for knowledge-base completion, which allow the user to explore data and facts that are not *explicitly* contained in their dataset.

Initial forays into querying by example date back to a homonymous study by Zloof in 1975 [121], although this work arguably concerned itself with building a user-friendly syntax for formulating relational algebra queries rather than fundamentally altering the querying paradigm. The study of querying by example quickly diversified and deepened into reverse engineering and query-by-output approaches and the study of the definability problem (which will be discussed below). These fields progressed in parallel and in interaction with the advance of fields such as learning theory [7, 8, 106, 27] and usability of database systems [63].

The previous body of work mainly addresses the problem of learning the query, and this has an important limitation: in many cases the quality and completeness of the data itself may become an obstacle to a user wishing to obtain answers. Class information in graph knowledge bases is not always consistent across all entities; when dealing with hierarchical class ontologies, a given entity may be associated to a very specific subclass or the most general superclass, for example. Although modern semantic web databases include the possibility of inferring facts via specific rule sets [73], these rules usually refer to a restricted set of special keywords; there is a wealth of class inclusion information encoded with non-standard keywords, which limits the applicability of these schemes.

Hence, although learning a query can go a long way towards assisting a user in exploring a database, this process is essentially limited by the quality and completeness of the data, whereby studying forms of automatic knowledge base completion — in essence, learning the data — can be a powerful complement. Learning theory has given rise to machine learning techniques such as graph embeddings [79], which have been used to predict user preferences, entity matching, and knowledge base completion.

All in all, these are exciting times for the database community, as new applications and new techniques push the field to the forefront of computer science research. In the next section we describe in detail the contributions this thesis makes to the area.

1.1 Contributions

In a query reverse engineering scenario, which will be our main research object, the database and answer set become the *inputs* to the problem (and we will assume the inputs are to be provided by a user), while the query becomes the desired *output*, thus being *reverse engineered*. As a alternative querying paradigm, this scenario has been studied extensively for different data models, including relational databases, XML (Extensible Markup Language), and graph databases.

Increasing the level of specificity, a very basic — yet fundamental — form of query reverse engineering is the *definability decision problem*. Here, given a data model (e.g. relational databases) and a query language (e.g. relational algebra), a database instance D and an answer set R are given as inputs, the output being True if and only if there exists a query such that, when evaluated over D, gives R as an output. This formulation of the definability decision problem clearly illustrates how the standard querying paradigm is reversed, as the query loses its status as the quintessential input. The definability problem was initially studied in the context of the attempt to understand the expressivity of relational algebra; the work of Bancilhon and Paredaens in 1978 [83, 21] being the prime examples. In these works, a complexity upper bound for the relational algebra definability problem was obtained as a corollary of this work, although this fact was not mentioned explicitly at this stage. In addition to the case of (full) relational algebra, the definability problem has been studied for conjunctive queries [115] and for graph databases/query languages [9]. Slightly more general scenarios (accepting a *list* of instance-answer pairs) have also been studied for relational databases [47].

In this thesis, we have provided tight bounds for the first-order logic definability problem, thus closing an open question in the field. We do this by providing a polynomial-time algorithm for solving it, using calls to a graph-isomorphism subroutine (or oracle). As a consequence, the first-order definability problem is found to be complete for the class **GI** of all problems which are polynomial-time Turing reducible to the graph isomorphism problem, thus resolving its exact complexity. The technique used is also applied to a generalised version of the problem which accepts a finite set of relation pairs, and whose exact complexity was also open; this version is also found to be **GI**-complete. We also discuss practical applications that stem from this knowledge. In particular, the connection, via calls to a subroutine, to graph isomorphism allows algorithms for the definability problem to take advantage of efficient implementations of the graph isomorphism problem. This knowledge will hopefully lead to the implementation or improvement of query definability systems, thus furthering the objective of building usable database querying systems.

Although the previous results are useful in understanding the complexity of definability and have potential practical applications, from the point of view of a user faced with querying a database, the definability problem is far too restrictive to be considered a realistic scenario. A typical user will not have an exhaustive list of answers available, for example, especially if the database in question is extremely large. Thus, users who are unfamiliar with either the details of SPARQL or properties of the target dataset may find it easier to query *by example*, giving examples of the information they want (or examples of both what they want and what they do not want) and let the system *reverse engineer* the desired query from the examples. In this way, query reverse engineering generally allows for more flexible inputs, where each example in the answer set given can be labelled as either a positive or a negative example.

The corresponding decision problem then seeks to determine the existence of a query which is consistent with a set of labelled examples, i.e. a query which, when evaluated on the dataset, would return all positive examples and exclude all negative examples. These problems have been studied for XML data in conjunction with XPath (the standard query language for XML data) [99, 35] and for relational databases and join queries [109, 110], among others. In this context, we provide the first (to our knowledge) query reverse engineering study of RDF and SPARQL, formalising variants of the reverse engineering problem and giving tight bounds on its complexity. We have obtained complexity bounds for several variants of reverse engineering decision problems, including reverse engineering with only positive examples, with both positive and negative examples, and exact reverse engineering with a full answer set as input (a.k.a. the definability problem). We have found that these are inherently intractable problems, with exact complexities ranging from **NP** and **coNP** to Σ_2^p [1]. Notwithstanding the high complexity bounds, we provide practical algorithms for reverse engineering, showing that the more complex steps may be delegated to state-of-the-art SPARQL query engines, and providing experimental evidence that our algorithms perform well with both synthetic and real world inputs. We also implement a reverse engineering library for positive examples and perform an experimental analysis of the tool, showing that it scales well in the data size, number of examples, and in the size of the smallest query that fits the data.

The challenge of designing and building systems that assist in the querying of large databases constitutes an burgeoning area of research itself. Such systems have been built for join queries on relational databases, for example [29]. In the case of RDF databases and the SPARQL query language, we have built a system, which we name SPARQLByE, for querying RDF data by example, based on the reverse engineering algorithms developed. The SPARQLByE system centres the user interaction around a set of labelled examples, which are the main input. We demonstrate how reverse engineering enables our system to guide users who are unfamiliar with both the dataset and with SPARQL to the desired query and result set. This is achieved by supplementing reverse engineering of SPARQL with techniques for guiding the user to further positive and negative examples. The end objective is to converge towards a reverse engineered query which satisfies the user's expectation. We thus give evidence that reverse engineering tools can provide benefits on real-life datasets, beyond the theoretical computational properties of these problems.

In the final part of this thesis, we turn to graph embedding models as an alternative view on the problem of increasing the usability of database systems. Graph embeddings provide machine-learning-friendly representations of graph data and provide a tool for knowledge base completion, i.e. learning the data, a goal which complements the learning of queries in order to improve the information value that may be extracted from a dataset. The input to a graph embedding model is a graph database (e.g. an RDF graph), and the output is a mapping which associates to each entity in the graph an n-dimensional real-valued vector, which thus *embeds* said entity into the n-dimensional space. In particular, translational graph embedding models are able to perform efficiently for very large graphs and with high accuracy, when applied to standard link prediction and triplet classification tasks. In this context, recent work has focused on producing more expressive models that build on TransE, a particularly simple translational model which has achieved good results [31]. A key limitation of these models, however, is that they are agnostic with respect to the ontological information which is often present in large graph data. Ontological triples, if not outright removed, are treated as facts, and consequently the inferences produced by these models may produce triples which violate the semantics of the ontology. In this scenario, we design an ontology-aware graph embedding model which is able to model the semantics of RDFS keywords. We show EmbedS to be superior to simple models such as TransE and TransH [113], and provide experimental scenarios to ascertain whether the model is able to improve the quality of predictions in large, ontology-rich RDF graphs.

In summary, this thesis is concerned with improving the usability of RDF graph databases via the study of the definability and reverse engineering problems for the querying of SPARQL-RDF data, along with exploring the relation between Semantic Web ontologies and graph embedding models as a more structured way of predicting links in graph data.

This thesis is structured as follows: in Chapter 2 we study the *definability problem* for first-order logic, providing an exact complexity bound and discussing practical applications and extensions of the problem. In Chapter 3 we provide exact complexity bounds for several variants of the reverse engineering problem in the context of SPARQL, along with algorithmic considerations. In Chapter 4 we develop a web-based system for reverse engineering SPARQL queries with user interaction. In Chapter 5 we design a graph embedding model that allows for the inference of new data from global patterns in existing data. Finally, we discuss the ramifications of our results and further avenues of research in the Conclusion.

Chapter 2

First-order logic definability

2.1 Introduction

In a typical relational database querying scenario, a database instance I and a query Q are given and the objective is to *answer the query*, that is, calculate the answer relation R which Q produces for I. There are many real-world situations though, where the database I and answer relation R are known and it is the query that is unknown. For example, it is common for the results of a query to be published without the precise query being made available, requiring *reverse engineering* [109, 119]. In other cases, users of a database system may have difficulties formulating a query, in which case it is desirable to have the ability to *infer* or *learn* the query by having the user specify tuples which they want included in the result (i.e. positive examples), tuples which they want excluded from the result (i.e. negative examples), or a combination of both. Query learning has been studied in relational databases [4, 28] as well as in other data models such as XML [99, 35, 100], graph databases [26] and big data [27]. This situation also arises in data integration scenarios, where example source and target instances are used to derive data mapping queries between source and target [23, 53, 88, 106]. Finally, another application is checking whether a relation R is *redundant* in a database instance I, in the sense that there exists a query Q that produces R when evaluated over the other relations in I, implying that the information in R can be deduced from the other relations in the instance I [46].

A common ground for the different query discovery scenarios presented is the *definability problem* for a query language $\mathcal{Q}$. This decision problem takes as input an appropriate database instance I and answer R, and asks whether there exists a query $Q \in \mathcal{Q}$ such that Q evaluated on I results in R. Here, the semantics of the query language $\mathcal{Q}$ determines what appropriate database instances and answer sets are. In the case of first-order logic (without constants a linear order on the domain) we denote the definability problem FO-DEF, and the input is a relational database instance I and an answer relation R. The definability problem has been studied for relational databases and first-order logic (equivalently, relational algebra), as well other data models and query languages. In particular, this problem has been studied for nested relational databases and nested relational algebra [57, 59], for XML and XPath [60, 48], and for graph databases and conjunctive regular path queries [9].

The study of the computational complexity of FO-DEF dates back to 1978 [83, 21], where a *semantic characterisation* of the problem, based on automorphisms, placed FO-DEF in **coNP** (see Section 2.3 and [37] for more details). Although this provided a complexity upper bound for the problem, an exact complexity result has not been found since then [47, 105]; in particular, the problem was never found to be **coNP**-hard. Despite the open question for the first-order logic case, the corresponding definability problem for conjunctive queries was determined to be **coNEXPTIME**-complete [115]. Here, the complexity upper bound (i.e. the inclusion in **coNEXPTIME**) stems from an analogous semantic characterisation of the CQ definability problem in terms of polymorphisms [64]. In a different direction, a natural generalisation of FO-DEF, dubbed BP-PAIRS [47], accepts a finite set of relation pairs $\{(S_1, R_1), \ldots, (S_n, R_n)\}$ and asks whether there exists a first-order query Q such that $Q(S_i) = R_i$, for all $i \in [1, n]$. By means of an analogous semantic characterisation, the authors found BP-PAIRS to be included in **coNP**.

In this chapter, we provide a novel polynomial-time algorithm for the first-order logic definability problem, which uses calls to a graph-isomorphism subroutine (oracle). As a consequence of the existence of this algorithm, FO-DEF is found to be included in the complexity class **GI** (defined as the set of all languages which are polynomial-time Turing reducible to the graph isomorphism problem). Moreover, we also show that FO-DEF is **GI**-hard, which allows us to conclude that FO-DEF is **GI**-complete and, thus, allows us to close the open question regarding the exact complexity of FO-DEF. This result also has consequences in practical applications, as implementations for the definability problem may now take advantage of algorithmic optimisations for the graph isomorphism problem [87, 76, 20]. For example, for several restricted classes of graphs it is possible to solve the isomorphism problem in polynomial time [55], and this performance will be inherited by definability problem algorithms which use the characterisation presented in this chapter. Finally, we find that the technique used is also applicable to the BP-PAIRS problem and show that it is also included in **GI**, solving a problem left open in [47]. This chapter is based on work published in [10].

2.2 Preliminaries

Let U be an infinite countable universe. A *relational schema* $\mathbf{R} = \{R_1, \ldots, R_m\}$ is a set of *relation names*, each with an associated *arity*, denoted by $\text{arity}(R_i)$. Given a relational schema, a *relational instance I over* $\mathbf{R}$ is a set of *relations* $\{R_1^I, \ldots, R_m^I\}$, with each R_i^I a finite subset of $\mathsf{U}^{\text{arity}(R_i)}$. The *active domain of I*, denoted by $\text{adom}(I)$, is the set of elements of U which appear in some relation of I (we define the active domain of a relation analogously).

We assume familiarity with the syntax and semantics of first-order logic [3, 44]. Let $\mathbf{R}$ be a relational schema and I an instance over $\mathbf{R}$. A k-ary FO-query Q over $\mathbf{R}$ is given by an FO-formula $\varphi(\bar{x})$, where $\bar{x} = (x_1, \ldots, x_k)$ is the tuple of free variables of φ. Moreover, the evaluation of Q over I, denoted by $Q(I)$, is defined as the set of tuples $\bar{a}$ such that $\varphi(\bar{a})$ holds in I.

Example 2.1. *Consider the relational schema* $\mathbf{R} = \{\mathsf{Person}, \mathsf{Knows}\}$ *with arities* 1 *and* 2*, respectively. Also consider the relational instance* $I = \{\mathsf{Person}^I, \mathsf{Knows}^I\}$ *defined as follows:*

Person^I
`Ada`
`John`
`Dana`
`Peter`

Knows^I	
`Ada`	`John`
`John`	`Ada`
`Dana`	`Peter`

Then a query Q_1 given by FO-formula $\mathsf{Person}(x)$ *returns the set of persons in I, while a query Q_2 given by FO-formula* $\exists y\, (\mathsf{Person}(x) \wedge \mathsf{Knows}(x, y) \wedge x \neq y)$ *returns the set of persons in I that know someone else.* □

Given instances I_1, I_2 over a relational schema $\mathbf{R}$, a function $f : \mathsf{U} \to \mathsf{U}$ is an *isomorphism from I_1 to I_2* if and only if (i) f is a bijection, and (ii) for every $R \in \mathbf{R}$ such that $\text{arity}(R) = n$, and for every $t \in \mathsf{U}^n$, it is the case that $t \in R^{I_1}$ if and only if $f(t) \in R^{I_2}$, where $f(t)$ is defined as $(f(a_1), \ldots, f(a_n))$ if $t = (a_1, \ldots, a_n)$. Given an instance I over a relational schema $\mathbf{R}$, a function $f : \mathsf{U} \to \mathsf{U}$ is an *automorphism of I* if f is an isomorphism from I to I. The notions of isomorphism and automorphism for a relation are defined analogously. In what follows, we use $\textsc{Aut}(I)$, $\textsc{Aut}(R)$ to denote the set of automorphisms for an instance I and a relation R, respectively.

Given a tuple $t = (a_1, \ldots, a_n)$, define the i^{th} *prefix* of t in the following way:

$$\pi_{\leq i}(t) = \begin{cases} (a_1, \ldots, a_i) & \text{if } 1 \leq i \leq n, \\ () & \text{otherwise.} \end{cases}$$

In order to extract only one column, we use the notation $\pi_i(t) = a_i$. We also allow an overloaded version of the operator, where R is a relation of arity n:

$$\pi_{\leq i}(R) = \{\pi_{\leq i}(t) \mid t \in R\}.$$

In other words, $\pi_{\leq i}(R)$ is the image of every tuple $t \in R$ under $\pi_{\leq i}$.

The graph isomorphism problem and the complexity class GI The graph isomorphism problem is defined as GRAPH-ISO $= \{(G_1, G_2) \mid G_1$ and G_2 are isomorphic graphs$\}$ [16, 68]. A major open problem in computational complexity is to determine the exact complexity of this problem, in particular whether it can be solved in polynomial time, it is **NP**-complete, or it is **NP**-intermediate [68].

The problem GRAPH-ISO gives rise to the complexity class $\mathbf{GI} = \{L \mid L \leq_T^p$ GRAPH-ISO$\}$, where $L_1 \leq_T^p L_2$ indicates that there is a polynomial-time Turing reduction from the decision problem L_1 to the decision problem L_2. In other words, **GI** is the class of all problems L that can be solved in polynomial-time by an algorithm that uses an oracle (or subroutine) for the graph isomorphism problem [1]. Furthermore, a decision problem L is said to be **GI**-hard if and only if $L' \leq_T^p L$ for every problem $L' \in \mathbf{GI}$. Notice that this is a relaxation with respect to the traditional definition of hardness for **NP**, as only a Turing reduction is required (as opposed to a polynomial-time many-to-one reduction for the standard notion of hardness). Examples of **GI**-complete problems, that will be used in this chapter, are: the relational instance isomorphism problem, REL-ISO $= \{(I_1, I_2) \mid I_1$ and I_2 are isomorphic relational instances$\}$ [118], and the automorphism with one anti-fixed point problem, AUT-1-AFP $= \{(G, v) \mid G = (V, E)$ is a graph with $v \in V$ such that there exists an automorphism f of G for which $f(v) \neq v\}$ [72]. It is important to notice that although GRAPH-ISO $\in$ **NP**, the class **GI** is not known to be a subset of **NP**, since **GI** is defined in terms of polynomial-time Turing reductions and **NP** is not known to be closed under such reductions (**NP** is known to be closed under polynomial-time many-to-one reductions).

Finally, given a decision problem L, we denote the *complement* of L as $\overline{L}$.

2.3 Definability Problem for First-Order Logic

The first-order logic definability problem is defined as FO-DEF $= \{(I, R) \mid I$ is a relational instance, R is a relation, and there is a first-order query Q such that $Q(I) = R\}$. Notice that both the schema $\mathbf{R}$ of I and the arity n of R are not fixed but can be deduced from I and R, respectively, and also that the query does not mention any constants. This problem was studied in [83, 21], where it was determined that given a relational instance I and a relation R, $(I, R) \in$ FO-Def (that is, R is definable from I by a first-order query) if and only if (i) $\text{adom}(R) \subseteq \text{adom}(I)$ and (ii) $\text{Aut}(I) \subseteq \text{Aut}(R)$. We first build some intuition on this semantic characterisation.

For the "if" direction, assume that $Q(I) = R$, where Q is an FO-query. Then R cannot mention a value that does not occur in I, as first-order logic cannot invent new values. Thus, it should be the case that $\text{adom}(R) \subseteq \text{adom}(I)$. Moreover, assume that a and b are values occurring in I and h is an automorphism of I such that $h(a) = b$. We know that if we replace in I every value c by $h(c)$, then we obtain the same instance I. Hence, a and b are indistinguishable in I. In particular, these two values cannot be differentiated by Q, as Q is defined by an FO-formula whose vocabulary is $\mathbf{R}$ and, thus, by an FO-formula that does not mention any constant. Hence, given that $R = Q(I)$, if any of a or b occurs in R, then the other value has to occur in R, and, more generally, a and b have to be indistinguishable in R. Formally, h has to be an automorphism of R, and, therefore, every automorphism of I has to be an automorphism of R (that is, $\text{Aut}(I) \subseteq \text{Aut}(R)$). For the "only if" direction, we notice that it is possible to construct an FO-query Q which constructs all automorphisms of I and then uses projection to build the required tuples in R, the only restriction being that automorphisms will be preserved in any result. Thus, as long as conditions (i) and (ii) hold, the input will be definable. Interestingly, though, the witness Q to the definability is not of much practical use.

Example 2.2. *Let* $\mathbf{R}$ *and* I *be the relational schema and instance shown in Example 2.1, respectively, and assume that* R *and* S *are the following relations:*

R
John

S	
Ada	John
John	Ada

In this case, the pair $(I, S) \in$ FO-Def*, i.e. it is possible to find a first-order query* Q *such that* $Q(I) = S$*. In fact, in this case* Q *is given by FO-formula* $(\mathsf{Knows}(x, y) \wedge \mathtt{Knows}(y, x))$*. On the other hand, the relation* R *is not definable from* I *by a first-order query. To see why this is the case, notice that* Ada *and* John *are interchangeable in* I*, so that if* R *can be obtained as the result of evaluating an FO-query over* I*, then* Ada *has to occur in* R *as* John *occurs in* R*. We can formalise this intuition, and prove that* $(I, R) \notin$ FO-Def*, by using the semantic characterisation of the definability problem in terms of automorphisms. More precisely, consider the function* $h : \mathsf{U} \to \mathsf{U}$ *such* *that $h(\texttt{Ada}) = \texttt{John}$, $h(\texttt{John}) = \texttt{Ada}$, and for any other element $u \in \mathsf{U}$, $h(u) = u$. The function h thus defined is an automorphism of I. However, h is not an automorphism of R, as $h(\texttt{John}) \notin R$, from which we conclude that $(I, R) \notin$ FO-DEF.* □

2.4 The exact complexity of FO-DEF

From the characterisation of FO-DEF in the previous section, it is clear that FO-DEF $\in$ **coNP**, as an automorphism of I which is not an automorphism of R provides a (polynomially sized) witness to the fact that $(I, R) \notin$ FO-DEF. Although this provides an upper bound for the complexity of this problem, the exact complexity of FO-DEF is an open problem; in particular, the problem is not known to be **coNP**-hard. The following is the main result of this chapter, in which we close this problem:

Theorem 2.1. FO-DEF *is* **GI**-*complete.* □

This result allows us to revisit the status of the FO-DEF problem with respect to **PTIME** and **NP**. As a first corollary of Theorem 2.1, we can now say that if GRAPH-ISO $\in$ **PTIME** then it will also be the case that FO-DEF $\in$ **PTIME**. As second corollary of Theorem 2.1, we obtain strong evidence against the **coNP**-hardness of FO-DEF. Recall that if $\mathcal{C}$ is a complexity class, then $\mathbf{NP}^{\mathcal{C}}$ is the class of decision problems that can be solved in polynomial time by a non-deterministic Turing machine with an oracle for a decision problem $L \in \mathcal{C}$. Moreover, recall that the second level of the polynomial hierarchy [101] consists of the complexity classes $\Sigma_2^p = \mathbf{NP}^{\mathbf{NP}}$ and $\Pi_2^p = \mathbf{co}\Sigma_2^p = \{\overline{L} \mid L \in \Sigma_2^p\}$, which are widely believed to be different. From Theorem 2.1 we have that:

Corollary 1. *If* FO-DEF *is* **coNP**-*complete, then* $\Sigma_2^p = \Pi_2^p$.

To understand this corollary, we must consider the second level of the *low hierarchy* of **NP** [94, 61]. Let $\mathbf{Low}_2$ be the class of decision languages $L \in \mathbf{NP}$ such that:

$$\mathbf{NP}^{(\mathbf{NP}^L)} = \mathbf{NP}^{\mathbf{NP}},$$

that is, the class of languages $L \in \mathbf{NP}$ such that the computational power of the second level of the polynomial hierarchy is not augmented if L is available as an oracle. It is known that GRAPH-ISO $\in \mathbf{Low}_2$ [95], and also that if a language in $\mathbf{Low}_2$ is **NP**-complete (under the usual notion of polynomial-time many-to-one reduction), then $\Sigma_2^p = \Pi_2^p$ [94]. From Theorem 2.1 and the fact that GRAPH-ISO $\in$ **NP**, we have that $\overline{\text{FO-DEF}} \in \mathbf{NP} \cap \mathbf{GI}$, from which we conclude that $\overline{\text{FO-DEF}} \in \mathbf{Low}_2$. Hence, if $\overline{\text{FO-DEF}}$ is **NP**-complete, then $\Sigma_2^p = \Pi_2^p$, from which Corollary 1 follows.

As a final comment, it is important to mention that Theorem 2.1 holds for any relational query language that is BP-complete [33, 58]. Thus, for example, the definability problem for Datalog is also **GI**-complete, where the input of this problem is an instance I and a relation R, and the question to answer is whether there exists a Datalog program that evaluated over I produces R.

In the rest of this section, we concentrate in proving Theorem 2.1.

2.4.1 Proof of Theorem 2.1

The **GI**-hardness of FO-DEF is shown via a polynomial-time Turing reduction from AUT-1-AFP, whereas the inclusion of FO-DEF in **GI** is shown via a polynomial-time Turing reduction to REL-ISO. Recall that these problems were defined in Section 2.2, and that they are both known to be **GI**-complete [118, 72].

In order to motivate the proof of the inclusion of FO-DEF in **GI**, consider the following algorithm for FO-DEF using an oracle for REL-ISO, which constitutes a failed attempt to show that FO-DEF $\leq_T^p$ REL-ISO. On input (I, R), with $\text{arity}(R) = n$, we wish show the existence of a function $f \in \text{AUT}(I)$ for which there is a tuple $t \in R$ such that $f(t) = s$ and $s \notin R$ (note that $s \in \text{adom}(I)^n$).[1] This scenario can be restated in the following way: does there exist a tuple $t \in R$, a *bad* tuple $s \in \text{adom}(I)^n \smallsetminus R$, and an automorphism $f \in \text{AUT}(I)$ such that $f(t) = s$? Then for every $t = (a_1, \ldots, a_n) \in R$ and $s = (b_1, \ldots, b_n) \in \text{adom}(I)^n \smallsetminus R$, the procedure builds the relational instances I_1 and I_2 from I by *marking*, for every $i \in [1, n]$, the pair a_i, b_i in such a way that any isomorphism from I_1 to I_2 must map a_i to b_i (these markings can be achieved by placing a_i and b_i in fresh unary relations). We then consult the REL-ISO oracle with input (I_1, I_2) to decide whether such an isomorphism exists.

Example 2.3. *Consider the pair (I, R) from Examples 2.1 and 2.2. In order to decide whether (I, R) is definable, we iterate over every tuple in R; the only such tuple being $t = (\texttt{John})$. For this fixed t, we iterate over all possible bad tuples $s \in \text{adom}(I) \smallsetminus R$. Upon reaching the case $s = (\texttt{Ada})$ we build the following instances:*

- *We first create a fresh relation name* Fresh *such that* $\text{arity}(\mathsf{Fresh}) = \text{arity}(R) = 1$,

[1]This would actually shows that (I, R) is *not definable*, but as this is a deterministic algorithm, we may simply return the opposite answer.

- *We prepare an instance I_1 over the relational schema* $\{\mathsf{Person}, \mathsf{Knows}, \mathsf{Fresh}\}$ *such that* $\mathsf{Person}^{I_1} = \mathsf{Person}^{I}$, $\mathsf{Knows}^{I_1} = \mathsf{Knows}^{I}$, *and* $\mathsf{Fresh}^{I_1} = \{(\texttt{John})\}$
- *We prepare an instance I_2 over the relational schema* $\{\mathsf{Person}, \mathsf{Knows}, \mathsf{Fresh}\}$ *such that* $\mathsf{Person}^{I_2} = \mathsf{Person}^{I}$, $\mathsf{Knows}^{I_2} = \mathsf{Knows}^{I}$, *and* $\mathsf{Fresh}^{I_2} = \{(\texttt{Ada})\}$.

We now call a REL-ISO *oracle with input (I_1, I_2) in order to decide whether there is an isomorphism from I_1 to I_2. Note that, with the addition of the* Fresh^{I} *relation, we are actually asking whether there is an automorphism of I which maps* `John` *to* `Ada`*. The oracle will respond* True*, and this will serve as a witness to the non-definability of (I, R), whereby we return* False*.* □

The previous algorithm does not constitute a proof that FO-DEF $\leq_T^p$ REL-ISO due to the fact that there are exponentially many bad tuples $s \in \mathrm{adom}(I)^n \smallsetminus R$ to be checked. This problem can be avoided by considering an *incremental* characterisation of the first-order definability problem, which we turn to now.

Lemma 2.1. *Let I be an instance of a relational schema $\mathbf{R}$ and R a relation of arity n such that $\mathrm{adom}(R) \subseteq \mathrm{adom}(I)$. Given $f \in \mathrm{AUT}(I)$, f is not an automorphism of R if and only if there exists a tuple $t \in R$ and an integer $i \in [0, n-1]$ such that:*

1. $f(\pi_{\leq i}(t)) \in \pi_{\leq i}(R)$,
2. $f(\pi_{\leq i+1}(t)) \notin \pi_{\leq i+1}(R)$. □

Intuitively, f is not an automorphism of R if there is a tuple $t \in R$ for which $f(t) \notin R$; however, we can refine this notion by finding the *column* i such that f maps t correctly in the i^{th} prefix (condition (1)), but fails to map t correctly for the $(i+1)^{\text{th}}$ prefix (condition (2)).

Proof of Lemma 2.1. Let I be an instance of a relational schema $\mathbf{R}$ and R a relation of arity n such that $\mathrm{adom}(R) \subseteq \mathrm{adom}(I)$. The following is a well-known characterisation of the notion of automorphism of a relation.

Claim 2.1. *$f \in \mathrm{AUT}(I)$ is not an automorphism of R if and only if there exists a tuple $t \in R$ such that $f(t) \notin R$.* □

To prove the direction ($\Rightarrow$) of the lemma, we assume that f is not an automorphism of R. In that case, we know by Claim 2.1 that there is a tuple $t_0 \in R$ such that

$f(t_0) \neq t$ for every $t \in R$. Then for every $t \in R$ define k_t as the minimum element of the set:

$$\{i \in [1, n] \mid f(\pi_i(t_0)) \neq \pi_i(t)\}.$$

That is, k_t represents the left-most column for which f fails to map t_0 to t. With the previous, define:

$$i_0 = \left(\max_{t \in R} k_t\right) - 1.$$

Then we have that i_0 satisfies the conditions stated in the lemma:

1. Let $t' = \text{argmax}_{t \in R}\, k_t$. Then, by definition of i_0 and t', we have that $f(\pi_{\leq i_0}(t_0)) = \pi_{\leq i_0}(t')$ (whereby $f(\pi_{\leq i_0}(t_0)) \in \pi_{\leq i_0}(R)$),

2. Let $t'' \in R$. Then, by definition of i_0, we have that $f(\pi_{\leq i_0+1}(t_0)) \neq \pi_{\leq i_0+1}(t'')$. As t'' is arbitrary, this implies that $f(\pi_{\leq i_0+1}(t)) \notin \pi_{\leq i_0+1}(R)$.

For the direction ($\Leftarrow$), assume there exists tuple $t \in R$ and integer $i \in [0, n-1]$ such that items (1) and (2) hold. In particular, item (2) implies that $f(t) \notin R$, whereby f is not an automorphism of R by Claim 2.1. $\square$

We finally have all the necessary ingredients to prove Theorem 2.1.

Proof of Theorem 2.1. We first show that FO-Def $\in$ **GI** by determining that the following relation holds: FO-Def $\leq_T^p$ Graph-Iso. We use the result of Lemma 2.1 to produce Algorithm 1, a deterministic polynomial-time algorithm which uses an oracle for the Rel-Iso decision problem.

Let I be a relational instance and R a relation such that $\text{arity}(R) = n$, and assume that $\text{adom}(R) \subseteq \text{adom}(I)$. On input (I, R), Algorithm 1 proceeds in a similar way as the naïve algorithm described at the beginning of this section, but trying to show the existence of a function $f \in \text{Aut}(I)$ which fails as an automorphism of R in a specific column of R. More precisely, Algorithm 1 starts by picking the values of i and t in its first two loops, which will be used as stated in Lemma 2.1 to show that a function $f \in \text{Aut}(I)$ is not an automorphism of R. As $f(\pi_{\leq i}(t))$ must be a tuple in $\pi_{\leq i}(R)$ according to Lemma 2.1, there must exist a tuple $s \in R$ such that $f(\pi_{\leq i}(t)) = \pi_{\leq i}(s)$. This tuple is chosen in the third loop of the algorithm. Besides, given that $f(\pi_{\leq i+1}(t)) \notin \pi_{\leq i+1}(R)$ according to Lemma 2.1, then it must be the case that $f(\pi_{i+1}(t)) \neq \pi_{i+1}(s)$. But in fact, for every tuple $r \in R$ such that $\pi_{\leq i}(r) = \pi_{\leq i}(s)$, it must be the case that $f(\pi_{i+1}(t)) \neq \pi_{i+1}(r)$, otherwise $f(\pi_{\leq i+1}(t))$ would be a tuple in $\pi_{\leq i+1}(R)$. The set BadElements contains all the possible values a for $f(\pi_{i+1}(t))$ that make $f(\pi_{i+1}(t))$ to satisfy this condition. Thus, in its innermost loop, Algorithm

ALGORITHM 1: Algorithm for deciding first-order logic definability.

```
Input: Relational instance $I$, relation $R$ with $\mathrm{arity}(R) = n$.
Output: True if $\mathrm{adom}(R) \subseteq \mathrm{adom}(I)$ and every automorphism of $I$ is also an
        automorphism of $R$, and False otherwise.
1  if $\mathrm{adom}(R) \not\subseteq \mathrm{adom}(I)$ then
2      return False
3  end
4  for $i = 0$ to $n - 1$ do
5      foreach $t \in R$ do
6          foreach $s \in R$ do
7              BadElements $\leftarrow \{a \in \mathrm{adom}(I) \mid \forall r \in R :$ if $\pi_{\leq i}(r) = \pi_{\leq i}(s)$, then $\pi_{i+1}(r) \neq$
                $a\}$;
8              foreach $a \in$ BadElements do
9                  if CheckForIso($I, t, s, i, a$) then
10                     return False
11                 end
12             end
13         end
14     end
15 end
16 return True

17 Function CheckForIso($I, t, s, i, a$)
       Input: Relational instance $I$, $n$-ary tuples $t$ and $s$, values $i \in [0, n]$ and $a \in \mathrm{adom}(I)$.
       Output: True if there exists an automorphism $f$ of $I$ such that $f(\pi_{\leq i}(t)) = \pi_{\leq i}(s)$ and
               $f(\pi_{i+1}(t)) = a$, and False otherwise.
18     $\mathbf{R} \leftarrow$ Relational schema of $I$;
19     $\mathbf{R}^{\star} \leftarrow \mathbf{R} \cup \{R_1, \ldots, R_i, R_a\}$, where each $R_j$ $(1 \leq j \leq i)$ and $R_a$ are fresh unary relation
        names;
20     $I_1 \leftarrow$ empty instance of $\mathbf{R}^{\star}$;
21     $I_2 \leftarrow$ empty instance of $\mathbf{R}^{\star}$;
22     foreach $R \in \mathbf{R}$ do
23         $R^{I_1} \leftarrow R^I$;
24         $R^{I_2} \leftarrow R^I$;
25     end
26     for $j = 1$ to $i$ do
27         $R_j^{I_1} \leftarrow \{(\pi_j(t))\}$;
28         $R_j^{I_2} \leftarrow \{(\pi_j(s))\}$;
29     end
30     $R_a^{I_1} \leftarrow \{(\pi_{i+1}(t))\}$;
31     $R_a^{I_2} \leftarrow \{(a)\}$;
32     if there exists an isomorphism from $I_1$ to $I_2$ (that is, $(I_1, I_2) \in$ Rel-Iso) then return
        True;
33     else return False ;
```

1 picks a value $a \in \textsc{BadElements}$, and makes the call `CheckForIso(`$I, t, s, i, a$`)` to check whether there exists an automorphism f of I such that $f(\pi_{\leq i}(t)) = \pi_{\leq i}(s)$ and $f(\pi_{i+1}(t)) = a$. If this is the case, then Algorithm 1 knows that $f \in \textsc{Aut}(I)$ and f is not an automorphism of R, so it returns False. Otherwise, after trying all possibilities for i, t, s and a, Algorithm 1 knows by Lemma 2.1 that every automorphism of I is an automorphism of R, so it returns True.

To check whether there exists an automorphism f of I such that $f(\pi_{\leq i}(t)) = \pi_{\leq i}(s)$ and $f(\pi_{i+1}(t)) = a$, function `CheckForIso` generalises the approach given in Example 2.3, and uses an oracle for the Rel-Iso decision problem (in its penultimate line). More precisely, this function starts by creating two copies I_1 and I_2 of I. Then it adds to I_1 the fresh facts $R_1(\pi_1(t))$, …, $R_i(\pi_i(t))$, $R_a(\pi_{i+1}(t))$, and it adds to I_2 the fresh facts $R_1(\pi_1(s))$, …, $R_i(\pi_i(s))$, $R_a(a)$. Finally, it calls the oracle to verify whether there exists an isomorphism from I_1 to I_2, which represents an automorphism of I satisfying the aforementioned conditions, as it has to map $\pi_j(t)$ to $\pi_j(s)$ $(1 \leq j \leq i)$ and $\pi_{i+1}(t)$ to a.

Example 2.4. *Continuing with Examples 2.1 and 2.2, now consider the definability problem for the pair (I, T), where I is defined as in Example 2.1 and T is the following relation:*

T		
John	Dana	John
Ada	Dana	John

In this case, for $i = 0$ the algorithm will not find any automorphism of I which fails to be an automorphism of $\pi_1(T)$. In fact, the only non-trivial automorphism of I is the one that maps John $\rightarrow$ Ada, Ada $\rightarrow$ John, Dana $\rightarrow$ Dana *and* Peter $\rightarrow$ Peter, *and this one maps T correctly up to column $i = 1$. Similarly, for $i = 1$ we have that the only non-trivial automorphism of I is also an automorphism of $\pi_{\leq 2}(T)$, so again the algorithm will not find the witness automorphism. For value $i = 2$, consider the iteration step at which $t = (\texttt{John}, \texttt{Dana}, \texttt{John})$ and $s = (\texttt{Ada}, \texttt{Dana}, \texttt{John})$. Then we have that:*

$$\textsc{BadElements} = \{\texttt{Ada}, \texttt{Dana}, \texttt{Peter}\}.$$

We now iterate over the elements of $\textsc{BadElements}$. *For $a =$ Ada we build instances I_1 and I_2 as follows. For I_1 we have that* $\mathsf{Person}^{I_1} = \mathsf{Person}^{I}$ *and* $\mathsf{Knows}^{I_1} = \mathsf{Knows}^{I}$, *and we add the fresh facts:*

$T_1^{I_1}$
John *$(= \pi_1(t))$*

$T_2^{I_1}$
Dana *$(= \pi_2(t))$*

$T_a^{I_1}$
John *$(= \pi_3(t))$*

For I_2 we have that $\mathsf{Person}^{I_2} = \mathsf{Person}^{I}$ *and* $\mathsf{Knows}^{I_2} = \mathsf{Knows}^{I}$*, and:*

$T_1^{I_2}$
`Ada` *(=* $\pi_1(s)$*)*

$T_2^{I_2}$
`Dana` *(=* $\pi_2(s)$*)*

$T_a^{I_1}$
`Ada` *(=* a*)*

Then we have that I_1 and I_2 are in fact isomorphic, whereby the Rel-Iso *will return* True*. Therefore, as a witness has been found, the algorithms returns* False*.* □

Algorithm 1 runs in time polynomial in the size of the input, assuming that every call to the subroutine for the Rel-Iso decision problem takes constant time (that is, assuming that Algorithm 1 has access to an oracle for the Rel-Iso decision problem). More precisely, let $|S|$ be the number of elements in a set S, and recall that $n = \text{arity}(R)$. Then the outer loops of Algorithm 1 complete at most $|R|^2 \times n$ iterations; for each of these iterations the set BadElements is computed in polynomial time, as at most $|\operatorname{adom}(I)|$ candidate elements a are tested, where for each element a the condition defining the set BadElements can be checked in polynomial-time on $|R|$, $i \leq n$ and $|\operatorname{adom}(I)|$. Moreover, as to the subroutine CheckForIso, it builds the relational instances I_1 and I_2 in polynomial time as well.

From the discussion in the previous paragraph, the fact that Rel-Iso $\in$ **GI** and the transitivity of polynomial-time Turing reductions, we conclude that FO-Def $\leq_T^p$ Graph-Iso, whereby FO-Def $\in$ **GI**.

We will now show that FO-Def is **GI**-hard by showing that Aut-1-AFP $\leq_T^p$ FO-Def (we actually show a many-to-one reduction to the complement of FO-Def, which is a stronger result than we need). Given a graph $G = (V, E)$ and a node $v \in V$, build a relational instance I with only one relation E copying the edge relation of G. Finally, build the relation R in the following way: $R^I = \{(v)\}$, that is, R has arity 1 and only contains one tuple with the distinguished node v. Note that, as built, an automorphism f of I which is not an automorphism of R will be such that $f(v) \neq v$. Thus, we have that $(G, v) \in$ Aut-1-AFP if and only if $(I, R) \notin$ FO-Def. Hence, given that (I, R) can be constructed in polynomial-time from (G, v), we conclude that the problem Aut-1-AFP can be solved in polynomial-time by using an oracle for FO-Def.

We therefore conclude that Aut-1-AFP $\leq_T^p$ FO-Def, whereby FO-Def is **GI**-complete. □

2.5 Practical Considerations and Extensions

Having established the exact complexity of the first-order logic definability problem, we now turn to possible variations of the problem and practical considerations. The definability problem, while of great theoretical interest, should be considered in the broader context of database research. As was mentioned in the introduction, the definability problem provides a common basis for research in reverse engineering [109, 119], querying by example [4, 28], view definitions [92], etc. In fact, FO-DEF may be interpreted as a basic query reverse-engineering scenario, where a user who has access to a dataset and an answer relation needs to discover the query (first-order query, in this case) which produced such an answer over the data. A natural extension of this scenario is that in which we must fit several such example source-target pairs [47], which we discuss in Section 2.5.1. Such scenarios find applications in areas such as schema matching and data integration [23, 53, 88, 106]. In each of these areas it may be interesting to explore the consequences of the graph-isomorphism based approach to the definability problems presented here.

In terms of practical implementations of FO-DEF itself, an algorithm for FO-DEF whose efficiency depends on an external subroutine for the graph isomorphism problem —a heavily studied problem in its own right—comes with several benefits for optimisation. Not only can we now tap into the power of highly optimised graph-isomorphism [89, 17, 107, 68, 87, 76, 20], we can also consider all restrictions on the input graphs that produce efficiently solvable versions of the graph isomorphism problem, and inherit those benefits in our FO-DEF implementations (for example, see [54, 55]).

As a final consideration, in Section 2.5.2 we comment on the use of constants in the queries. Although we will see that unrestricted constants results in an uninteresting problem, a more restricted use of constants may have practical applications that make this case worth looking into.

2.5.1 The BP-PAIRS problem

Expanding on the definability problem as a reverse engineering scenario, where a query must be obtained to match a source-target (relational instance-relation) pair, the situation where several such pairs are given is represented by the following decision problem:

$$\begin{aligned}\text{BP-PAIRS} \;=\; \{((S_1, T_1), \ldots, (S_k, T_k)) \mid\ & S_1, T_1, \ldots S_k, T_k \text{ are relations and} \\ & \text{there exists a first-order query } Q \text{ such that for every } i \in [1, k] : Q(S_i) = T_i\}.\end{aligned}$$

In [47], it was shown that $\overline{\text{GRAPH-ISO}} \leq^p_m$ BP-PAIRS, that is, there exists a polynomial-time many-to-one reduction from $\overline{\text{GRAPH-ISO}}$ to BP-PAIRS (this was referred to as *cograph-isomorphism-hardness* in [47]). Moreover, it was also shown in [47] that BP-Pairs $\in$ **coNP**. A corollary of the first result is that BP-Pairs is **GI**-hard, as a many-to-one reduction also constitutes a Turing reduction (the **GI**-hardness of BP-Pairs can be alternatively derived using the results from Section 2.4). The key insight regarding this generalised version of the definability problem is its semantic characterisation: an input $((S_1, T_1), \dots, (S_k, T_k))$ is in BP-Pairs if and only if (i) for every $i \in [1, k]$, we have $\mathrm{adom}(T_i) \subseteq \mathrm{adom}(S_i)$, and (ii) for every $i, j \in [1, k]$, we have that if f is an isomorphism from S_i to S_j, then it is also an isomorphism from T_i to T_j [47].

Algorithm 1 can be adapted to solve this decision problem as well, leading to the following:

Theorem 2.2. BP-Pairs $\in$ **GI**. □

The previous result, along with the **GI**-hardness of BP-Pairs, as proven in [47], gives the following result:

Corollary 2. BP-Pairs *is* **GI**-*complete.* □

The previous corollary establishes the exact complexity of BP-Pairs and, thus, closes a problem that was left open in [47].

In order to prove Theorem 2.2, consider the following extension of Lemma 2.1:

Lemma 2.2. *Let S_1, S_2, T_1, T_2 be relations such that $\mathrm{adom}(T_i) \subseteq \mathrm{adom}(S_i)$ for $i \in [1, 2]$. Given an isomorphism f from S_1 to S_2, f is not an isomorphism from T_1 to T_2 if and only if there exists a tuple $t \in T_1$ and an integer $i \in [0, n-1]$ such that:*

1. $f(\pi_{\leq i}(t)) \in \pi_{\leq i}(T_2)$,
2. $f(\pi_{\leq i+1}(t)) \notin \pi_{\leq i+1}(T_2)$. □

With this result an algorithm analogous to that shown in Section 2.4.1 is used to prove Theorem 2.2.

2.5.2 Including constants in the definability problem

As was mentioned previously, FO-Def considers the existence of a first-order logic query without constants. Let FO-Def-Const the decision problem consisting of pairs (I, R) such that there exists a first-order query *with constants* Q such that $Q(I) = R$. Then FO-Def-Const can be decided in polynomial time due to the fact that a pair (I, R) will be included in FO-DEF-CONST if and only if $\text{adom}(R) \subseteq \text{adom}(I)$. It is evident that this problem has become uninteresting, as a query can always be found, with the sole exception that a first-order query may not introduce new constants into the answer. The actual reverse engineered query Q such that $Q(I) = R$ is not very informative though; given an input (I, R) such that $\text{adom}(R) \subseteq \text{adom}(I)$ and $\text{arity}(R) = n$, the proof of FO-DEF-CONST $\in$ **PTIME** constructs a query Q of the form $\{(x_1, \ldots, x_n) \mid \bigvee_{t \in R} Q_t(x_1, \ldots, x_n)\}$, where, for a tuple $t = (a_1, \ldots, a_n)$ in R, the query $Q_t(x_1, \ldots, x_n)$ is the expression $\bigwedge_{i \in [1,n]} x_i = a_i$. This query is fine tuned to the specific pair (I, R) and does not shed light on the original unknown query which might have produced this pair. In fact, this query becomes useless if some constants in the input (I, R) are renamed and, thus, it is an example of over-fitting.

A more restricted, and useful, use of constants is formalised in the following problem: FO-DEF-CONST-S $= \{(I, R, C)\} \mid I$ is a relational instance, R is a relation, C is a set of constants, and there exists a first-order query Q, which may mention constants in C only, such that $Q(I) = R\}$. This extension of FO-DEF is **GI**-complete. To see this, note that FO-DEF $\leq_m^p$ FO-DEF-CONST-S is trivial (by setting $C = \emptyset$) and that FO-DEF-CONST-S $\leq_m^p$ FO-DEF admits a simple proof as well. On input (I, R, C) to FO-DEF-CONST-S, construct an instance (I', R') to FO-DEF by encoding the constants in C into the instance I', using singleton relations. More precisely, set $R' = R$ and let I' have all the relations in I plus a unary singleton relation $C_i = \{(c_i)\}$ for each constant $c_i \in C$. The previous arrangement for (I', R') allows constants to be referred to indirectly by using the expression $C_i(x)$ in a first-order query, as it will only be true when x is assigned to c_i.

Consider the problem FO-DEF-CONST-$\leq$ $= \{(I, R, 0^n) \mid I$ is a relational instance, R is a relation, and n is a natural number, such that there exists a first-order query Q which mentions at most n distinct constants, and $Q(I) = R\}$ as a more elaborate, and also interesting, setting. Notice that the input n in FO-DEF-CONST-$\leq$ is encoded in unary as a string of 0's of length n. Although FO-DEF-CONST-$\leq$ remains **GI**-hard (set $n = 0$ in a Turing reduction), it is no longer obviously in **GI**. Actually, FO-DEF-CONST-$\leq$ $\in \mathbf{NP}^{\mathbf{GI}}$, as a non-deterministic polynomial-time algorithm may guess a set C of n constants and use an oracle for the FO-DEF-CONST-S problem. The question remains, then, whether FO-DEF-CONST-$\leq$ is in **GI**.

Finally, consider the case where the queries have access to a linear order over the constants in the relational instance, which exhibits underlying similarities to the unrestricted constants case FO-DEF-CONST. Formally, consider the problem

FO-DEF-LIN $= \{(I, R) \mid I$ is a relational instance having a binary relation $<$ which is a linear order over all elements in $\text{adom}(I)$, R is a relation, and there exists a first-order query Q such that $Q(I) = R\}$. In the presence of the linear order, and using the semantic characterisation, the only automorphism of I is the identity (i.e. the function $h(x) = x$), which is trivially also an automorphism of R. Hence, in this case an algorithm must only ensure that $\text{adom}(R) \subseteq \text{adom}(I)$ to check whether $(I, R) \in$ FO-DEF-LIN, which may be completed in polynomial time. Therefore, FO-DEF-LIN $\in$ **PTIME**, and once again the problem becomes trivial. Moreover, this is also an example of over-fitting, as every element in I can be identified by its position in the linear order, which is used as in the case of FO-DEF-CONST to define a query Q such that $Q(I) = R$.

Chapter 3

Reverse Engineering SPARQL Queries

3.1 Introduction

The data model used in Semantic Web systems views data as collections of RDF triples, a fairly low-level representation, but the Web APIs expose SPARQL endpoints, a high level declarative query language which allows users to pose queries that combine and filter information. Declarative query languages are powerful, but they are known to have disadvantages in terms of ease of use by wider audiences. The alternative query-by-example paradigm, where users present examples of what they want, and the system generalises them [109, 29, 110], is particularly attractive in an open data setting, since to harness the power of these interfaces users must understand the structure of data as well as the features of SPARQL needed to express their information needs, and this is frequently not the case. Even users familiar with SPARQL and with a given dataset may prefer to explore the data via example and have the system suggest generalisations.

In this chapter we study of the problem of querying via examples for SPARQL. We formalise the problem as "reverse engineering SPARQL queries from examples", following a line of research that has been developed for other problems dealing with learning from examples [52], including regular languages [6, 7, 8], relational database queries [119], XML queries [99, 36], and queries in graph databases [26]. A common baseline for these approaches lies in the definability problem [83, 21, 115, 9], which was discussed in the previous chapter.

We present several variations of the problem, depending on whether the user presents a set of positive examples, a set of positive and negative examples, or an exact answer set. We also vary the subset of SPARQL that the system is permitted to synthesise, starting with simple graph patterns, appending the SPARQL operator for obtaining optional information, and finally considering an extension with SPARQL's feature for filtering results according to some conditions.

Our first contribution is theoretical: we study the complexity of all variants of the reverse engineering problem. We provide tight complexity bounds for the reverse engineering problem with positive examples, with positive and negative examples, and with an exact answer, looking at reverse engineering queries in fragments of SPARQL ranging from very expressive (allowing all the operators mentioned above) to very limited (allowing only conjunction).

Having completed the picture of the theory of reverse engineering, we turn to a practical implementation of it. We provide a parametrised algorithm for reverse engineering from positive examples, where the parameters allow tuning several features of the target class. We evaluate our algorithms on real-world and synthetic queries, showing that it scales well both with the input size and the complexity of a target query needed to match a set of examples, that it can reverse engineer complex queries with high accuracy, and that it can be useful as a supplement to an existing SPARQL engine. Portions of this chapter were published in [11], not including full proofs and derivations.

3.2 Preliminaries

3.2.1 RDF and the query language SPARQL

The RDF (Resource Description Framework) data model is used to represent information about World Wide Web resources, and was released as a W3C (World Wide Web Consortium) recommendation in 2004 [56]. Along with RDF, the W3C defined the query language SPARQL as a recommendation for querying RDF data. We shall present only a simplified version of the full definitions, in line with the formalisation given by [84].

Assume two countably infinite disjoint sets U and L of *URIs* and *literals*, respectively. An *(RDF) triple* is a tuple $(s, p, o) \in \mathsf{U} \times \mathsf{U} \times (\mathsf{U} \cup \mathsf{L})$, and an *(RDF) graph* is a finite set of RDF triples. Define another countably infinite set V of *variables*, disjoint from U and L.

Next we define the fragment of the SPARQL query language that will be considered in this chapter. We start by introducing the notion of (SPARQL) *built-in condition*, defined inductively as follows:

- if $\mathbf{v_1}, \mathbf{v_2} \in \mathsf{V}$ and $a \in (\mathsf{U} \cup \mathsf{L})$, then $\mathbf{v_1} = a$, $\mathbf{v_1} = \mathbf{v_2}$, and $\mathsf{bound}(\mathbf{v_1})$ are built-in conditions,
- if R_1 and R_2 are built-in conditions, then $(\neg R_1)$, $(R_1 \vee R_2)$, and $(R_1 \wedge R_2)$ are built-in conditions.

The notion of (SPARQL) *graph pattern* is inductively defined next:

- a triple from $(\mathsf{U} \cup \mathsf{V}) \times (\mathsf{U} \cup \mathsf{V}) \times (\mathsf{U} \cup \mathsf{V} \cup \mathsf{L})$ is a graph pattern (called *triple pattern*),
- if P_1 and P_2 are graph patterns, then $(P_1 \,\textbf{AND}\, P_2)$ and $(P_1 \,\textbf{OPT}\, P_2)$ are graph patterns,
- if P is a graph pattern and R is a built-in condition, then $(P \,\textbf{FILTER}\, R)$ is a graph pattern.

Example 3.1. *Assume that* $?\mathtt{X} \in \mathsf{V}$*,* $\{\mathtt{type}, \mathtt{Person}, \mathtt{age}\} \subseteq \mathsf{U}$*, and* $\mathtt{32} \in \mathsf{L}$*. Then, we have that* $P_1 = (?\mathtt{X}, \mathtt{type}, \mathtt{Person})$ *and* $P_2 = (?\mathtt{X}, \mathtt{type}, \mathtt{Person}) \textbf{ AND } (?\mathtt{X}, \mathtt{age}, 32)$ *are graph patterns, which intuitively ask for the list of all people (more precisely, all elements of type person) and for the list of people whose age is 32, respectively. Moreover, the following is also a graph pattern:* $P_3 = [(?\mathtt{X}, \mathtt{type}, \mathtt{Person}) \textbf{ AND } (?\mathtt{X}, \mathtt{age}, 32)]$ $\textbf{OPT } (?\mathtt{X}, \mathtt{email}, ?\mathtt{Y})$*, where the* **OPT** *operator is used to retrieve the email of each person stored in the variable* $?\mathtt{X}$ *if this information is available.*

The SPARQL query language includes also union, projection and some additional built-in predicates; these features of SPARQL are not considered, and are left for future work. To distinguish the entire SPARQL query language from the fragment considered in this chapter, the latter is denoted by SP[AOF], where A, O and F stand for the operators **AND**, **OPT** and **FILTER**, respectively.

To define the semantics of graph patterns, we define a *mapping* μ as a partial function from V to $\mathsf{U} \cup \mathsf{L}$ with finite domain, which is denoted by $\mathrm{dom}(\mu)$. Two mappings μ_1 and μ_2 are *compatible*, denoted $\mu_1 \sim \mu_2$, if $\mu_1(\mathbf{v}) = \mu_2(\mathbf{v})$ for all $\mathbf{v} \in \mathrm{dom}(\mu_1) \cap \mathrm{dom}(\mu_2)$ (i.e. when $\mu_1 \cup \mu_2$ is also a mapping). Given sets Ω_1 and Ω_2 of mappings, define the following operations on them:

$$\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ and } \mu_1 \sim \mu_2\},$$
$$\Omega_1 - \Omega_2 = \{\mu \mid \mu \in \Omega_1 \text{ and } \forall \mu_2 \in \Omega_2 : \mu_1 \not\sim \mu_2\}.$$

We start by defining the semantics of built-in conditions. Given a built-in condition R and a mapping μ, we say that μ satisfies R, denoted by $\mu \models R$, if

- R is $\mathbf{v} = a$ for some $\mathbf{v} \in \mathsf{V}$, $a \in \mathsf{U} \cup \mathsf{L}$, and $\mu(\mathbf{v}) = a$,
- R is $\mathbf{v_1} = \mathbf{v_2}$ for some $\mathbf{v_1}, \mathbf{v_2} \in \mathsf{V}$ and $\mu(\mathbf{v_1}) = \mu(\mathbf{v_2})$,
- R is $\mathsf{bound}(\mathbf{v})$ for some $\mathbf{v} \in \mathsf{V} \cap \mathrm{dom}(\mu)$,
- R is $(\neg R_1)$ for a built-in condition R_1 and $\mu \not\models R_1$,
- R is $(R_1 \vee R_2)$ for some R_1, R_2 and $\mu \models R_1$ or $\mu \models R_2$ (or both),
- R is $(R_1 \wedge R_2)$ for R_1, R_2, and both $\mu \models R_1$ and $\mu \models R_2$.

We now move to the definition of the semantics of graph patterns. Given a graph pattern P, we define $\mathrm{var}(P)$ to be the set of variables which occur in P, noting that $\mathrm{var}(P) \subseteq \mathsf{V}$, and analogously for a built-in condition R. Given a triple pattern t and a mapping μ such that $\mathrm{var}(t) \subseteq \mathrm{dom}(\mu)$, we define $\mu(t)$ as the RDF triple obtained from t by replacing every variable $\mathbf{v}$ occurring in t by $\mu(\mathbf{v})$. Then given a graph pattern P and an RDF graph D, the evaluation of P over D, denoted by $[\![P]\!]_D$, is the set of mappings defined recursively as follows:

- If P is a triple pattern t, then $[\![P]\!]_D = \{\mu \mid \mathrm{var}(t) = \mathrm{dom}(\mu) \text{ and } \mu(t) \in D\}$,
- If $P = (P_1 \,\mathbf{AND}\, P_2)$ for graph patterns P_1 and P_2, then $[\![P]\!]_D = [\![P_1]\!]_D \bowtie [\![P_2]\!]_D$,
- If $P = (P_1 \,\mathbf{OPT}\, P_2)$ for graph patterns P_1 and P_2, then $[\![P]\!]_D = ([\![P_1]\!]_D \bowtie [\![P_2]\!]_D) \cup ([\![P_1]\!]_D - [\![P_2]\!]_D)$,
- If $P = (P_1 \,\mathbf{FILTER}\, R)$ for a graph pattern P_1 and a built-in condition R, then $[\![P]\!]_D = \{\mu \in [\![P_1]\!]_D \mid \mu \models R\}$.

Note that the official W3C semantics for SPARQL 1.1 defines **OPT** via a ternary 'LeftJoin' operator that includes a **FILTER** expression. However, in the well-designed fragments of SPARQL considered in this work, the difference is of no consequence. Also noteworthy is the fact that SPARQL includes many more features that have not been considered in this study, including projection, union, negation, path expressions, among others. These limitations restrict the possible patterns in the data that may be effectively reverse engineered into a query.

From now on, we use notation $[?\mathtt{X}_1 \mapsto a_1, \ldots, ?\mathtt{X}_k \mapsto a_k]$ to represent a mapping μ such that $\mathrm{dom}(\mu) = \{?\mathtt{X}_1, \ldots, ?\mathtt{X}_k\}$ and $\mu(?\mathtt{X}_i) = a_i$ for all $i \in [1, k]$.

Note that, as can be shown by simple induction on the structure, for any graph D and pattern P in SP[AOF] the semantics is such that any two different mappings in the set of mappings $[\![P]\!]_D$ are incompatible, and the intersection of the domains of mappings in $[\![P]\!]_D$ is non-empty. These two properties play an important role in the chapter, and we call sets of mappings satisfying them *consistent*.

Example 3.2. *Continuing with Example 3.1, consider an RDF graph D consisting of following triples:*

$$\begin{array}{ll}(\texttt{John}, \texttt{type}, \texttt{Person}), & (\texttt{John}, \texttt{age}, 26), \\ (\texttt{Peter}, \texttt{type}, \texttt{Person}), & (\texttt{Peter}, \texttt{age}, 32), \\ (\texttt{Susan}, \texttt{type}, \texttt{Person}), & (\texttt{Susan}, \texttt{age}, 32), \\ (\texttt{Susan}, \texttt{email}, \texttt{susan@example.org}). & \end{array}$$

Then we have that $[\![P_1]\!]_D$ consists of the mappings $[?\texttt{X} \mapsto \texttt{John}]$, $[?\texttt{X} \mapsto \texttt{Peter}]$ and $[?\texttt{X} \mapsto \texttt{Susan}]$, which correspond to the list of people in D. Moreover, we have that $[\![P_2]\!]_D = \{[?\texttt{X} \mapsto \texttt{Peter}], [?\texttt{X} \mapsto \texttt{Susan}]\}$, as the age of John *is* 26*. Finally, we have that $[\![P_3]\!]_D$ consists of the mappings*

$$\begin{aligned} \mu_1 &= [?\texttt{X} \mapsto \texttt{Peter}], \\ \mu_2 &= [?\texttt{X} \mapsto \texttt{Susan}, ?\texttt{Y} \mapsto \texttt{susan@example.org}]. \end{aligned}$$

On the one hand, we do not have the email of Peter *in D, so only variable* ?X *is assigned with a value in the mapping μ_1. On the other hand, the email of* Susan *is in D, so we have a value for variable* ?Y *in the mapping μ_2.*

Finally, the *size* of a graph pattern P, denoted by $\mathsf{size}(P)$, is the number of triple patterns and atomic filter conditions in P.

3.2.2 Well-designed patterns

Previous work on SPARQL has identified some anomalies that arise when the **OPT** operator can be used arbitrarily, and [84] showed that one can avoid these anomalies by making a natural restriction on the use of **OPT**. A graph pattern P is said to be *safe* if for every sub-pattern $(P_1 \, \textbf{FILTER} \, R)$ of P, it holds that $\text{var}(R) \subseteq \text{var}(P_1)$. A graph pattern P is said to be *well-designed* if P is safe and for every sub-pattern P_1 of P of the form $(P_L \, \textbf{OPT} \, P_R)$, and for every variable $\mathbf{v} \in \text{var}(P_R)$, if $\mathbf{v}$ is mentioned outside of P_1 in P, then $\mathbf{v} \in \text{var}(P_L)$. With the previous, we define the fragments SP[AOwd] and SP[AOFwd] as classes of well-designed graph patterns using the respective operators.

Example 3.3. *The graph pattern P_3 defined in Example 3.1 is well-designed. On the other hand, if*

$$P_4 = ((?\mathtt{Y}, \mathtt{type}, \mathtt{Publication})\,\mathbf{OPT}\,(?\mathtt{X}, \mathtt{email}, ?\mathtt{Z})),$$

then the graph pattern $P_5 = (?\mathtt{X}, \mathtt{type}, \mathtt{Person})\,\mathbf{AND}\,P_4$ is not well-designed. To see why this is the case, notice that the variable $?\mathtt{X}$ is mentioned in the right-hand side of P_4 and outside P_4 in the triple $(?\mathtt{X}, \mathtt{type}, \mathtt{Person})$, but it is not mentioned in the left-hand side of P_4. What is unnatural about the graph pattern P_5 is that the triple $(?\mathtt{X}, \mathtt{email}, ?\mathtt{Z})$ has been placed to give optional information to the triple $(?\mathtt{Y}, \mathtt{type}, \mathtt{Publication})$, but it is actually giving optional information to the outside triple $(?\mathtt{X}, \mathtt{type}, \mathtt{Person})$.

Empirical studies have shown that well-designed graph patterns are commonly used in practice [85]. Well-designed patterns have many desirable properties. First, the complexity of the query evaluation problem for well-designed graph patterns is lower than for the entire language [84] (**coNP** versus **PSPACE**), even if only the **OPT** operator is considered [93]. Second, well-designed graph patterns are suitable for query optimisation [84, 70, 86]; in particular, this restriction allows the definition of some simple reordering and optimisation rules [84, 93, 70]. Third, this class of graph patterns captures the intuition behind the **OPT** operator, which is to allow information to be added to an answer whenever it is available, and not to reject such an answer if the information is not present. This intuition, which is formalised in the following paragraph, does not hold for an arbitrary query in SP[AOF].

A mapping μ_1 is *subsumed by* a mapping μ_2, denoted by $\mu_1 \sqsubseteq \mu_2$, if $\text{dom}(\mu_1) \subseteq \text{dom}(\mu_2)$ and $\mu_1(\mathbf{v}) = \mu_2(\mathbf{v})$ for every variable $\mathbf{v} \in \text{dom}(\mu_1)$. Assume that we have two RDF graphs D_1 and D_2 such that $D_1 \subseteq D_2$. If a SPARQL pattern P mentioning only the operators **AND** and **FILTER** is evaluated over D_1 and D_2, then we have that $[\![P]\!]_{D_1} \subseteq [\![P]\!]_{D_1}$. Thus, such a query P is monotone in the sense that if new information is added to an RDF graph then no answer is lost. If we are also allowed to use the **OPT** operator in P, then we would expect a similar behaviour, which is referred to as *weak monotonicity* [14]. More precisely, given that $D_1 \subseteq D_2$, it should be the case that for every mapping $\mu_1 \in [\![P]\!]_{D_1}$, there exists a mapping $\mu_2 \in [\![P]\!]_{D_2}$ such that $\mu_1 \sqsubseteq \mu_2$, as the **OPT** operator was designed only to add information to an answer whenever it is available. However, there exist patterns in SP[AOF] that are not weakly monotone [14]; in fact, P_5 in Example 3.3 is such a query. Well-designed graph patterns come as a solution to this fundamental problem, as shown in the following proposition.

Proposition 3.1 ([14])**.** *Every* SP[AOFwd] *graph pattern is weakly monotone.*

Last, but not least, the semantics of well-designed graph patterns can be characterised in terms of the notion of subsumption of mappings, which is a useful property that will be utilised in this work. A graph pattern P' is a *reduction* of a graph pattern P if P' can be obtained from P by replacing a sub-pattern $(P_1 \,\mathbf{OPT}\, P_2)$ of P by P_1, that is, if P' is obtained by deleting some optional part of P. The reflexive and transitive closure of the reduction relation is denoted by $\trianglelefteq$. Moreover, andify(P) is the graph pattern obtained from P by replacing every **OPT** operator in P by the **AND** operator. The set of partial answers of a graph pattern P over an RDF graph D, denoted by $\mathsf{partials}(P, D)$, is the set of all mappings μ for which there exists $P' \trianglelefteq P$ such that $\mu \in [\![\text{andify}(P')]\!]_D$. As shown in the following proposition, partial answers and the notion of subsumption of mappings can be used to characterise the evaluation of a well-designed graph pattern.

Proposition 3.2 ([84])**.** *For every RDF graph D, graph pattern P in* SP[AOFwd] *and mapping μ, it holds that $\mu \in [\![P]\!]_D$ if and only if μ is a maximal mapping (with respect to $\sqsubseteq$) in* $\mathsf{partials}(P, D)$.

Well-designed patterns also admit **OPT** *normal form*, in which arguments of **AND** and **FILTER** do not use **OPT** [84]; this normalisation can be performed in polynomial time. Moreover, in well-designed patterns the bound operator is moot, as in **OPT** normal form it can be always replaced by True [120]. In this work we assume normalised, well-designed patterns without bound.

Besides the class SP[AOFwd] of all well-designed patterns, we also study the reverse engineering problem for the fragment SP[AOwd] of well-designed graph patterns formed using only **AND** and **OPT**, and the fragment SP[$\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}$] of SP[$\mathsf{AOFwd}$] obtained by disallowing the use of disjunction in filter expression and restricting the use of negation to only inequalities.

3.2.3 Pattern trees

Well-designed graph patterns can also be represented as pattern trees, a representation which will be used heavily in what follows. In this subsection, we define this representation.

Firstly, define a *pattern tree* to be a tuple $P = (V_P, E_P, r_P, \lambda_P, \delta_P)$, where V_P is a set of nodes, $E_P \subseteq V_P \times V_P$ is a set of edges such that for every node $n \in V_P \setminus \{r_P\}$ there exists exactly one node $m \in V_P$ such that $(m, n) \in E_P$ (i.e. each node, except

r_P, has exactly one parent), $r_P \in V_P$ is the root node which has no parent, λ_P is a function which associates to each node $n \in V_P$ a set $\lambda_P(n)$ of triple patterns, and δ_P is a function which associates to each node $n \in V_P$ a SPARQL built-in condition $\delta_P(n)$.

As with SP[AOFwd] queries, let $\text{var}(P)$ be the variables mentioned in P, and we say a pattern tree P is well-designed if and only if (i) the subtree induced by each variable is connected, (ii) P is *safe*, meaning that for each node $n \in V_P$ it is the case that $\text{var}(\delta_P(n)) \subseteq \text{var}(\lambda_P(n))$. We will abuse notation by stating that the set of well-designed pattern trees is equal to SP[AOFwd] (below we define the semantics of said trees, and the equivalence was shown in [70]).

For evaluating pattern trees over RDF graphs, we use analogous concepts to those presented in the preliminaries. Firstly, we say pattern tree $P' \in$ SP[AOFwd] is a reduction of $P \in$ SP[AOFwd] if and only if P' is obtained from P by removing a leaf node, and denote by $\trianglelefteq$ the reflexive and transitive closure of the reduction relation (in other words, $P' \trianglelefteq P$ if and only if P' is obtained from P by iteratively removing leaf nodes). Now, given an RDF graph D, a mapping μ, and a pattern tree $P \in$ SP[AOFwd], we define $\mu \in [\![P]\!]_D$ if and only if there exists a pattern tree P_μ such that $P_\mu \trianglelefteq P$ such that $\mu \in [\![\text{andify}(P_\mu)]\!]_D$ (i.e. $\mu \in \mathsf{partials}(D, P)$), and furthermore μ is maximal in $\mathsf{partials}(D, P)$ with respect to $\sqsubseteq$. Note that andify(P) is the query obtained by connecting all triple patterns in all nodes of P and filtering this with the conjunction of all **FILTER** expressions present in all nodes of P (again, analogously to the case of graph patterns).

Given pattern tree $P \in$ SP[AOFwd], let the function top_P associate to each variable $\mathbf{v} \in \text{var}(P)$ the node $m = \mathsf{top}_P(\mathbf{v}) \in V_P$ such that $\mathbf{v} \in \text{dom}(\lambda_P(m))$ and m is an ancestor of all nodes $n \in V_P$ for which $\mathbf{v} \in \text{dom}(\lambda_P(n))$ (note that $\mathsf{top}_P(\mathbf{v})$ exists and is unique, due to the connectedness condition on well-designed pattern trees). Also, given a node $n \in V_P$, let $\mathsf{scope}_P(n)$ be the set of variables $\mathsf{scope}_P(n) = \{\mathbf{v} \mid \mathsf{top}_P(\mathbf{v})$ is an ancestor of n in $P\}$; intuitively, these are the variables that may appear in n. Finally, we may assume that pattern trees are in *productive form*, that is, for every node $n \in V(P)$ there is a variable $\mathbf{v} \in \text{var}(P)$ such that $n = \mathsf{top}_P(\mathbf{v})$[70].

3.2.4 Complexity classes

In the study of the computational complexity of the reverse engineering problems, we consider the usual complexity classes **PTIME**, **NP** and **coNP**, along with the complexity classes Σ_2^p and **DP**. A prototypical complete problem for Σ_2^p is $\exists\forall$3SAT, that is, the problem of verifying, given a quantified Boolean formula ϕ of the form

$\exists\bar{x}\forall\bar{y}\,\neg\psi$ with ψ a propositional formula (without quantifiers) in conjunctive normal form with each clause using exactly three literals, whether ϕ is valid. **DP** is the class of problems L for which there exist languages L_1 and L_2 in **NP** such that $L = L_1 - L_2$ [82] (or, equivalently, for which there exist L_1 in **NP** and L_2 in **coNP** such that $L = L_1 \cap L_2$). The problem 3SatUnsat of deciding validity of a quantified formula ϕ of the form $\exists\bar{x}\,\psi_1 \wedge \forall\bar{y}\,\neg\psi_2$, where ψ_1 and ψ_2 are in conjunctive normal form with three literals per clause, is **DP**-complete. It is known that **NP** $\subseteq$ **DP**, **coNP** $\subseteq$ **DP** and **DP** $\subseteq \Sigma_2^p$. The previous inclusions are believed to be proper.

3.3 Reverse engineering problems

We now formally define the reverse engineering problems considered in this chapter. Informally, reverse engineering problems ask whether there exists a query, or *realizer*, which fits a set of examples. Let $\mathcal{F}$ be any of the SPARQL fragments defined in Section 3.2.2 (e.g. SP[AOFwd]). Then, the positive examples reverse engineering problem is defined as:

$$\textsc{RevEng}^{+}(\mathcal{F}) = \{(D,\Omega) \mid D \text{ is a nonempty RDF graph}, \\ \Omega \text{ is a set of mappings, and } \exists P \in \mathcal{F} : \Omega \subseteq [\![P]\!]_D\}.$$

Moreover, the positive-and-negative examples reverse engineering problem is defined as:

$$\textsc{RevEng}^{\pm}(\mathcal{F}) = \{(D,\Omega,\bar{\Omega}) \mid D \text{ is a non-empty RDF graph}, \\ \Omega, \bar{\Omega} \text{ are sets of mappings, and} \\ \exists P \in \mathcal{F} : \Omega \subseteq [\![P]\!]_D \text{ and } \bar{\Omega} \cap [\![P]\!]_D = \emptyset\}.$$

In this case, without loss of generality we always silently assume that $\Omega \cap \bar{\Omega} = \emptyset$, since otherwise the problem is trivial. Finally, the exact-answers reverse engineering problem is defined as:

$$\textsc{RevEng}^{\mathsf{E}}(\mathcal{F}) = \{(D,\Omega) \mid D \text{ is a non-empty RDF graph}, \\ \Omega \text{ is a set of mappings, and } \exists P \in \mathcal{F} : \Omega = [\![P]\!]_D\}.$$

We will call (D,Ω) (and $(D,\Omega,\bar{\Omega})$) an *instance* of the problem and P a *realizer* for the instance.

In order to study the complexity of these reverse engineering problems, we first must understand the corresponding problem of verifying that a given pattern (a.k.a.

query) fits some example data. Hence, the positive examples verification problem is defined as:

$$\begin{aligned}\textsc{Verify}^{+}(\mathcal{F}) = \{(D, P, \Omega) \mid & D \text{ is a nonempty RDF graph,}\\ & \Omega \text{ is a set of mappings}, P \in \mathcal{F}, \text{ and } \Omega \subseteq [\![P]\!]_D\}.\end{aligned}$$

The positive-and-negative examples verification problem is:

$$\begin{aligned}\textsc{Verify}^{\pm}(\mathcal{F}) = \{(D, P, \Omega, \bar{\Omega}) \mid & D \text{ is a non-empty RDF graph,}\\ & \Omega, \bar{\Omega} \text{ are sets of mappings,}\\ & P \in \mathcal{F}, \Omega \subseteq [\![P]\!]_D, \text{ and } \bar{\Omega} \cap [\![P]\!]_D = \emptyset\}.\end{aligned}$$

Finally, the exact-answers verification problem is defined as:

$$\begin{aligned}\textsc{Verify}^{\mathsf{E}}(\mathcal{F}) = \{(D, P, \Omega) \mid & D \text{ is a non-empty RDF graph,}\\ & \Omega \text{ is a set of mappings}, P \in \mathcal{F}, \text{ and } \Omega = [\![P]\!]_D\}.\end{aligned}$$

Alongside the general versions of the reverse engineering and verification problems, we will consider the case where the number of variables mentioned in Ω (and $\bar{\Omega}$) is bounded by a fixed constant. An upper bound on the complexity of the verification problem when the number of variables is fixed will imply the same upper bound when the pattern is fixed but the data is scaled, i.e. an upper bound in the *data complexity* of verification.

Note that the problems with both positive and negative examples are clearly at least as difficult as their counterparts with only positive examples. However, the exact-answers problems are not immediately reducible to others.

3.4 Complexity of Reverse Engineering

Having defined the decision problems to be studied, we now turn to their computational complexity. In this section we will first study the complexity of the auxiliary verification problems. While interesting in their own right, the verification problems will be important mainly for their role in obtaining complexity results for the reverse engineering problems. We first give a general overview of the strategy to be used to decide a reverse engineering problem: given a fixed SPARQL fragment $\mathcal{F}$ and a set of example mappings, we may divide the problem into two main challenges:

- Since the fragment $\mathcal{F}$ will usually permit infinitely many queries, we cannot test every query in the fragment as a candidate. Thus, we seek to *bound the set of queries to be considered.* In fact, we will seek to show that if there is any query realising the input, then there is a *canonical realizer.*

- Given a "candidate query" in the fragment, we must decide if it fits the example instance by solving the corresponding verification problem. Then, once we obtain a set of candidate queries it becomes a matter of verifying them one by one.

The algorithmic strategy, then, will be to first construct the canonical query, and then verify that it in fact fits the examples. In what follows we first study the verification problems, and then define the canonical queries in order to study the reverse engineering problems.

3.4.1 Complexity of verification problems

We now examine the complexity of the verification problems, starting with the upper bounds, and progressing towards the results summarised in Table 3.1. The positive-and-negative examples problem for SP[A], that is, VERIFY$^{\pm}$(SP[A]), admits a straightforward polynomial-time algorithm:

Proposition 3.3. *The following problems are in* **PTIME***:*

1. VERIFY$^{\pm}$(SP[A]),

2. VERIFY^{+}(SP[A]).

Proof. **Item 1.** For VERIFY$^{\pm}$(SP[A]), given an input $(D, \Omega, \bar{\Omega}, P)$, we must first check that $\Omega \subseteq [\![P]\!]_D$: for each $\mu \in \Omega$ we confirm that $\mathrm{dom}(\mu) = \mathrm{var}(P)$ and that $\mu(t) \in D$ for every triple pattern t in P. Secondly, we must check that $\bar{\Omega} \cap [\![P]\!]_D = \emptyset$: for every $\bar{\mu} \in \bar{\Omega}$ we must confirm that either $\mathrm{dom}(\bar{\mu}) \neq \mathrm{var}(P)$ or that there is a tuple t in P such that $\mu(t) \notin D$.

Item 2. This is a trivial corollary of item 1. □

We defer the case of VERIFY$^{\mathsf{E}}$(SP[A]) for the moment. If we now allow the **OPT** operator, the problem becomes harder. We give an upper bound for VERIFY^{+} (SP[AOFwd]); here, checking each example mapping μ involves showing that it is maximal, generating the added complexity. In what follows we use the notation $\mu \sqsubset \nu$ to indicate that mapping μ is properly subsumed by mapping ν, that is, $\mu \sqsubseteq \nu$ and $\mu \neq \nu$:

Proposition 3.4. $\textsc{Verify}^{+}(\text{SP}[\mathsf{AOFwd}])$ *is in* **coNP**.

Proof. Consider, for the complement of $\textsc{Verify}^{+}(\text{SP}[\mathsf{AOFwd}])$, the following **NP** algorithm. Given input (D, P, Ω) we must decide whether $\Omega \nsubseteq [\![P]\!]_D$, for which we use the characterisation provided in Proposition 3.2. More precisely, for each $\mu \in \Omega$, first check whether $\mu \in \mathsf{partials}(P, D)$; if not return accept (this can be done in polynomial time, as shown in [84]). Otherwise, we now attempt to verify that μ is not a maximal partial solution by guessing a mapping ν such that (i) $\text{dom}(\nu)$ consists of variables used in Ω and P, and (ii) the range of ν consists of constants used in D. Notice that ν is of polynomial size in the size of the input. We check whether $\nu \in \mathsf{partials}(P, D)$ and $\mu \sqsubset \nu$. If these conditions are both true, then $\mu \notin [\![P]\!]_D$ and we accept; otherwise, we reject. □

Corollary 3. *The following problems are in* **coNP***:*

1. $\textsc{Verify}^{+}(\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$,
2. $\textsc{Verify}^{+}(\text{SP}[\mathsf{AOwd}])$.

Interestingly, if the number of variables in Ω is assumed to be bounded by a fixed constant, then guessing is not necessary and the problem is in **PTIME**. In particular, this implies that the verification problem can be solved in polynomial time in data complexity.

We now show that $\textsc{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{AOFwd}])$ is also in **coNP**, for which the following characterisation will be useful:

Lemma 3.1. *Given RDF graph D, set of mappings Ω, and pattern $P \in \text{SP}[\mathsf{AOFwd}]$, it holds that $[\![P]\!]_D \nsubseteq \Omega$ if and only if there exists a mapping $\mu \in \mathsf{partials}(P, D)$ such that (i) $\mu \notin \Omega$ and (ii) for every $\nu \in \Omega$, if $\mu \sqsubset \nu$ then $\nu \notin \mathsf{partials}(P, D)$.*

Proof. First assume that there is a mapping $\mu \in \mathsf{partials}(P, D)$ such that both items hold. There are two options for μ:

- Suppose $\mu \in [\![P]\!]_D$. Then, due to the first item, we have that $[\![P]\!]_D \nsubseteq \Omega$.
- Suppose $\mu \notin [\![P]\!]_D$. In this case, since $\mu \in \mathsf{partials}(P, D)$, there exists a mapping $\mu^* \in [\![P]\!]_D$ such that $\mu \sqsubset \mu^*$. There are two cases to consider for μ^*:
  - if $\mu^* \in \Omega$, then, due to the second item, we have that $\mu^* \notin \mathsf{partials}(P, D)$. Thus, $\mu^* \notin [\![P]\!]_D$, which leads to a contradiction.

– if $\mu^* \notin \Omega$ then we also conclude that $[\![P]\!]_D \not\subseteq \Omega$.

To prove the other direction of the lemma, assume that $[\![P]\!]_D \not\subseteq \Omega$. Then there exists a mapping $\mu \in [\![P]\!]_D$ such that $\mu \notin \Omega$. By definition we have that $\mu \in \mathsf{partials}(P, D)$ and, thus, we only have to show that the second item of the lemma holds for μ to conclude the proof. Now consider a mapping $\nu \in \Omega$ such that $\mu \sqsubset \nu$. If $\nu \in \mathsf{partials}(P, D)$, then we obtain a contradiction with the fact that $\mu \in [\![P]\!]_D$, as μ is a maximal partial mapping in $\mathsf{partials}(P, D)$. □

Lemma 3.1 allows us to obtain an upper bound for $\text{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{AOFwd}])$:

Proposition 3.5. $\text{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{AOFwd}])$ *is in* **coNP**.

Proof. We prove that $\text{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{AOFwd}]) \in$ **coNP** by showing that its complement is in **NP**. On input (D, Ω, P), we first check in **NP** whether $\Omega \not\subseteq [\![P]\!]_D$ as in the proof of Proposition 3.4. If this holds, accept. Otherwise, check whether $[\![P]\!]_D \not\subseteq \Omega$. Now by Lemma 3.1, we must guess a mapping μ such that $\mu \notin \Omega$, $\text{dom}(\mu)$ consists of variables occurring in P, the range of μ consists of the constants used in D and $\mu \in \mathsf{partials}(P, D)$; in particular, notice that μ is of polynomial size in the size of the input. Then, for every $\nu \in \Omega$ such that $\mu \sqsubset \nu$, we verify whether $\nu \notin \mathsf{partials}(P, D)$, which can be done in polynomial time. If this verification succeeds, accept; otherwise, reject. □

Corollary 4. *The following problems are in* **coNP***:*

1. $\text{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$,
2. $\text{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{AOwd}])$,
3. $\text{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{A}])$.

As before, if the number of variables in Ω is bounded by a fixed constant, then the resulting problem is again in **PTIME**.

Our final upper bound is for $\text{Verify}^{\pm}(\text{SP}[\mathsf{AOFwd}])$:

Proposition 3.6. $\text{Verify}^{\pm}(\text{SP}[\mathsf{AOFwd}])$ *is in* **DP**.

Proof. Let $\text{Verify}^{-}(\text{SP}[\mathsf{AOFwd}])$ be the following decision problem: on input $(D, \bar{\Omega}, P)$, return True if and only if $\bar{\Omega} \cap [\![P]\!]_D = \emptyset$. The complement of this problem is easily seen to be in **NP**: for each mapping $\bar{\mu} \in \bar{\Omega}$, verify that either $\bar{\mu} \notin \mathsf{partials}(D, P)$ (which can be checked in polynomial time) or, if $\bar{\mu} \in \mathsf{partials}(D, P)$, guess a mapping

ν and check that $\bar{\mu} \sqsubset \nu$ and $\nu \in \mathsf{partials}(D, P)$, accepting if this holds. Now note that an input $(D, \Omega, \bar{\Omega}, P)$ is in $\textsc{Verify}^{\pm}(\text{SP}[\mathsf{AOFwd}])$ if and only if $(D, \Omega, P) \in \textsc{Verify}^{+}(\text{SP}[\mathsf{AOFwd}])$ and $(D, \bar{\Omega}, P) \in \textsc{Verify}^{-}(\text{SP}[\mathsf{AOFwd}])$, from which it may be straightforwardly concluded that $\textsc{Verify}^{\pm}(\text{SP}[\mathsf{AOFwd}]) \in \mathbf{DP}$. $\square$

Corollary 5. *The following problems are in* **DP***:*

1. $\textsc{Verify}^{\pm}(\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$*,*
2. $\textsc{Verify}^{\pm}(\text{SP}[\mathsf{AOwd}])$*.*

Having established complexity upper bounds for all the verification problems, we now turn to the matching lower bounds. We will first show **coNP**-hardness for $\textsc{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{A}])$, which is also the matching lower bound for $\textsc{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{AOwd}])$, $\textsc{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$, and $\textsc{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{AOFwd}])$.

Proposition 3.7. $\textsc{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{A}])$ *is* **coNP***-hard.*

Proof. Consider the following problem, which is known to be **NP**-complete:

3−Colourability

Input:	An undirected graph $G = (V, E)$ with vertices V and edges E.
Output:	True if it is possible to colour all the vertices in 3 different colours such that adjacent vertices do not have the same colour, and False otherwise.

We prove the statement by showing $\overline{3\text{−}\textsc{Colourability}} \leq^{p}_{m} \textsc{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{AOFwd}])$, i.e. there is a polynomial-time many-one reduction from the complement of the problem 3−Colourability to $\textsc{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{AOFwd}])$.

Given a graph $G = (V, E)$, construct an instance (D_G, Ω_G, P_G) as follows:

- $D_G = \{(\mathsf{c}_1, \mathsf{e}, \mathsf{c}_2), (\mathsf{c}_2, \mathsf{e}, \mathsf{c}_1), (\mathsf{c}_1, \mathsf{e}, \mathsf{c}_3), (\mathsf{c}_3, \mathsf{e}, \mathsf{c}_1), (\mathsf{c}_2, \mathsf{e}, \mathsf{c}_3), (\mathsf{c}_3, \mathsf{e}, \mathsf{c}_2)\}$, where URIs $\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3 \in \mathsf{U}$ represent the three colours (note that D_G does not depend on G in this reduction),
- $\Omega_G = \emptyset$,
- P_G is a conjunction of triple patterns $(?\mathtt{X}_i, \mathsf{e}, ?\mathtt{X}_j)$ and $(?\mathtt{X}_j, \mathsf{e}, ?\mathtt{X}_i)$ for every edge $\{c_i, c_j\} \in E$ (note that for every node $c_i \in V$ we have a variable $?\mathtt{X}_i \in \text{var}(P_G)$).

It is straightforward to show that G is not 3-colourable if and only if $\Omega_G = [\![P_G]\!]_{D_G}$, as each mapping in $[\![P_G]\!]_{D_G}$ will represent a valid colouring of the nodes in V. $\square$

The proposition above completes the picture for the complexity of verification problems for SP[A], as well as the VERIFY$^{\mathsf{E}}$ problem for all the fragments considered. We now continue with the lower bounds on verification problems for SP[AOwd] and SP[AOFwd]:

Proposition 3.8. VERIFY^{+}(SP[AOwd]) *is* **coNP**-*hard.*

Proof. Consider the following decision problem:

EVAL(SP[AOwd])

Input:	(D, μ, P) where D is an RDF graph, μ is a mapping, and $P \in$ SP[AOwd].
Output:	True if $\mu \in [\![P]\!]_D$, and False otherwise.

In [84] it was shown that EVAL(SP[AOwd]) is **coNP**-hard. We show that EVAL (SP[AOwd]) $\leq^p_m$ VERIFY^{+}(SP[AOwd]), i.e. there exists a polynomial-time many-one reduction from the EVAL(SP[AOwd]) problem to VERIFY^{+}(SP[AOwd]). We will thus conclude that VERIFY^{+}(SP[AOwd]) is also **coNP**-hard.

Given an input (D, μ, P) to the EVAL(SP[AOwd]) problem, construct the input $(D, \{\mu\}, P)$ to the VERIFY^{+}(SP[AOwd]) problem and note that $\mu \in [\![P]\!]_D$ if and only if $\{\mu\} \subseteq [\![P]\!]_D$. □

Corollary 6. *The following problems are* **coNP**-*hard:*

1. VERIFY^{+}(SP[AOF$_{\wedge,=,\neq}$wd]),
2. VERIFY^{+}(SP[AOFwd]).

We conclude this section with the matching lower bound for the decision problem VERIFY$^{\pm}$(SP[AOwd]) (and, hence, for the fragments with **FILTER**), followed by Theorem 3.1 which summarises the results:

Proposition 3.9. VERIFY$^{\pm}$(SP[AOwd]) *is* **DP**-*hard.*

Proof. Consider the following problem, which is known to be **DP**-complete.

3SATUNSAT

Input:	A quantified formula ϕ of the form $\exists \bar{x}\, \psi^1 \wedge \forall \bar{y}\, \neg \psi^2$, where $\bar{x}$ and $\bar{y}$ are disjoint tuples of variables, and ψ^1 and ψ^2 are conjunctions of clauses of the form $(\ell_1 \vee \ell_2 \vee \ell_3)$, where ℓ_1, ℓ_2, ℓ_3 are either variables in $\bar{x}$ and $\bar{y}$, respectively, or their negations.
Output:	True if ϕ is valid, and False otherwise.

We will show that 3SatUnsat $\leq_m^p$ Verify$^\pm$(SP[AOwd]), that is, there is a polynomial-time many-one reduction from 3SatUnsat to Verify$^\pm$(SP[AOwd]).

Consider an instance $\phi = \exists \bar{x}\, \psi^1 \wedge \forall \bar{y}\, \neg\psi^2$ of the 3SatUnsat problem. Assume that $\psi^1 = \gamma_1 \wedge \cdots \wedge \gamma_m$, where for each $i \in [1,m]$ we have $\gamma_i = r_{i,1} \vee r_{i,2} \vee r_{i,3}$, and each $r_{j,k}$ is either a variable in $\bar{x}$ or the negation of one. Analogously, assume that $\psi^2 = \delta_1 \wedge \cdots \wedge \delta_n$, where for each $j \in [1,n]$ we have $\delta_j = s_{j,1} \vee s_{j,2} \vee s_{j,3}$, and each $s_{j,k}$ is either a variable in $\bar{y}$ or the negation of one. For each literal $r_{i,k}$ let $\mathrm{var}(r_{i,k})$ be the variable that is mentioned (be it negated or not); similarly, for each literal $s_{j,k}$ let $\mathrm{var}(s_{j,k})$ be the variable that is mentioned.

We construct a graph D, sets of mappings Ω and $\bar{\Omega}$, and a pattern $P \in$ SP[AOwd] such that it holds that $\Omega \subseteq [\![P]\!]_D$ and $\bar{\Omega} \cap [\![P]\!]_D = \emptyset$ if and only if ϕ is valid.

We start the construction with the pattern P. Let $P = P_0$ **OPT** Q with $P_0 = (?a, R, ?b)$ and Q a conjunction of the following triple patterns:

1. $(?\mathtt{b}, U, ?\mathtt{u})$, $(?\mathtt{b}, W, ?\mathtt{w})$,
2. $(?\mathtt{u}, C_i, ?\mathtt{c}_i)$ for each $i \in [1,m]$ (one for each clause γ_i in ψ^1),
3. $(?\mathtt{w}, D_j, ?\mathtt{d}_j)$ for each $j \in [1,n]$ (one for each clause δ_j in ψ^2),
4. $(?\mathtt{c}_i, V_1, ?\mathtt{v}_{\mathrm{var}(r_{i,1})}), (?\mathtt{c}_i, V_2, ?\mathtt{v}_{\mathrm{var}(r_{i,2})}), (?\mathtt{c}_i, V_3, ?\mathtt{v}_{\mathrm{var}(r_{i,3})})$ for each $i \in [1,m]$, with $\gamma_i = r_{i,1} \vee r_{i,2} \vee r_{i,3}$,
5. $(?\mathtt{d}_j, V_1, ?\mathtt{v}_{\mathrm{var}(s_{j,1})}), (?\mathtt{d}_j, V_2, ?\mathtt{v}_{\mathrm{var}(s_{j,2})}), (?\mathtt{d}_j, V_3, ?\mathtt{v}_{\mathrm{var}(s_{j,3})})$ for each $j \in [1,n]$, with $\delta_j = s_{j,1} \vee s_{j,2} \vee s_{j,3}$.

Next we describe the graph D. It consists of two disjoint parts D^1 and D^2, defined as follows. Let D^1 contain the following triples:

- (a^1, R, b^1), (b^1, U, u^1), (b^1, W, w^1),
- $(u^1, C_i, c^1_{i,p})$ for each $i \in [1,m]$ and $p \in [1,7]$,
- $(c^1_{i,p}, V_1, h^1_{\mathrm{var}(r_{i,1})}), (c^1_{i,p}, V_2, h^1_{\mathrm{var}(r_{i,2})}), (c^1_{i,p}, V_3, h^1_{\mathrm{var}(r_{i,3})})$ for each $i \in [1,m]$ and $p \in [1,7]$, where each σ_p is one of the seven satisfying assignments of $\gamma_i = r_{i,1} \vee r_{i,2} \vee r_{i,3}$ and for each $h^1_{\mathrm{var}(r_{i,k})}$:

$$h^1_{\mathrm{var}(r_{i,k})} = \begin{cases} t^1_{\mathrm{var}(r_{i,k})} & \text{if } \sigma_p(\mathrm{var}(r_{i,k})) = \mathsf{True}, \\ f^1_{\mathrm{var}(r_{i,k})} & \text{if } \sigma_p(\mathrm{var}(r_{i,k})) = \mathsf{False}, \end{cases}$$

- (w^1, D_j, d^1_j) for each $j \in [1,n]$,

	SP[A]	SP[AOwd]	SP[AOF$_{\wedge,=,\neq}$wd]	SP[AOFwd]
VERIFY$^+$	in **PTIME**	**coNP**-c	**coNP**-c	**coNP**-c
VERIFY$^{\mathsf{E}}$	**coNP**-c	**coNP**-c	**coNP**-c	**coNP**-c
VERIFY$^\pm$	in **PTIME**	**DP**-c	**DP**-c	**DP**-c

Table 3.1: Complexity of verification problems.

- $(d^1_j, V_1, v^1_{\mathrm{var}(s_{j,1})}), (d^1_j, V_2, v^1_{\mathrm{var}(s_{j,2})}), (d^1_j, V_3, v^1_{\mathrm{var}(s_{j,3})})$ for each $j \in [1, n]$, with $\delta_j = s_{j,1} \vee s_{j,2} \vee s_{j,3}$.

Part D^2 is defined analogously to D^1, and consists of the following triples:

- (a^2, R, b^2), (b^2, U, u^2), (b^2, W, w^2),
- (u^2, C_i, c^2_i) for each $i \in [1, m]$,
- $(c^2_i, V_1, v^2_{\mathrm{var}(r_{i,1})}), (c^2_i, V_2, v^2_{\mathrm{var}(r_{i,1})}), (c^2_i, V_3, v^2_{\mathrm{var}(r_{i,1})})$ for each $i \in [1, m]$, with $\gamma_i = r_{i,1} \vee r_{i,2} \vee r_{i,3}$,
- $(w^2, D_j, d^2_{j,p})$ for each $j \in [1, n]$ and $p \in [1, 7]$,
- $(d^2_{j,p}, V_1, h^2_{\mathrm{var}(s_{j,1})}), (d^2_{j,p}, V_2, h^2_{\mathrm{var}(s_{j,2})}), (d^2_{j,p}, V_3, h^2_{\mathrm{var}(s_{j,3})})$ for each $j \in [1, n]$ and $p \in [1, 7]$, where each σ_p is one of the seven satisfying assignments of $\delta_j = s_{j,1} \vee s_{j,2} \vee s_{j,3}$, and each $s_{j,k}$ and each $h^2_{\mathrm{var}(s_{j,k})}$:

$$h^2_{\mathrm{var}(s_{j,k})} = \begin{cases} t^2_{\mathrm{var}(s_{j,k})} & \text{if } \sigma_p(\mathrm{var}(s_{j,k})) = \mathsf{True}, \\ f^2_{\mathrm{var}(s_{j,k})} & \text{if } \sigma_p(\mathrm{var}(s_{j,k})) = \mathsf{False}, \end{cases}$$

Finally, we define the sets of mappings $\bar{\Omega}$ and Ω as follows:

$$\begin{array}{rcccl} \bar{\Omega} & = & \{\bar{\mu}\} & = & \{[?\mathsf{a} \mapsto a^1, ?\mathsf{b} \mapsto b^1]\}, \\ \Omega & = & \{\mu\} & = & \{[?\mathsf{a} \mapsto a^2, ?\mathsf{b} \mapsto b^2]\}. \end{array}$$

It is a matter of technicality to show that $\Omega \subseteq [\![P]\!]_D$ and $\bar{\Omega} \cap [\![P]\!]_D = \emptyset$ if and only if ϕ is satisfiable. □

Theorem 3.1. *Complexity bounds for the verification problems are as stated in in Table 3.1.*

Proof. This result is due to the propositions and corollaries proven in this subsection. □

Symbol	Concept	Reference
atype	Atomic type	Page 40
Cov	Coverage	Page 44
$\mathcal{C}(\Omega)$	Structure	Page 44
scope	Scope	Page 45
P_{can}	Canonical query	Page 45
	Compatibility (between a query and a set of mappings)	Page 45
	Closedness (of a set of variables)	Page 46

Table 3.2: Glossary of concepts used in this chapter.

3.4.2 Reverse engineering without OPTIONAL

We now turn to the reverse engineering decision problems, and progress towards the results summarised in Table 3.3 (page 80). In this subsection we provide complexity upper bounds for the different variants. As mentioned previously, the algorithms which witness these upper bounds follow a common pattern, which consists in first constructing a candidate query (or *realizer*) with the property that if it does not correctly fit the input examples, then there does not exist any query which does; then we must only verify that this realizer fits the input examples.

To assist the reader, Table 3.2 contains a list of concepts that are defined and used throughout the rest of this chapter.

We first consider the SP[A] fragment, and thus the REVENG^{+}(SP[A]) decision problem, the REVENG$^{\pm}$(SP[A]) problem, and the REVENG$^{\mathsf{E}}$(SP[A]) problem. Given an input (D, Ω) to REVENG$^{\mathsf{E}}$(SP[A]), the candidate query will be the set of all triple patterns which are true in D over all the positive examples in Ω. Intuitively, this corresponds to the query which is the conjunction of all restrictions that are satisfied by the positive examples.

We now construct this canonical query more precisely. Given an RDF graph D and a mapping μ, define the *atomic type of μ in D*, denoted by atype(D, μ), as

$$\text{atype}(D,\mu) = \{t \in (\mathsf{U} \cup \mathsf{V}) \times (\mathsf{U} \cup \mathsf{V}) \times (\mathsf{U} \cup \mathsf{V} \cup \mathsf{L}) \mid \mu(t) \in D\} \setminus \mathsf{U}^3,$$

Essentially, the atomic type is the set of triple patterns that are true in D under μ. We now generalise this notion to a set of mappings Ω. However, for this fragment of SPARQL we restrict to sets of mappings Ω which are *homogeneous*, that is, for every pair of mappings $\mu, \nu \in \Omega$ it is the case that $\text{dom}(\mu) = \text{dom}(\nu)$. For a homogeneous set of mappings Ω, define the *atomic type of* Ω in D, denoted atype(D, Ω), as $\bigcap_{\mu \in \Omega} \text{atype}(D, \mu)$. Abusing notation, we identify the set atype(D, Ω) and the **AND**-combination of its triple patterns (i.e. an SP[A] pattern).

For the fragment SP[A], the atomic type as constructed above is precisely the desired candidate query. The following lemma presents the crucial property of the candidate query:

Lemma 3.2. *Given an RDF graph D and sets of mappings Ω and $\bar{\Omega}$ the following holds:*

1. *if* $(D,\Omega) \in \textsc{RevEng}^{+}(\text{SP}[\mathsf{A}])$ *then* $\Omega \subseteq [\![\text{atype}(D,\Omega)]\!]_D$*,*
2. *if* $(D,\Omega,\bar{\Omega}) \in \textsc{RevEng}^{\pm}(\text{SP}[\mathsf{A}])$ *then we have both* $\Omega \subseteq [\![\text{atype}(D,\Omega)]\!]_D$ *and* $\bar{\Omega} \cap [\![\text{atype}(D,\bar{\Omega})]\!]_D = \emptyset$*,*
3. *if* $(D,\Omega) \in \textsc{RevEng}^{\mathsf{E}}(\text{SP}[\mathsf{A}])$ *then* $\Omega = [\![\text{atype}(D,\Omega)]\!]_D$*.*

Proof. We proceed item by item.

Item 1. Consider an input (D,Ω) and assume that there exists a query $P \in \text{SP}[\mathsf{A}]$ such that $\Omega \subseteq [\![P]\!]_D$, and note that P may be interpreted as a set of triple patterns. For every mapping $\mu \in \Omega$, $\mu \in [\![P]\!]_D$ holds, whereby for every triple pattern t in P we have $\mu(t) \in D$. Equivalently, for every $t \in P$ it is the case that $\mu(t) \in D$ for every $\mu \in \Omega$, which implies $t \in \text{atype}(D,\Omega)$, by definition. Therefore, P is a subset of atype(D,Ω), and thus atype$(D,\Omega) \neq \emptyset$ (note also that $\text{var}(P) = \text{var}(\text{atype}(D,\Omega)) = \text{dom}(\Omega)$). For any mapping $\mu \in \Omega$, $\mu(t) \in D$ for every $t \in \text{atype}(D,\Omega)$ by construction, whereby $\mu \in [\![\text{atype}(D,\Omega)]\!]_D$. Thus, $\Omega \subseteq [\![\text{atype}(D,\Omega)]\!]_D$.

Item 2. Consider an input $(D,\Omega,\bar{\Omega})$ to the $\textsc{RevEng}^{\pm}(\text{SP}[\mathsf{A}])$ decision problem, and assume there exists a query $P \in \text{SP}[\mathsf{A}]$ such that $\Omega \subseteq [\![P]\!]_D$ and $\bar{\Omega} \cap [\![P]\!]_D = \emptyset$. Due to item 1, we have that $\Omega \subseteq [\![\text{atype}(D,\Omega)]\!]_D$; furthermore, we have that $\text{var}(P) = \text{dom}(\Omega) = \text{var}(\text{atype}(D,\Omega))$ and that P is a subset of atype(D,Ω) (when interpreted as sets of triple patterns). Now consider a mapping $\bar{\mu} \in \bar{\Omega}$, and note that if $\text{dom}(\bar{\mu}) \neq \text{dom}(\Omega)$ then $\bar{\mu} \notin [\![\text{atype}(D,\Omega)]\!]_D$; if $\text{dom}(\bar{\mu}) = \text{dom}(\Omega)$ then as $\bar{\mu} \notin [\![P]\!]_D$ then there must exist a triple t in P such that $\bar{\mu}(t) \notin D$. However, as P is a subset of atype(D,Ω) we have that t is in atype(D,Ω) as well, whereby $\bar{\mu} \notin [\![\text{atype}(D,\Omega)]\!]_D$. As the mapping $\bar{\mu}$ was arbitrary, we conclude that $\bar{\Omega} \cap [\![\text{atype}(D,\Omega)]\!]_D = \emptyset$.

Item 3. Consider an input (D,Ω) to the $\textsc{RevEng}^{\mathsf{E}}(\text{SP}[\mathsf{A}])$ decision problem, and assume that there exists a query $P \in \text{SP}[\mathsf{A}]$ such that $\Omega = [\![P]\!]_D$. Due to item 1, we have that $\Omega \subseteq [\![\text{atype}(D,\Omega)]\!]_D$; furthermore, $\text{dom}(\Omega) = \text{var}(P) = \text{var}(\text{atype}(D,\Omega))$ and P is a subset of atype(D,Ω). We now claim that $[\![\text{atype}(D,\Omega)]\!]_D \subseteq \Omega$. For this, consider an mapping $\nu \in [\![\text{atype}(D,\Omega)]\!]_D$, whereby $\text{dom}(\nu) = \text{dom}(\Omega)$ and for every triple pattern t in atype(D,Ω) we have that $\nu(t) \in D$. In particular, as P is a subset of $\text{atype}(D, \Omega)$ we have that for every t in P it is the case that $\nu(t) \in D$, and thus $\nu \in [\![P]\!]_D = \Omega$. We therefore conclude that $\Omega = [\![\text{atype}(D, \Omega)]\!]_D$. $\square$

Example 3.4. *Let $\Omega = \{\mu_1, \mu_2\}$ with $\mu_1 = [?\texttt{X} \mapsto \texttt{a}]$ and $\mu_2 = [?\texttt{X} \mapsto \texttt{b}]$, and let $D = \{(\texttt{a}, \texttt{type}, \texttt{Person}), (\texttt{b}, \texttt{type}, \texttt{Person}), (\texttt{a}, \texttt{field}, \texttt{CS}), (\texttt{b}, \texttt{field}, \texttt{CS})\}$. Then $\text{atype}(D, \Omega) = \{(?\texttt{X}, \texttt{type}, \texttt{Person}), (?\texttt{X}, \texttt{field}, \texttt{CS})\}$, as $t_1 = (?\texttt{X}, \texttt{type}, \texttt{Person})$ and $t_2 = (?\texttt{X}, \texttt{field}, \texttt{CS})$ are the only triple patterns t for which both $\mu_1(t) \in D$ and $\mu_2(t) \in D$ hold. Therefore, the corresponding candidate query $P = (?\texttt{X}, \texttt{type}, \texttt{Person})$ **AND** $(?\texttt{X}, \texttt{field}, \texttt{CS})$ realises the input pair. This is not the only possible realizer though; for example, the query $Q = (?\texttt{X}, \texttt{field}, \texttt{CS})$ also has the property that $\Omega \subseteq [\![P]\!]_D$.*

Lemma 3.2 leads to the following algorithm template for reverse-engineering in the SP[A] case: first build the atomic type of Ω—which can be done in polynomial time in this case—and then check if it works. Combining this with our results on the verification problem, we obtain the following complexity upper bounds:

Proposition 3.10. *The decision problems* $\text{RevEng}^+(\text{SP[A]})$ *and* $\text{RevEng}^\pm(\text{SP[A]})$ *are in* **PTIME***, while* $\text{RevEng}^{\mathsf{E}}(\text{SP[A]})$ *is in* **coNP**.

Proof. For $\text{RevEng}^+(\text{SP[A]})$, we provide a deterministic polynomial-time algorithm which decides it. Given an input (D, Ω), if Ω is not homogeneous, then return False. Now, build $\text{atype}(D, \Omega)$; this can be done in polynomial time by iterating through all the possible triple patterns t which mention variables from Ω and constants (i.e. URIs or literals) from D, and testing whether $\mu(t) \in D$ holds for all $\mu \in \Omega$. Note that if there exists a query $P \in \text{SP[A]}$ such that $\Omega \subseteq [\![P]\!]_D$ then it must be the case that P is a subset of $\text{atype}(D, \Omega)$, whereby if $\text{atype}(D, \Omega)$ thus built is such that $\text{atype}(D, \Omega)$ is empty, or $\text{var}(\text{atype}(D, \Omega)) \subsetneq \text{dom}(\Omega)$, then we return False. If, on the other hand, $\text{atype}(D, \Omega)$ is such that $\text{dom}(\Omega) = \text{var}(\text{atype}(D, \Omega))$, then by construction we have that for every $\mu \in \Omega$ it is the case that $\mu \in [\![\text{atype}(D, \Omega)]\!]_D$ whereby we return True.

For $\text{RevEng}^\pm(\text{SP[A]})$ we also provide a deterministic polynomial-time algorithm which decides it. Given an input $(D, \Omega, \bar{\Omega})$, if Ω is not homogeneous, then return False. Now build $\text{atype}(D, \Omega)$ as before; if $\text{var}(\text{atype}(D, \Omega)) \subsetneq \text{dom}(\Omega)$ then return False. Now, assume that $\text{var}(\text{atype}(D, \Omega)) = \text{dom}(\Omega)$. We may now execute the deterministic polynomial-time algorithm provided for the $\text{Verify}^\pm(\text{SP[A]})$ decision problem, with $(D, \Omega, \text{atype}(D, \Omega))$ as input. If the previous accepts, then return True, as $\text{atype}(D, \Omega)$ is a witness. If, on the other hand, the previous subroutine rejects, then there must exist a mapping $\bar{\mu} \in \bar{\Omega}$ such that $\bar{\mu} \in [\![\text{atype}(D, \Omega)]\!]_D$ (note that $\Omega \subseteq [\![\text{atype}(D, \Omega)]\!]_D$ holds by construction). However, if there exists a query

$P \in \mathrm{SP}[\mathsf{A}]$ such that $\Omega \subseteq [\![P]\!]_D$ and $\bar{\Omega} \cap [\![P]\!]_D = \emptyset$ then it is the case that P is a subset of $\mathrm{atype}(D, \Omega)$, whereby if $\bar{\mu} \in [\![\mathrm{atype}(D, \Omega)]\!]_D$ then $\bar{\mu} \in [\![P]\!]_D$ as well, which is a contradiction. Therefore, if the subroutine rejects we return False.

For $\textsc{RevEng}^{\mathsf{E}}(\mathrm{SP}[\mathsf{A}])$, we provide a non-deterministic polynomial-time algorithm which decides the complement of the problem. Given an input (D, Ω), if Ω is not homogeneous, then accept. Now, we build $\mathrm{atype}(D, \Omega)$; if $\mathrm{var}(\mathrm{atype}(D, \Omega)) \subsetneq \mathrm{dom}(\Omega)$ then accept, as any query $P \in \mathrm{SP}[\mathsf{A}]$ such that $\Omega \subseteq [\![P]\!]_D$ must be a subset of $\mathrm{atype}(D, \Omega)$ and thus does not mention all variables in $\mathrm{dom}(\Omega)$ in this case, which is a contradiction. Now, assume that $\mathrm{dom}(\Omega) = \mathrm{atype}(D, \Omega)$, and execute a non-deterministic polynomial-time algorithm for the complement of the $\textsc{Verify}^{\mathsf{E}}(\mathrm{SP}[\mathsf{A}])$ decision problem (note that the complement of $\textsc{Verify}^{\mathsf{E}}(\mathrm{SP}[\mathsf{A}])$ is in $\mathbf{NP}$). If the previous rejects, then $\mathrm{atype}(D, \Omega)$ is such that $\Omega = [\![\mathrm{atype}(D, \Omega)]\!]_D$ and thus we reject. If the previous subroutine accepts, then $\mathrm{atype}(D, \Omega)$ is such that $\Omega \neq [\![\mathrm{atype}(D, \Omega)]\!]_D$, whereby the only possibility is that $[\![\mathrm{atype}(D, \Omega)]\!]_D \subsetneq \Omega$ (as $\Omega \subseteq [\![\mathrm{atype}(D, \Omega)]\!]_D$ by construction); consider then, the mapping $\nu \in [\![\mathrm{atype}(D, \Omega)]\!]_D$ such that $\nu \notin \Omega$ and note that for any query $P \in \mathrm{SP}[\mathsf{A}]$ such that $\Omega = [\![P]\!]_D$ it is the case that P is a subset of $\mathrm{atype}(D, \Omega)$, whereby $\nu \in [\![P]\!]_D$, which is a contradiction, implying that $(D, \Omega) \notin \textsc{RevEng}^{\mathsf{E}}(\mathrm{SP}[\mathsf{A}])$. Therefore, we accept. □

3.4.3 Reverse engineering with OPTIONAL

Next, we generalise the previous process for the $\mathrm{SP}[\mathsf{AOwd}]$ fragment. To build intuition, first consider, as an example, a query $P = P_1 \,\mathbf{OPT}\, P_2$ where both P_1 and P_2 are in $\mathrm{SP}[\mathsf{A}]$. For every variable $\mathbf{v} \in \mathrm{var}(P)$ there are two possibilities: if $\mathbf{v} \in \mathrm{var}(P_1)$, then for every mapping $\mu \in [\![P]\!]_D$ it will be the case that $\mathbf{v} \in \mathrm{dom}(\mu)$; otherwise, if $\mathbf{v} \in \mathrm{var}(P_2)$ and $\mathbf{v} \notin \mathrm{var}(P_1)$, then there may exist mappings $\mu \in [\![P]\!]_D$ such that $\mathbf{v} \notin \mathrm{dom}(\mu)$. This example illustrates the existence of a *hierarchy of variables*. In fact, for two variables $\mathbf{v}, \mathbf{u}$ such that $\mathbf{v} \in \mathrm{var}(P_1)$ and $\mathbf{u} \in \mathrm{var}(P_2) \setminus \mathrm{var}(P_1)$ it will be the case that for every mapping $\mu \in [\![P]\!]_D$, if $\mathbf{u} \in \mathrm{dom}(\mu)$ then $\mathbf{v} \in \mathrm{dom}(\mu)$ (in this example this statement is trivial, as $\mathbf{x}$ is in the domain of every mapping).

The previous reasoning can now be used to outline the same example, now from a reverse engineering perspective. Consider an RDF graph D and a set of mappings Ω as inputs to $\textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOwd}])$, where Ω can be divided into two subsets Ω_1 and Ω_2 such that $\Omega = \Omega_1 \cup \Omega_2$, for every mapping $\mu \in \Omega_1$ we have $\mathrm{dom}(\mu) = \{?\mathtt{X}, ?\mathtt{Y}\}$, and for every mapping $\nu \in \Omega_2$ we have $\mathrm{dom}(\nu) = \{?\mathtt{X}\}$. In this case, the form of a candidate query P can be determined by observing the variable hierarchy, concluding that $P = P_1 \,\mathbf{OPT}\, P_2$ for some P_1 such that $\mathrm{var}(P_1) = \{?\mathtt{X}\}$ and some P_2 such that

$$\begin{array}{llllll}
\mu_1 = [?\mathtt{X} \mapsto 1, & & &] \\
\mu_2 = [?\mathtt{X} \mapsto 2, & ?\mathtt{Y}_1 \mapsto a & &] \\
\mu_3 = [?\mathtt{X} \mapsto 3, & & ?\mathtt{Y}_2 \mapsto b &] \\
\mu_4 = [?\mathtt{X} \mapsto 4, & & ?\mathtt{Y}_2 \mapsto c, \quad ?\mathtt{Z} \mapsto d &]
\end{array}$$

(a) A non-homogeneous set of mappings Ω.

$$\begin{tikzcd}
 & \{\mu_1,\mu_2,\mu_3,\mu_4\}^{(?\mathtt{X})} & \\
{}^{(?\mathtt{Y}_1)}\{\mu_2\} \arrow[ur] & & \{\mu_3,\mu_4\}^{(?\mathtt{Y}_2)} \arrow[ul] \\
 & & \{\mu_4\}^{(?\mathtt{Z})} \arrow[u]
\end{tikzcd}$$

(b) The structure $\mathcal{C}(\Omega)$. Each node $\Lambda = \mathsf{Cov}_\Omega(\mathbf{v})$ has been marked with a superscript indicating the corresponding variable $\mathbf{v}$.

Figure 3.1: An example of how a tree structure is generated by a set of mappings.

$\text{var}(P_2) \subseteq \{?\mathtt{X}, ?\mathtt{Y}\}$. Crucially, it will be determined that no further **OPT** operators are necessary, i.e. $P_1, P_2 \in \text{SP}[\mathsf{A}]$. The precise triple patterns in P_1 and P_2 will be determined by a construct analogous to the atomic type.

We now formalise the previous intuition, showing that the relation among variables in the set of mappings Ω can be used to determine the form of the candidate query P, and that a generalisation of the atomic type can be used to determine the triple patterns in each subquery of P.

For every variable $\mathbf{v}$ mentioned in Ω define the *coverage* of $\mathbf{v}$ in Ω, denoted $\mathsf{Cov}_\Omega(\mathbf{v})$ as the set $\{\mu \in \Omega \mid \mathbf{v} \in \text{dom}(\mu)\}$. Define the *structure* of Ω as the set $\mathcal{C}(\Omega) = \{\mathsf{Cov}_\Omega(\mathbf{v}) \mid \mathbf{v} \in \text{dom}(\Omega)\}$, containing the coverage of each variable in Ω. The subset relation naturally defines a partial order on the structure $\mathcal{C}(\Omega)$.

Example 3.5. *Consider the set of mappings Ω in Figure 3.1a. The corresponding structure $\mathcal{C}(\Omega)$ is shown in Figure 3.1b, where arrows represents the minimal proper superset relation. The topmost node corresponds to the coverage of variable $?\mathtt{X}$, which is present in all mappings. The two nodes at the second level correspond to the coverages of $?\mathtt{Y}_1$ and $?\mathtt{Y}_2$, respectively, while the lone node at the third level represents the coverage of $?\mathtt{Z}$.*

Intuitively, the structure of Ω defined above restricts the possible **OPT** structures of a candidate query. We say that $\mathcal{C}(\Omega)$ (and, by extension, Ω) is *tree-like* if and only if for every set $\Lambda \in \mathcal{C}(\Omega)$ there is at most one minimal proper superset (i.e. parent) of Λ in $\mathcal{C}(\Omega)$. Note that since Ω is *consistent* (see page 27), there exists only one set in $\mathcal{C}(\Omega)$ without proper supersets and this set is Ω itself.

Example 3.6. *While the set of mappings in Figure 3.1 is tree-like, consider replacing mapping* μ_4 *by* $\mu_4' = [?\mathtt{X} \mapsto 4, ?\mathtt{Y}_1 \mapsto e, ?\mathtt{Y}_2 \mapsto c, ?\mathtt{Z} \mapsto d]$*. In this case, the coverage of* $?\mathtt{Y}_1$ *changes to* $\{\mu_2, \mu_4'\}$ *and the node* $\{\mu_4'\}$ *becomes a subset of both* $\{\mu_2, \mu_4'\}$ *and* $\{\mu_3, \mu_4'\}$*. The resulting set of mappings* $\{\mu_1, \mu_2, \mu_3, \mu_4'\}$ *is not tree-like, as* $\{\mu_4'\}$ *has two minimal proper supersets (i.e. parents):* $\{\mu_2, \mu_4'\}$ *and* $\{\mu_3, \mu_4'\}$*.*

For tree-like sets of mappings we can give a detailed process for constructing a candidate query, and we turn to this now. Given a set of mappings Ω and a set of variables S, let Ω_S consist of all $\mu \in \Omega$ such that $S \subseteq \mathrm{dom}(\mu)$. Finally, for a node Λ in $\mathcal{C}(\Omega)$, let the *scope of* Λ in Ω, denoted $\mathsf{scope}_\Omega(\Lambda)$, be the set of variables $\mathbf{v}$ with $\Lambda \subseteq \mathsf{Cov}_\Omega(\mathbf{v})$ (note that this includes all variables corresponding to nodes along the path towards the root).

We can now define a canonical query for tree-like sets of mappings. To do this we will recursively define a pattern $P_{\mathsf{can}}(\Lambda, D, \Omega)$ for each Λ in $\mathcal{C}(\Omega)$. If Λ has k maximal proper subsets (i.e. children) $\Lambda_1, \ldots, \Lambda_k$, then $P_{\mathsf{can}}(\Lambda, D, \Omega)$ is the graph pattern

$$\begin{aligned}(\cdots((\mathrm{atype}(D, \Omega_{\mathsf{scope}_\Omega(\Lambda)})\ \mathbf{OPT}\ P_{\mathsf{can}}(\Lambda_1, D, \Omega))\\ \mathbf{OPT}\ P_{\mathsf{can}}(\Lambda_2, D, \Omega))\ \cdots\ \mathbf{OPT}\ P_{\mathsf{can}}(\Lambda_k, D, \Omega)).\end{aligned} \tag{3.1}$$

Note that if Λ has no children then $P_{\mathsf{can}}(\Lambda, D, \Omega)$ is just $\mathrm{atype}(D, \Omega_{\mathsf{scope}_\Omega(\Lambda)})$. Finally, we define $P_{\mathsf{can}}(D, \Omega)$ to be $P_{\mathsf{can}}(\Omega, D, \Omega)$.

In the construction above the order of the children $\Lambda_1, \ldots, \Lambda_k$ is arbitrary, so the canonical pattern is not unique. However, all of them are equivalent, since $(P_1\ \mathbf{OPT}\ P_2)\ \mathbf{OPT}\ P_3$ is equivalent to $(P_1\ \mathbf{OPT}\ P_3)\ \mathbf{OPT}\ P_2$ for all queries which are well-designed. This blurred order of the children amounts to considering canonical queries as *pattern trees* [70], whose tree structure is the same as that of $\mathcal{C}(\Omega)$.

For tree-like sets of mappings we can formulate a generalisation of Lemma 3.2, for input instances where Ω is restricted to being tree-like. However, we will first turn to a treatment of the tree-like restriction from the point of view of *pattern trees*. These definitions will be used heavily in proofs.

3.4.4 Reverse engineering with pattern trees

We introduce a simple yet key notion of *compatibility*. Given a set of mappings Ω and a pattern tree $P \in \mathrm{SP}[\mathsf{AOFwd}]$ such that $\mathrm{var}(P) = \mathrm{dom}(\Omega)$, we say P is *compatible* with Ω if and only if for every pair of variables $\mathbf{v}, \mathbf{u} \in \text{var}(P)$, if $\mathsf{top}_P(\mathbf{v})$ is an ancestor of $\mathsf{top}_P(\mathbf{u})$ in P, then $\mathsf{Cov}_\Omega(\mathbf{u}) \subseteq \mathsf{Cov}_\Omega(\mathbf{v})$.

Recall that when dealing with a tree-like set of mappings Ω, $\mathcal{C}(\Omega)$ is a tree, and for each node (i.e. coverage) $\Lambda \in \mathcal{C}$ we associate the set of variables $\mathsf{VarsOf}(\Lambda) = \{\mathbf{v} \in \text{dom}(\Omega) \mid \mathsf{Cov}_\Omega(\mathbf{v}) = \Lambda\}$. With the previous, we define that a pattern tree $P = (V_P, E_P, r_P, \lambda_P, \delta_P) \in \text{SP}[\mathsf{AOFwd}]$ *has the same structure as* Ω if and only if there is a bijective function $f : V_P \to \mathcal{C}(\Omega)$ which preserves the tree structure, and furthermore for every node $n \in V_P$ it is the case that $\{\mathbf{v} \in \text{var}(P) \mid n = \mathsf{top}_P(\mathbf{v})\} = \mathsf{VarsOf}(f(n))$; intuitively, P and $\mathcal{C}(\Omega)$ have the same tree structure and associate to each pair of corresponding nodes the same variables.

Finally, note that for an RDF graph D and a tree-like set of mappings Ω, the corresponding canonical query $P_{\mathsf{can}}(D, \Omega)$ is a pattern tree which has the same structure of Ω and such that for every node $n \in V_P$ it is the case that $\lambda_P(n) = \text{atype}(D, \Omega_{\mathsf{scope}_P(n)})$, and $\delta_P(n) = \mathsf{True}$ is a trivial built-in condition. Also, the generalisation for the fragment $\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}]$, $P^{=,\neq}_{\mathsf{can}}(D, \Omega)$ adds to each node of P_{can} the built-in condition $\delta_P(n) = \text{atype}_{=,\neq}(D, \Omega_{\mathsf{scope}_P(n)})$, where $\text{atype}_{=,\neq}$ is defined analogously to atype, including all equalities and inequalities between variables and constants that hold.

Lemma 3.3. *Given a set of mappings Ω and a pattern tree $P \in \text{SP}[\mathsf{AOFwd}]$ such that $\text{var}(P) = \text{dom}(\Omega)$, if $\Omega \subseteq [\![P]\!]_D$ then P is compatible with Ω.*

Proof. For the sake of contradiction, assume that P is not compatible with Ω, whereby there exist a pair of variables $\mathbf{v}, \mathbf{u} \in \text{var}(P)$ such that $\mathsf{top}_P(\mathbf{v})$ is an ancestor of $\mathsf{top}_P(\mathbf{u})$ but $\mathsf{Cov}_\Omega(\mathbf{u}) \not\subseteq \mathsf{Cov}_\Omega(\mathbf{u})$, whereby there is a mapping $\mu \in \Omega$ such that $\mathbf{u} \in \text{dom}(\mu)$ but $\mathbf{v} \notin \text{dom}(\mu)$.

As $\Omega \subseteq [\![P]\!]_D$ we have that $\mu \in [\![P]\!]_D$, whereby there exists a reduction $P_\mu \trianglelefteq P$ such that $\text{dom}(\mu) = \text{var}(P_\mu)$. Now, as $\mathbf{u} \in \text{dom}(\mu)$ we must have that the node $\mathsf{top}_P(\mathbf{u})$ is included in P_μ. Furthermore, as $\mathsf{top}_P(\mathbf{v})$ is an ancestor of $\mathsf{top}_P(\mathbf{u})$ in P, we have that $\mathsf{top}_P(\mathbf{v})$ is also included in P_μ, and thus $\mathbf{v} \in \text{dom}(\mu)$, which is a contradiction. We therefore conclude that P is compatible with Ω. $\square$

Given a set of mappings Ω and a set of variables $S \subseteq \text{dom}(\Omega)$, we say S is *closed* in Ω if and only if: for every pair of variables $\mathbf{v}, \mathbf{u} \in \text{dom}(\Omega)$, if $\mathsf{Cov}_\Omega(\mathbf{u}) \subseteq \mathsf{Cov}_\Omega(\mathbf{v})$ and $\mathbf{u} \in S$ then $\mathbf{v} \in S$.

Lemma 3.4. *Consider a set of mappings Ω, a pattern tree $P \in \text{SP}[\mathsf{AOFwd}]$, and a set of variables $S \subseteq \text{dom}(\Omega)$, such that: (i) $\text{var}(P) \subseteq \text{dom}(\Omega)$, (ii) P is compatible with Ω, (iii) S is closed in Ω. Then there exists a pattern tree $P' \trianglelefteq P$ such that $\text{var}(P') = S$.*

Proof. We obtain P' in the following way: Initially, let $Q \leftarrow P$. If $\mathrm{var}(Q) = S$ then stop; otherwise, note that $S \subsetneq \mathrm{var}(Q)$ and let $\mathbf{v}$ be a variable in $\mathrm{var}(Q) \setminus S$ and let n be the node in Q such that $n = \mathsf{top}_Q(\mathbf{v})$. Note that for every variable $\mathbf{u} \in \mathrm{var}(Q)$ such that $\mathsf{top}_Q(\mathbf{v})$ is an ancestor of $\mathsf{top}_Q(\mathbf{u})$ it is the case that $\mathsf{Cov}_\Omega(\mathbf{u}) \subseteq \mathsf{Cov}_\Omega(\mathbf{v})$ (as P is compatible with Ω), whereby $\mathbf{u} \notin S$ ($\mathbf{u} \in S$ contradicts the fact that S is closed in Ω). Therefore, we remove the subtree of Q whose root is n (that is, update $V_Q \leftarrow V_Q \setminus \{m \in V_Q \mid n \text{ is an ancestor of } m \text{ in } Q\}$ and remove the appropriate edges from E_Q as well). We iterate in this fashion until $\mathrm{var}(Q) = S$ and set $P' \leftarrow Q$. □

With the previous, note that for a set of mappings Ω it is the case that for any $\mu \in \Omega$, $\mathrm{dom}(\mu)$ is closed in Ω, whereby for any $P \in \mathrm{SP}[\mathsf{AOFwd}]$ which is compatible with Ω (e.g. potential realizers) we will be able to find a reduction $P_\mu \trianglelefteq P$ such that $\mathrm{var}(P_\mu) = \mathrm{dom}(\mu)$. This fact will be used heavily in the following proofs.

3.4.5 Reverse engineering upper bounds

We begin by presenting a series of lemmas that assert the desirable properties of our canonical (candidate) queries.

Lemma 3.5. *Given an RDF graph D, tree-like set of mappings Ω,*

1. *if* $(D, \Omega) \in \textsc{RevEng}^{+}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$ *then* $\Omega \subseteq [\![P_{\mathsf{can}}(D, \Omega)]\!]_D$,
2. *if* $(D, \Omega) \in \textsc{RevEng}^{\mathsf{E}}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$ *then* $\Omega = [\![P_{\mathsf{can}}(D, \Omega)]\!]_D$.

Proof. We proceed item by item.

Item 1. Assume that $(D, \Omega) \in \textsc{RevEng}^{+}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$, whereby there exists a pattern tree $Q \in \mathrm{SP}[\mathsf{AOwd}]$ such that $\Omega \subseteq [\![Q]\!]_D$. Without loss of generality, we may assume that $\mathrm{var}(Q) = \mathrm{dom}(\Omega)$ and that Q is in productive form. By Lemma 3.3, then, Q is compatible with Ω. For the following, let $P = P_{\mathsf{can}}(D, \Omega)$, and note that $P = (V_P, E_P, r_P, \lambda_P)$, where $V_P = \mathcal{C}(\Omega)$ and $E_P = \{(\Lambda, \Lambda') \mid \Lambda' \text{ is a maximal proper superset of } \Lambda\}$ (i.e. P has the same structure as Ω).

Now consider a mapping $\mu \in \Omega$. As $\mu \in [\![Q]\!]_D$, there exists a pattern tree $Q_\mu \trianglelefteq Q$ such that $\mathrm{var}(Q_\mu) = \mathrm{dom}(\mu)$ and $\mu \in [\![\mathrm{andify}(Q_\mu)]\!]_D$ (i.e. $\mu \in \mathsf{partials}(D, Q)$) and μ is maximal in $\mathsf{partials}(D, Q)$. We must now show that $\mu \in [\![P]\!]_D$.

Now build the pattern tree $P_\mu \trianglelefteq P$ in the following way: for every node $\Lambda \in V_P$, $\Lambda \in V_{P_\mu}$ if and only if $\mu \in \Lambda$ (notice that this, in fact, forms a reduction of P), whereby we have that $\mathrm{dom}(\mu) = \mathrm{var}(P_\mu)$. Now, by construction of P, each node Λ in P_μ will only contain triple patterns which are true for every mapping $\nu \in \Omega$

which mentions variables from $\mathsf{scope}_{P_\mu}(\Lambda)$, whereby we have $\mu \in [\![\text{andify}(P_\mu)]\!]_D$ by construction.

We must now show that μ is maximal in $\mathsf{partials}(D,P)$. For the sake of contradiction, assume that there exists a mapping $\nu \in \mathsf{partials}(D,P)$ such that $\mu \sqsubset \nu$, whereby there exists a pattern tree $P_\nu \trianglelefteq P$ such that $\text{var}(P_\nu) = \text{dom}(\nu)$ and $\nu \in [\![\text{andify}(P_\nu)]\!]_D$ (also note that $P_\mu \trianglelefteq P_\nu \trianglelefteq P$ and $P_\mu \neq P_\nu$). Now, as P has the same structure as Ω, we have that $\text{var}(P_\nu)$ is closed in Ω, whereby there is a pattern tree $Q_\nu \trianglelefteq Q$ such that $\text{var}(Q_\nu) = \text{dom}(\nu)$.

We now claim that $\nu \in [\![\text{andify}(Q_\nu)]\!]_D$. For this, consider a node $n \in V_{Q_\nu}$ and a triple pattern $t \in \lambda_{Q_\nu}(n)$. We claim we can find the triple pattern t in P_ν. For this, consider a variable $?\mathtt{X} \in \text{var}(Q_\nu)$ such that $n = \mathsf{top}_Q(?\mathtt{X})$ (this variable must exist, as Q is in productive form). As $\text{var}(Q_\nu) = \text{var}(P_\nu)$ there is a node $m \in V(P_\nu)$ such that $m = \mathsf{top}_P(?\mathtt{X})$. We claim that $\mathsf{scope}_Q(n) \subseteq \mathsf{scope}_P(m)$ and we show this by contradiction next. For the sake of contradiction, assume that there is a variable $?\mathtt{Y} \in \mathsf{scope}_Q(n)$ such that $?\mathtt{Y} \notin \mathsf{scope}_P(m)$. Then it is the case that $\mathsf{top}_Q(?\mathtt{Y})$ is an ancestor of $\mathsf{top}_Q(?\mathtt{X})$ in Q, but the same does not hold in P. As P has the same structure as Ω this contradicts the fact that Q is compatible with Ω. Therefore, we conclude that $\mathsf{scope}_Q(n) \subseteq \mathsf{scope}_P(m)$. Now, as $\Omega \subseteq [\![Q]\!]_D$ we have that for every mapping $\sigma \in \Omega$ such that $\mathsf{scope}_Q(n) \subseteq \text{dom}(\sigma)$, it will be the case that $\sigma(t) \in D$, and thus, for every mapping $\sigma \in \Omega$ such that $\mathsf{scope}_P(m) \subseteq \text{dom}(\sigma)$ it will also be the case that $\sigma(t) \in D$. This implies that $t \in \text{atype}(D, \Omega|_{\mathsf{scope}_P(m)})$, whereby, by construction, $t \in \lambda_P(m)$. Finally, as $\nu \in [\![\text{andify}(P_\nu)]\!]_D$ we have that $\nu(t) \in D$. As both the node n and the triple pattern t are arbitrary, we have that $\nu \in [\![\text{andify}(Q_\nu)]\!]_D$.

The previous discussion implies that $\nu \in \mathsf{partials}(D,Q)$, and as $\mu \sqsubset \nu$ this contradicts the fact that μ is maximal in $\mathsf{partials}(D,Q)$. Therefore, μ is maximal in $\mathsf{partials}(D,P)$. Thus, $\mu \in [\![P]\!]_D$, and we conclude that $\Omega \subseteq [\![P]\!]_D$.

Item 2. Now assume that $(D,\Omega) \in \textsc{RevEng}^{\mathsf{E}}(\text{SP}[\mathsf{AOwd}])$, whereby there exists a pattern tree $Q \in \text{SP}[\mathsf{AOwd}]$ such that $\Omega = [\![Q]\!]_D$. In particular, $(D,\Omega) \in \textsc{RevEng}^{+}(\text{SP}[\mathsf{AOwd}])$, whereby due to item 1 we have that $\Omega \subseteq [\![P_{\mathsf{can}}(D,\Omega)]\!]_D$, and therefore we need only prove that $[\![P_{\mathsf{can}}(D,\Omega)]\!]_D \subseteq \Omega$. Again, let $P = P_{\mathsf{can}}(D,\Omega)$.

Consider a mapping $\mu \in [\![P]\!]_D$, whereby there exists $P_\mu \trianglelefteq P$ such that $\text{var}(P_\mu) = \text{dom}(\mu)$, $\mu \in [\![\text{andify}(P_\mu)]\!]_D$, and μ is maximal in $\mathsf{partials}(D,P)$. We must show that $\mu \in \Omega$, or equivalently, that $\mu \in [\![Q]\!]_D$. For this, notice that as P has the same structure as Ω, $\text{var}(P_\mu)$ is closed in Ω, and as Q is compatible with Ω there exists $Q_\mu \trianglelefteq Q$ such that $\text{var}(Q_\mu) = \text{dom}(\mu)$. Now consider a node n in Q_μ, a triple pattern t in n, and a variable $?\mathtt{X}$ such that $n = \mathsf{top}_{Q_\mu}(?\mathtt{X})$. As $\Omega \subseteq [\![Q]\!]_D$ we have that for every

$\nu \in \Omega$ such that $\mathsf{scope}_{Q_\mu}(n) \subseteq \mathrm{dom}(\nu)$ it must be the case that $\nu(t) \in D$, whereby $t \in \mathrm{atype}(D, \Omega_{\mathsf{scope}_{Q_\mu}(n)})$. Now, consider the node m in P_μ such that $m = \mathsf{top}_{P_\mu}(?\mathtt{X})$ and note that $\mathsf{scope}_{Q_\mu}(n) \subseteq \mathsf{scope}_{P_\mu}(m)$, whereby $t \in \mathrm{atype}(D, \Omega_{\mathsf{scope}_{P_\mu}(m)})$, and thus t is mentioned in m, by construction of P. Now, as $\mu \in [\![\mathrm{andify}(P_\mu)]\!]_D$ we have that $\mu(t) \in D$ and thus we conclude that $\mu \in [\![\mathrm{andify}(Q_\mu)]\!]_D$ (as both t and n we arbitrarily chosen).

We must now show that μ is maximal in $\mathsf{partials}(D, Q)$. For the sake of contradiction, assume that μ is not maximal, whereby there exists a mapping $\nu \in \mathsf{partials}(D, Q)$ which *is* maximal, and such that $\mu \sqsubset \nu$. This implies that $\nu \in [\![Q]\!]_D = \Omega$, but as $\Omega \subseteq [\![P]\!]_D$ we have $\nu \in [\![P]\!]_D$, which implies that ν is maximal in $\mathsf{partials}(D, P)$, contradicting the fact that $\mu \in [\![P]\!]_D$. Therefore, we have that μ is maximal in $\mathsf{partials}(D, Q)$, whereby $\mu \in [\![Q]\!]_D$. We have thus shown that $[\![P]\!]_D \subseteq [\![Q]\!]_D = \Omega$, whereby $\Omega = [\![P]\!]_D$. $\square$

The notion of canonical query for tree-like sets of mappings can be straightforwardly adapted to the case of SP[$\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}$], considering the generalisation $\mathrm{atype}_{=,\neq}(D, \Omega)$ of $\mathrm{atype}(D, \Omega)$ (which, recall, contains all equalities and inequalities on the variables and URIs that are true in D. When seen as a query, this set is the **AND**-combination of its triple patterns, filtered by the conjunction of all its equalities and inequalities that mention only variables in the triple patterns. The analog of Lemma 3.5 holds for SP[$\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}$] (see Lemma 3.6) and similarly for SP[AOFwd] (see Lemma 3.7).

Lemma 3.6. *Given an RDF graph D, tree-like set of mappings Ω,*

1. *if* $(D, \Omega) \in \textsc{RevEng}^{+}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$ *then* $\Omega \subseteq [\![P^{=,\neq}_{\mathsf{can}}(D, \Omega)]\!]_D$,
2. *if* $(D, \Omega) \in \textsc{RevEng}^{\mathsf{E}}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$ *then* $\Omega = [\![P^{=,\neq}_{\mathsf{can}}(D, \Omega)]\!]_D$.

Proof. **Item 1.** Assume that $(D, \Omega) \in \textsc{RevEng}^{+}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$, whereby there exists a pattern tree $Q \in \mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}]$ such that $\Omega \subseteq [\![Q]\!]_D$. Without loss of generality, we may assume that $\mathrm{var}(Q) = \mathrm{dom}(\Omega)$ and that Q is in productive form. By Lemma 3.3, then, Q is compatible with Ω. For the following, let $P = P^{=,\neq}_{\mathsf{can}}(D, \Omega)$, and note that $P = (V_P, E_P, r_P, \lambda_P, \delta_P)$, where $V_P = \mathcal{C}(\Omega)$ and $E_P = \{(\Lambda, \Lambda') \mid \Lambda'$ is a maximal proper superset of $\Lambda\}$ (in other words, P has the same structure as Ω).

Now consider a mapping $\mu \in \Omega$. As $\mu \in [\![Q]\!]_D$, there exists a pattern tree $Q_\mu \trianglelefteq Q$ such that $\mathrm{var}(Q_\mu) = \mathrm{dom}(\mu)$ and $\mu \in [\![\mathrm{andify}(Q_\mu)]\!]_D$ (i.e. $\mu \in \mathsf{partials}(D, Q)$) and μ is maximal in $\mathsf{partials}(D, Q)$. We must now show that $\mu \in [\![P]\!]_D$.

Now build the pattern tree $P_\mu \trianglelefteq P$ in the following way: for every node $\Lambda \in V_P$, $\Lambda \in V_{P_\mu}$ if and only if $\mu \in \Lambda$ (notice that this, in fact, forms a reduction of P), whereby we have that $\text{dom}(\mu) = \text{var}(P_\mu)$. Now, by construction of P, each node Λ in P_μ will only contain triple patterns which are true for every mapping $\nu \in \Omega$ which mentions variables from $\mathsf{scope}_{P_\mu}(\Lambda)$; on the other hand, each node Λ will also only contain filter equalities and inequalities in $\delta_P(\Lambda)$ which are true for every mapping $\nu \in \Omega$ such that $\mathsf{scope}_P(\Lambda) \subseteq \text{dom}(\nu)$. Therefore, we have $\mu \in [\![\text{andify}(P_\mu)]\!]_D$ by construction.

We must now show that μ is maximal in $\mathsf{partials}(D, P)$. For the sake of contradiction, assume that there exists a mapping $\nu \in \mathsf{partials}(D, P)$ such that $\mu \sqsubset \nu$, whereby there exists a pattern tree $P_\nu \trianglelefteq P$ such that $\text{var}(P_\nu) = \text{dom}(\nu)$ and $\nu \in [\![\text{andify}(P_\nu)]\!]_D$ (also note that $P_\mu \trianglelefteq P_\nu \trianglelefteq P$ and $P_\mu \neq P_\nu$). Now, as P has the same structure as Ω, we have that $\text{var}(P_\nu)$ is closed in Ω, whereby there is a pattern tree $Q_\nu \trianglelefteq Q$ such that $\text{var}(Q_\nu) = \text{dom}(\nu)$.

We now claim that $\nu \in [\![\text{andify}(Q_\nu)]\!]_D$. For this, consider a node $n \in V(Q_\nu)$, a triple pattern $t \in \lambda_Q(n)$, and a filter (in)equality $f \in \delta_Q(n)$. We wish to find the triple pattern t in P. For this, consider a variable $?\texttt{X} \in \text{var}(Q_\nu)$ such that $n = \mathsf{top}_Q(?\texttt{X})$ (this variable must exist, as Q is in productive form). As $\text{var}(Q_\nu) = \text{var}(P_\nu)$ there is a node $m \in V(P_\nu)$ such that $m = \mathsf{top}_P(?\texttt{X})$. We claim that $\mathsf{scope}_Q(n) \subseteq \mathsf{scope}_P(m)$. For the sake of contradiction, assume that there is a variable $?\texttt{Y} \in \mathsf{scope}_Q(n)$ such that $?\texttt{Y} \notin \mathsf{scope}_P(m)$. Then it is the case that $\mathsf{top}_Q(?\texttt{Y})$ is an ancestor of $\mathsf{top}_Q(?\texttt{X})$ in Q, but the same does not hold in P. As P has the same structure as Ω this contradicts the fact that Q is compatible with Ω. Therefore, we conclude that $\mathsf{scope}_Q(n) \subseteq \mathsf{scope}_P(m)$. Now, as $\Omega \subseteq [\![Q]\!]_D$ we have that for every mapping $\sigma \in \Omega$ such that $\mathsf{scope}_Q(n) \subseteq \text{dom}(\sigma)$, it will be the case that $\sigma(t) \in D$, and thus, for every mapping $\sigma \in \Omega$ such that $\mathsf{scope}_P(m) \subseteq \text{dom}(\sigma)$ it will also be the case that $\sigma(t) \in D$. This implies that $t \in \text{atype}(D, \Omega|_{\mathsf{scope}_P(m)})$, whereby, by construction, $t \in \lambda_P(m)$. Finally, as $\nu \in [\![\text{andify}(P_\nu)]\!]_D$ we have that $\nu(t) \in D$. As both the node n and the triple pattern t are arbitrary, we have that $\nu \in [\![\text{andify}(Q_\nu)]\!]_D$, considering a very similar argument for showing that ν also satisfies all the filter (in-)equalities as well.

The previous discussion implies that $\nu \in \mathsf{partials}(D, Q)$, and as $\mu \sqsubset \nu$ this contradicts the fact that μ is maximal in $\mathsf{partials}(D, Q)$. Therefore, μ is maximal in $\mathsf{partials}(D, P)$. Thus, $\mu \in [\![P]\!]_D$, and we conclude that $\Omega \subseteq [\![P]\!]_D$.

Item 2. The proof of item 2 is similar and has been omitted. □

We now turn to the full filter, whereby we define a new type of canonical query. Given (D, Ω), define the query $P^F_{\mathsf{can}}(D, \Omega)$, and denote it with P for brevity, such that

$P = (V_P, E_P, r_P, \lambda_P, \delta_P)$, P has the same structure as Ω, for each node $n \in V_P$ it is the case that $\lambda_P(n) = \text{atype}(D, \Omega_{S_n})$, where $S_n = \mathsf{scope}_\Omega(n)$, and letting $\Omega_n = \{\mu \in \Omega \mid \mathsf{scope}_\Omega(n) \subseteq \text{dom}(\mu)\}$ and $\text{dom}_n(\mu) = \{?\mathtt{X} \mid ?\mathtt{X} \in \text{dom}(\mu) \cap \mathsf{scope}_\Omega(n)\}$:

$$\delta_P(n) = \bigvee_{\mu \in \Omega_n} \bigwedge_{?\mathtt{X} \in \text{dom}_n(\mu)} ?\mathtt{X} = \mu(?\mathtt{X}).$$

Lemma 3.7. *Given an RDF graph D and a tree-like set of mappings Ω,*

1. *if $(D, \Omega) \in \text{RevEng}^+_{\mathsf{tree}}(\text{SP}[\mathsf{AOFwd}])$ then $\Omega \subseteq [\![P^F_{\mathsf{can}}(D, \Omega)]\!]_D$,*
2. *if $(D, \Omega) \in \text{RevEng}^{\mathsf{E}}_{\mathsf{tree}}(\text{SP}[\mathsf{AOFwd}])$ then $\Omega = [\![P^F_{\mathsf{can}}(D, \Omega)]\!]_D$.*

Proof. We proceed item by item.

Item 1. Assume that $(D, \Omega) \in \text{RevEng}^+(\text{SP}[\mathsf{AOFwd}])$, whereby there is a query $Q \in \text{SP}[\mathsf{AOFwd}]$ such that $\Omega \subseteq [\![Q]\!]_D$. Note also that Q is compatible with Ω. Now consider the canonical query $P = P^F_{\mathsf{can}}(D, \Omega)$ and note that P has the same tree structure as Ω. We must show that $\Omega \subseteq [\![P]\!]_D$.

Consider a mapping $\mu \in \Omega$ and note that $\text{dom}(\mu)$ is closed in Ω, implying that there exists a reduction $P_\mu \trianglelefteq P$ such that $\text{var}(P_\mu) = \text{dom}(\mu)$. Also, by construction, we have that for each node n in P_μ and triple pattern $t \in \lambda_{P_\mu}(n)$ it is the case that $\mu(t) \in D$. Furthermore, for each node n in P_μ, note that by construction we have that $\mu \models \delta_{P_\mu}(n)$ as well. Therefore, we have that $\mu \in \mathsf{partials}(D, P)$. We now must show that μ is maximal in $\mathsf{partials}(D, P)$.

For the sake of contradiction, assume that μ is not maximal in $\mathsf{partials}(D, P)$, whereby there exists a mapping $\nu \in \mathsf{partials}(D, P)$ such that $\mu \sqsubset \nu$. Also, note that there is a reduction $P_\nu \trianglelefteq P$ such that $\nu \in [\![\text{andify}(P_\nu)]\!]_D$, i.e. for every node n in P_ν and triple pattern $t \in \lambda_{P_\nu}(n)$ it is the case that $\nu(t) \in D$ and $\nu \models \delta_{P_\nu}(n)$. Now, due to the construction of $\delta_{P_\nu}(n)$, there must exist a mapping $\sigma \in \Omega$ such that $\mathsf{scope}_{P_\nu}(n) \subseteq \text{dom}(\sigma)$ and for every variable $?\mathtt{X} \in \mathsf{scope}_P(n)$ we have $\nu(?\mathtt{X}) = \sigma(?\mathtt{X})$. We now turn to the pattern tree Q, and claim that $\nu \in \mathsf{partials}(D, Q)$. For this, note that as P has the same structure as Ω, $\text{var}(P_\nu)$ is closed in Ω, whereby there is a reduction $Q_\nu \trianglelefteq Q$ such that $\text{var}(Q_\nu) = \text{dom}(\nu)$. Using arguments similar to previous proofs, we have that for every node n in Q_ν and triple pattern $t \in \lambda_{Q_\nu}(n)$ it will be the case that $\nu(t) \in D$. For the filter patterns, consider an arbitrary node n in Q_ν and the corresponding filter pattern $\phi_n = \delta_{Q_\nu}(n)$. Take the set of variables $S_n = \{?\mathtt{X} \mid \mathsf{top}_{Q_\nu}(?\mathtt{X})\}$ (this set must be non-empty, as Q is in productive form) and find the corresponding node m in P_ν such that $m = \mathsf{top}_{P_\nu}(?\mathtt{X})$ for all $?\mathtt{X} \in S_n$ (note that this node must exist in order for Q to be compatible with Ω).

Note that $\mathsf{scope}_{Q_\nu}(n) \subseteq \mathsf{scope}_{P_\nu}(m)$. Now, due to the discussion at the beginning of this paragraph, for the node m in P_ν there must exist a mapping $\sigma_m \in \Omega$ such that for every variable in $\mathsf{scope}_{P_\nu}(m)$ it is the case that $\nu(?\mathtt{X}) = \sigma_m(?\mathtt{X})$. Also, $\mathsf{scope}_{P_\nu}(m) \subseteq \mathrm{dom}(\sigma_m)$, whereby $\mathsf{scope}_{Q_\nu}(n) \subseteq \mathrm{dom}(\sigma)$. As $\Omega \subseteq [\![Q]\!]_D$ we have a reduction $Q_{\sigma_m} \trianglelefteq Q$ such that $\sigma_m \in [\![\mathrm{andify}(Q_{\sigma_m})]\!]_D$, and notice that Q_{σ_m} includes node n. Finally, as $\sigma_m(?\mathtt{X}) = \nu(?\mathtt{X})$ for every variable $?\mathtt{X} \in \mathsf{scope}_Q(n)$ we have that $\nu \models \phi_n$, as ϕ_n may only mention variables in $\mathsf{scope}_Q(n)$ and we must have that $\sigma \models \phi_n$. Therefore, $\nu \in \mathsf{partials}(D, Q)$, which contradicts the fact that μ is maximal in $\mathsf{partials}(D, Q)$. Thus, we conclude that μ is maximal in $\mathsf{partials}(D, P)$. In conclusion, we have that $\Omega \subseteq [\![P]\!]_D$, which is what was required.

Item 2. Assume that $(D, \Omega) \in \textsc{RevEng}^{\mathsf{E}}(\mathrm{SP}[\mathsf{AOFwd}])$, whereby there is a pattern tree $Q \in \mathrm{SP}[\mathsf{AOFwd}]$ such that $\Omega = [\![Q]\!]_D$. In particular, $\Omega \subseteq [\![Q]\!]_D$, and we use the result of item 1 to conclude that $P = P^F_{\mathsf{can}}(D, \Omega)$ is such that $\Omega \subseteq [\![P]\!]_D$. Therefore, we need only show that $[\![P]\!]_D \subseteq \Omega$, or equivalently, that $[\![P]\!]_D \subseteq [\![Q]\!]_D$.

For this, consider an arbitrary mapping $\mu \in [\![P]\!]_D$, whereby there exists a reduction $P_\mu \trianglelefteq P$ such that $\mu \in [\![\mathrm{andify}(P_\mu)]\!]_D$, i.e. $\mu \in \mathsf{partials}(D, P)$, and also μ is maximal in $\mathsf{partials}(D, P)$. As P has the same structure as Ω, we have that $\mathrm{dom}(\mu)$ is closed in Ω, and as Q is compatible with Ω, we have that there is a reduction $Q_\mu \trianglelefteq Q$ such that $\mathrm{var}(Q_\mu) = \mathrm{dom}(\mu)$. Consider a node n in Q_μ and a triple pattern $t \in \lambda_{Q_\mu}(n)$. As $\sigma(t) \in D$ for every $\sigma \in \Omega$ such that $\mathsf{scope}_Q(n) \subseteq \mathrm{dom}(\sigma)$, we have that $t \in \mathrm{atype}(D, \Omega_{\mathsf{scope}_Q(n)})$. Also, if we take variables $S_n = \{?\mathtt{X} \mid n = \mathsf{top}_Q(?\mathtt{X})\}$ and find the corresponding (and unique) node m in P_μ such that $m = \mathsf{top}_P(?\mathtt{X})$ for $?\mathtt{X} \in S_n$, we see that $\mathsf{scope}_Q(n) \subseteq \mathsf{scope}_P(m)$, whereby $t \in \mathrm{atype}(D, \Omega_{\mathsf{scope}_P(m)})$ as well, and therefore $t \in \lambda_P(m)$ by construction. Thus, as $\mu \in [\![\mathrm{andify}(P_\mu)]\!]_D$ we have that $\mu(t) \in D$. For the filter pattern, consider $\phi_n = \delta_{Q_\mu}(n)$. Using an argument similar to that of item 1, we see that $\mu \models \phi_n$. We conclude that $\mu \in \mathsf{partials}(D, Q)$. We now claim that μ is maximal in $\mathsf{partials}(D, Q)$.

Assume, for the sake of contradiction, that μ is not maximal in $\mathsf{partials}(D, Q)$, whereby there is another mapping $\nu \in \mathsf{partials}(D, Q)$ such that $\mu \sqsubset \nu$ and ν is maximal in $\mathsf{partials}(D, Q)$, and thus $\nu \in [\![Q]\!]_D$. As $\Omega = [\![Q]\!]_D$ we have that $\nu \in \Omega$, and as we have already shown, $\Omega \subseteq [\![P]\!]_D$ implies that $\nu \in [\![P]\!]_D$, which contradicts the fact that $\mu \in [\![P]\!]_D$, as both μ and ν cannot be maximal in $\mathsf{partials}(D, P)$. Therefore, μ is maximal in $\mathsf{partials}(D, Q)$. Thus, $\mu \in [\![Q]\!]_D$ and we have shown that $\Omega = [\![P]\!]_D$, as was needed. □

Proposition 3.11. *The following decision problems are in* **coNP***:*

1. $\textsc{RevEng}^{+}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$,

2. $\textsc{RevEng}^{\mathsf{E}}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$,

3. $\textsc{RevEng}^{+}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$,

4. $\textsc{RevEng}^{\mathsf{E}}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$,

Proof. We proceed item by item.

Item 1. For $\textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOwd}])$, consider an algorithm which, on input (D, Ω) builds $P_{\mathsf{can}}(D, \Omega)$ in polynomial time and then runs the algorithm presented in the proof of $\textsc{Verify}^{+}(\mathrm{SP}[\mathsf{AOwd}]) \in \mathbf{coNP}$ (as seen in the proof of Proposition 3.4), with input $(D, \Omega, P_{\mathsf{can}}(D, \Omega))$. Now, clearly, if $P_{\mathsf{can}}(D, \Omega)$ is such that $(D, \Omega, P_{\mathsf{can}}(D, \Omega)) \in \textsc{Verify}^{+}(\mathrm{SP}[\mathsf{AOwd}])$, then $(D, \Omega) \in \textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOwd}])$, with $P_{\mathsf{can}}(D, \Omega)$ as a witness. On the other hand, due to Lemma 3.5, item 1, if $\Omega \not\subseteq [\![P_{\mathsf{can}}(D, \Omega)]\!]_D$, then $(D, \Omega) \notin \textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOwd}])$. We thus conclude that the previous algorithm is correct, whereby $\textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOwd}]) \in \mathbf{coNP}$.

Item 2. For $\textsc{RevEng}^{\mathsf{E}}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$, the proof is very similar to the previous, and uses Lemma 3.5, item 2 (which corresponds to Lemma 3.5, item 3), and considering that $\textsc{Verify}^{\mathsf{E}}(\mathrm{SP}[\mathsf{AOwd}]) \in \mathbf{coNP}$ as well (see the discussion following Proposition 3.5).

Item 3. For $\textsc{RevEng}^{+}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$ the proof is very similar, building $P^{=,\neq}_{\mathsf{can}}(D, \Omega)$ and using the fact that $\textsc{Verify}^{+}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}]) \in \mathbf{coNP}$ and Lemma 3.6, item 1.

Item 4. For $\textsc{RevEng}^{\mathsf{E}}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$ the proof is again similar, building $P^{=,\neq}_{\mathsf{can}}(D, \Omega)$ and using the fact that $\textsc{Verify}^{\mathsf{E}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}]) \in \mathbf{coNP}$ and Lemma 3.6, item 2. $\square$

We now return to the general case of reverse engineering, i.e. where $\mathcal{C}(\Omega)$ is not necessarily tree-like. In this case, there exist many different candidate $\mathrm{SP}[\mathsf{AOwd}]$ queries for an input (D, Ω). Intuitively, these candidates still conform to the structure $\mathcal{C}(\Omega)$, yet in a more relaxed sense: these are queries whose maximal $\mathrm{SP}[\mathsf{A}]$ subqueries can be "merged" to obtain the structure $\mathcal{C}(\Omega)$.

To formalise the previous idea, first recall that a query $P \in \mathrm{SP}[\mathsf{AOwd}]$ is assumed to be in **OPT** normal form, whereby it is possible to consider P to be formed by $\mathrm{SP}[\mathsf{A}]$ queries, combined with the **OPT** operator. Given a query $P \in \mathrm{SP}[\mathsf{AOwd}]$, a subquery $Q \in \mathrm{SP}[\mathsf{A}]$ of P is *left-most* in P if and only if Q is such that there does not exist any subquery $(P_1 \mathbin{\mathbf{OPT}} P_2)$ of P such that Q is a subquery of P_2. Furthermore, given two subqueries $P_1, P_2 \in \mathrm{SP}[\mathsf{A}]$ of P, P_1 is an *ancestor* of P_2 in P if and only if either $P_1 = P_2$ or there exists a subquery $P'_1 \,\mathbf{OPT}\, P'_2$ of P such that P_1 is the left-most subquery of P'_1 and P_2 is a subquery of P'_2. Finally, given a subquery Q of P, a variable $?\mathtt{X}$ is *top-most* in Q if $?\mathtt{X}$ is present in the left-most subquery of Q yet absent outside of Q in P. With this, an $\mathrm{SP}[\mathsf{AOwd}]$ query P is a *candidate* for (D, Ω) if there exists a surjective function h from its maximal $\mathrm{SP}[\mathsf{A}]$ subqueries to $\mathcal{C}(\Omega)$ that:

1. Each variable $\mathbf{v} \in \mathrm{dom}(\Omega)$ can be associated to an $\mathrm{SP}[\mathsf{A}]$ subquery $Q_{\mathbf{v}}$ of P such that $h(Q_{\mathbf{v}}) = \mathsf{Cov}_\Omega(\mathbf{v})$ and $\mathbf{v}$ is top-most in $Q_{\mathbf{v}}$,

2. h preserves the query's tree structure: given $P_1, P_2 \in \mathrm{SP}[\mathsf{A}]$ subqueries of P, if P_1 is an ancestor of P_2 in P, then $h(P_1)$ is a superset (i.e. ancestor) of $h(P_2)$ in $\mathcal{C}(\Omega)$,

3. For each Λ in $\mathcal{C}(\Omega)$ and R in the pre-image $h^{-1}(\Lambda)$, R consists of all triple patterns in $\mathrm{atype}(D, \Omega_S)$, where S is the set containing every variable $\mathbf{v}$ for which $Q_{\mathbf{v}}$ is an ancestor of R in P.

Note that if Ω is tree-like, then there is exactly one candidate query, which is the canonical query. When considering pattern trees, the previous definition is equivalent to a pattern tree being such that P is compatible with Ω and containing in each node n the atomic type $\lambda_P(n) = \mathrm{atype}(D, \Omega_n)$ (and similarly for the filter cases).

A pattern tree $P \in \mathrm{SP}[\mathsf{AOwd}]$ is an $\mathrm{SP}[\mathsf{AOwd}]$-*candidate* for (Ω, D) if

1. P is compatible with Ω,

2. For each node n in P, if $S = \mathsf{scope}_P(n)$, then $\lambda_P(n) = \mathrm{atype}(D, \Omega_n)$.

Similarly, a pattern tree $P \in \mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}]$ is an $\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}]$-*candidate* for (Ω, D) if the previous two items hold, and also for each node n in P, if $S = \mathsf{scope}_P(n)$, then $\delta_P(n) = \mathrm{atype}_{=,\neq}(D, \Omega_n)$.

Finally, a pattern tree $P \in \mathrm{SP}[\mathsf{AOFwd}]$ is an $\mathrm{SP}[\mathsf{AOFwd}]$-*candidate* for (Ω, D) if the initial two items hold, and also for every node $n \in V_P$, $\lambda_P(n) = \mathrm{atype}(D, \Omega_{\mathsf{scope}_P(n)})$ and $\delta_P(n)$ is:

$$\delta_P(n) = \bigvee_{\mu \in \Omega_n} \bigwedge_{?\mathtt{X} \in \mathrm{dom}_n(\mu)} ?\mathtt{X} = \mu(?\mathtt{X}),$$

where $\Omega_n = \{\mu \in \Omega \mid \mathsf{scope}_P(n) \subseteq \mathrm{dom}(\mu)\}$ and $\mathrm{dom}_n(\mu) = \mathrm{dom}(\mu) \cap \mathsf{scope}_P(n)$.

In what follows, as the SPARQL fragment is usually obvious, we will shorten notation by simply saying a candidate query P for (D, Ω).

Example 3.7. *Consider the set of mappings $\Omega' = \{\mu_1, \mu_2, \mu_3, \mu_4'\}$ from Example 3.6. Depending on the graph D, a candidate query for the input (D, Ω') may follow either of two distinct* **OPT** *structures, both arising by choosing a parent for the $\{\mu_4'\}$ element of $\mathcal{C}(\Omega')$.*

We obtain the following generalisation of Lemma 3.5:

Lemma 3.8. *Given RDF graph D and set of mappings Ω,*

1. *if $(D, \Omega) \in \textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOwd}])$ then there is a candidate query P for (D, Ω) such that $\Omega \subseteq [\![P]\!]_D$;*

2. *if $(D, \Omega) \in \textsc{RevEng}^{\mathsf{E}}(\mathrm{SP}[\mathsf{AOwd}])$ then there is a candidate query P for (D, Ω) such that $\Omega = [\![P]\!]_D$.*

Proof. Recall that given an RDF graph D and a set of mappings Ω, a query $P \in \mathrm{SP}[\mathsf{AOwd}]$ is a candidate for (D, Ω) if and only if (a) P is compatible with Ω (when P is viewed as a pattern tree), and (b) for every node $n \in V_P$, $\lambda_P(n) = \mathrm{atype}(D, \Omega_{\mathsf{scope}_P(n)})$.

Item 1. Assume $(D, \Omega) \in \textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOwd}])$ whereby there exists a pattern tree $P \in \mathrm{SP}[\mathsf{AOwd}]$ such that $\Omega \subseteq [\![P]\!]_D$. Due to Lemma 3.3 we have that P is compatible with Ω. Now build a pattern tree $Q = (V_Q, E_Q, r_Q, \lambda_Q)$ such that $V_Q = V_P$, $E_Q = E_P$, $r_Q = r_P$, and for every node $n \in V_Q$, $\lambda_Q(n) = \mathrm{atype}(D, \Omega_{\mathsf{scope}_P(n)})$. We claim that $\Omega \subseteq [\![Q]\!]_D$. For this, consider a mapping $\mu \in \Omega$ and note that as $\mu \in [\![P]\!]_D$ there is a reduction $P_\mu \trianglelefteq P$ such that $\mu \in [\![\mathrm{andify}(P_\mu)]\!]_D$. As Q has the same tree structure as P we also have a reduction $Q_\mu \trianglelefteq Q$ such that $\mathrm{var}(Q_\mu) = \mathrm{dom}(\mu)$. Now consider a node n in Q_μ and a triple pattern $t \in \lambda_{Q_\mu}(n) = \mathrm{atype}(D, \Omega_{\mathsf{scope}_P(n)})$. Note that $\mathsf{scope}_P(n) \subseteq \mathrm{dom}(\mu)$ whereby by construction we have that $\mu(t) \in D$; thus $\mu \in [\![\mathrm{andify}(Q_\mu)]\!]_D$. We must now show that μ is maximal in $\mathsf{partials}(D, Q)$. For the sake of contradiction, assume that there is a mapping $\nu \in \mathsf{partials}(D, Q)$ such that $\mu \sqsubset \nu$, whereby there is a pattern tree $Q_\nu \trianglelefteq Q$ such that $\nu \in [\![\mathrm{andify}(Q_\nu)]\!]_D$ (also note that $Q_\mu \trianglelefteq Q_\nu$ and that $Q_\mu \neq Q_\nu$). As P has the same tree structure as Q, there is a pattern tree $P_\nu \trianglelefteq P$ such that $\mathrm{var}(P_\nu) = \mathrm{dom}(\nu)$. Now consider a node n in P_ν and a triple pattern $t \in \lambda_P(n)$. As $\Omega \subseteq [\![P]\!]_D$ it is the case that for every mapping $\sigma \in \Omega$ such that $\mathsf{scope}_P(n) \subseteq \sigma$ it is the case that $\sigma(t) \in D$, whereby t is also present in $\lambda_Q(n)$, implying that $\nu(t) \in D$ as well. Therefore, we have that $\nu \in [\![\mathrm{andify}(P_\nu)]\!]_D$, which contradicts the fact that μ is maximal. Therefore, Q is a candidate for (D, Ω).

Item 2. Assume $(D, \Omega) \in \textsc{RevEng}^{\mathsf{E}}(\mathrm{SP}[\mathsf{AOwd}])$, whereby there exists a pattern tree $P \in \mathrm{SP}[\mathsf{AOwd}]$ such that $\Omega = [\![P]\!]_D$. In particular, we may build the query Q as in the proof of item 1 and note that Q is a candidate for (D, Ω) and that $\Omega \subseteq [\![Q]\!]_D$. We must now prove that $[\![Q]\!]_D \subseteq \Omega$, or equivalently, that $[\![Q]\!]_D \subseteq [\![P]\!]_D$. For this, consider a mapping $\mu \in [\![Q]\!]_D$, whereby there exists a pattern tree $Q_\mu \trianglelefteq Q$ such that $\mu \in [\![\text{andify}(Q_\mu)]\!]_D$ and μ is maximal in $\mathsf{partials}(D, Q)$. Now, as P has the same tree structure as Q, there exists a pattern tree $P_\mu \trianglelefteq P$ such that $\text{var}(P_\mu) = \text{dom}(\mu)$. Consider a node n in P_μ and a triple pattern $t \in \lambda_{P_\mu}(n)$. As $\Omega \subseteq [\![P]\!]_D$ we have that for every $\sigma \in \Omega$ such that $\mathsf{scope}_{P_\mu}(n) \subseteq \text{dom}(\sigma)$ it is the case that $\sigma(t) \in D$, whereby $t \in \text{atype}(D, \Omega_{\mathsf{scope}_P(n)})$, and thus $\mu(t) \in D$ as well; thus we have that $\mu \in [\![\text{andify}(P_\mu)]\!]_D$. We must now show that μ is maximal in $\mathsf{partials}(D, P)$. For the sake of contradiction, assume that μ is not maximal, whereby there exists a mapping $\nu \in \mathsf{partials}(D, P)$ such that $\mu \sqsubset \nu$ such that ν is maximal in $\mathsf{partials}(D, P)$. In this case, we have that $\nu \in [\![P]\!]_D = \Omega$. However, as $\Omega \subseteq [\![Q]\!]_D$ we have that $\nu \in [\![Q]\!]_D$, which contradicts the fact that μ is maximal in $\mathsf{partials}(D, Q)$. Therefore, $\mu \in [\![P]\!]_D$. We conclude, then, that $[\![Q]\!]_D \subseteq [\![P]\!]_D$. Finally, Q (build as in the proof of item 1) is such that Q is a candidate for (D, Ω) and $\Omega = [\![Q]\!]_D$. □

If we replace atype with $\text{atype}_{=,\neq}$ in the definition of candidate queries above, we can prove a similar result to Lemma 3.8 for $\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}]$ (see Lemma 3.9) and $\text{SP}[\mathsf{AOFwd}]$ (see Lemma 3.10).

Lemma 3.9. *Given an RDF graph D and a set of mappings Ω,*

1. *if $(D, \Omega) \in \text{RevEng}^+(\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$ then there is a candidate query P for (D, Ω) such that $\Omega \subseteq [\![P]\!]_D$;*

2. *if $(D, \Omega) \in \text{RevEng}^{\mathsf{E}}(\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$ then there is a candidate query P for (D, Ω) such that $\Omega = [\![P]\!]_D$.*

Proof. This proof is very similar to that of Lemma 3.8, and has been omitted. □

Lemma 3.10. *Given an RDF graph D and set of mappings Ω,*

1. *if $(D, \Omega) \in \text{RevEng}^+(\text{SP}[\mathsf{AOFwd}])$ then there is an* $\text{SP}[\mathsf{AOF}]$*-candidate query P for (D, Ω) such that $\Omega \subseteq [\![P]\!]_D$,*

2. *if $(D, \Omega) \in \text{RevEng}^{\mathsf{E}}(\text{SP}[\mathsf{AOFwd}])$ then there is an* $\text{SP}[\mathsf{AOF}]$*-candidate query P for (D, Ω) such that $\Omega = [\![P]\!]_D$.*

Proof. A crucial difference between the result that must be shown here and the proof of Lemma 3.9 is that filter expressions in the fragment SP[AOFwd] cannot be regarded merely as sets of equalities and inequalities, and candidate queries no longer contain an *atomic type* of filter equalities and inequalities.

Firstly, notice that given an RDF graph D and a set of mappings Ω, a query $P \in$ SP[AOFwd] is a candidate for (D, Ω) if and only if (a) P is compatible with Ω (when P is viewed as a pattern tree), and (b) for every node $n \in V_P$, $\lambda_P(n) = \text{atype}(D, \Omega_{\mathsf{scope}_P(n)})$ and $\delta_P(n)$ is defined as for $P^F_{\mathsf{can}}(D, \Omega)$, that is,

$$\delta_P(n) = \bigvee_{\mu \in \Omega_n} \bigwedge_{?\mathtt{X} \in \text{dom}_n(\mu)} ?\mathtt{X} = \mu(?\mathtt{X}),$$

where recall that $\Omega_n = \{\mu \in \Omega \mid \mathsf{scope}_P(n) \subseteq \text{dom}(\mu)\}$ and $\text{dom}_n(\mu) = \text{dom}(\mu) \cap \mathsf{scope}_P(n)$.

Item 1. Assume $(D, \Omega) \in \text{REVENG}^+(\text{SP}[\mathsf{AOFwd}])$ whereby there exists a pattern tree $Q \in$ SP[AOFwd] such that $\Omega \subseteq [\![Q]\!]_D$. Due to Lemma 3.3 we have that Q is compatible with Ω. Now build a pattern tree $P = (V_P, E_P, r_P, \lambda_P, \delta_P)$ such that $V_P = V_Q$, $E_P = E_Q$, $r_P = r_Q$, and for every node $n \in V_P$, $\lambda_P(n) = \text{atype}(D, \Omega_{\mathsf{scope}_Q(n)})$ and $\delta_P(n)$ is defined as for the full-filter candidate. We claim that $\Omega \subseteq [\![P]\!]_D$.

For this, consider a mapping $\mu \in \Omega$ and note that as $\mu \in [\![Q]\!]_D$ there is a reduction $Q_\mu \trianglelefteq Q$ such that $\mu \in [\![\text{andify}(Q_\mu)]\!]_D$. As P has the same tree structure as Q we also have a reduction $P_\mu \trianglelefteq P$ such that $\text{var}(P_\mu) = \text{dom}(\mu)$. Now consider a node n in P_μ and a triple pattern $t \in \lambda_{P_\mu}(n) = \text{atype}(D, \Omega_{\mathsf{scope}_P(n)})$. Note that $\mathsf{scope}_Q(n) \subseteq \text{dom}(\mu)$ whereby by construction we have that $\mu(t) \in D$. Next, for node n in P_μ consider the filter expression $\phi_n = \delta_P(n)$ and notice that by construction we have that $\mu \models \phi_n$ (as $\mu \in \Omega$). Thus, $\mu \in [\![\text{andify}(P_\mu)]\!]_D$. We must now show that μ is maximal in $\mathsf{partials}(D, P)$.

For the sake of contradiction, assume that there is a mapping $\nu \in \mathsf{partials}(D, P)$ such that $\mu \sqsubset \nu$, whereby there is a pattern tree $P_\nu \trianglelefteq P$ such that $\nu \in [\![\text{andify}(P_\nu)]\!]_D$ (also note that $P_\mu \trianglelefteq P_\nu$ and that $P_\mu \neq P_\nu$). As P has the same tree structure as Q, there is a reduction $Q_\nu \trianglelefteq Q$ such that $\text{var}(Q_\nu) = \text{dom}(\nu)$ as well.

Now consider a node n in Q_ν and a triple pattern $t \in \lambda_Q(n)$. As $\Omega \subseteq [\![Q]\!]_D$ it is the case that for every mapping $\sigma \in \Omega$ such that $\mathsf{scope}_Q(n) \subseteq \sigma$ it is the case that $\sigma(t) \in D$, whereby t is also present in $\lambda_P(n)$, implying that $\nu(t) \in D$ as well. Therefore, we have that $\nu \in [\![\text{andify}(Q_\nu)]\!]_D$, which contradicts the fact that μ is maximal in $\mathsf{partials}(D, Q)$. Next consider the filter expression $\phi_n = \delta_P(n)$, and notice that by construction $\nu \models \phi_n$ implies that there exists a mapping $\sigma \in \Omega$ such that $\mathsf{scope}_P(n) \subseteq \mathrm{dom}(\sigma)$ and for every variable $?\mathtt{X} \in \mathsf{scope}_P(n)$ it is the case that $\nu(?\mathtt{X}) = \sigma(?\mathtt{X})$. Now, as $\Omega \subseteq [\![Q]\!]_D$ it must also be the case that $\sigma \models \delta_Q(n')$, where n' is the node in Q which corresponds to P (note that P and Q have the same variable structure). However, as $\delta_Q(n')$ may only mention variables in $\mathsf{scope}_Q(n')$ we also have that $\nu \models \delta_Q(n')$, whereby $\nu \in \mathsf{partials}(D, Q)$, contradicting the fact that μ is a maximal partial answer. Therefore, μ is maximal in $\mathsf{partials}(D, P)$, and we conclude that $\Omega \subseteq [\![P]\!]_D$. Therefore, P is a full-filter candidate for (D, Ω).

Item 2. Assume $(D, \Omega) \in \textsc{RevEng}^{\mathsf{E}}(\mathrm{SP}[\mathsf{AOFwd}])$, whereby there exists a pattern tree $Q \in \mathrm{SP}[\mathsf{AOFwd}]$ such that $\Omega = [\![Q]\!]_D$. In particular, we may build the query P as in the proof of item 1 and note that P is a full-filter candidate for (D, Ω) and that $\Omega \subseteq [\![P]\!]_D$. We need only prove that $[\![P]\!]_D \subseteq \Omega$, or equivalently, that $[\![P]\!]_D \subseteq [\![Q]\!]_D$.

For this, consider a mapping $\mu \in [\![P]\!]_D$, whereby there exists a pattern tree $P_\mu \trianglelefteq P$ such that $\mu \in [\![\mathrm{andify}(P_\mu)]\!]_D$ and μ is maximal in $\mathsf{partials}(D, P)$. Now, as Q has the same tree structure as P, there exists a pattern tree $Q_\mu \trianglelefteq Q$ such that $\mathrm{var}(Q_\mu) = \mathrm{dom}(\mu)$. Consider a node n in Q_μ and a triple pattern $t \in \lambda_{Q_\mu}(n)$. As $\Omega \subseteq [\![Q]\!]_D$ we have that for every $\sigma \in \Omega$ such that $\mathsf{scope}_{Q_\mu}(n) \subseteq \mathrm{dom}(\sigma)$ it is the case that $\sigma(t) \in D$, whereby $t \in \mathrm{atype}(D, \Omega_{\mathsf{scope}_Q(n)})$, and thus $\mu(t) \in D$ as well. For the filter expression, consider $\phi_n = \delta_{Q_\mu}(n)$. Firstly, the corresponding node m in P_μ (recall that P and Q have the same tree structure) has a filter expression φ_m built in such a way that there exists a mapping $\sigma \in \Omega$ such that $\mathsf{scope}_P(m) \subseteq \mathrm{dom}(\sigma)$ and for every variable $?\mathtt{X} \in \mathsf{scope}_P(m)$ it is the case that $\mu(?\mathtt{X}) = \sigma(?\mathtt{X})$; also, as $\sigma \in [\![\mathrm{andify}(Q_\mu)]\!]_D$ it must be the case that $\sigma \models \phi_n$, while ϕ_n mentions only variables in $\mathsf{scope}_Q(n) = \mathsf{scope}_P(m)$, whereby it must also be the case that $\mu \models \phi_n$. Thus we have that $\mu \in [\![\mathrm{andify}(Q_\mu)]\!]_D$, or $\mu \in \mathsf{partials}(D, Q)$. We must now show that μ is maximal in $\mathsf{partials}(D, Q)$.

For the sake of contradiction, assume that μ is not maximal in $\mathsf{partials}(D, Q)$, whereby there exists a mapping $\nu \in \mathsf{partials}(D, Q)$ such that $\mu \sqsubset \nu$ such that ν is maximal in $\mathsf{partials}(D, Q)$. In this case, we have that $\nu \in [\![Q]\!]_D = \Omega$. However, as $\Omega \subseteq [\![P]\!]_D$ we have that $\nu \in [\![P]\!]_D$, which contradicts the fact that μ is maximal in $\mathsf{partials}(D, P)$. Therefore, $\mu \in [\![Q]\!]_D$. We conclude, then, that $[\![P]\!]_D \subseteq [\![Q]\!]_D$. Finally, P is a candidate for (D, Ω) and $\Omega = [\![P]\!]_D$. □

Note also, that candidate patterns are always of polynomial size, so the above result leads to an upper bound for the complexity of the corresponding reverse engineering problems:

Proposition 3.12. *The following decision problems are in Σ_2^p:*

1. $\textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOwd}])$,

2. $\text{RevEng}^{\mathsf{E}}(\text{SP}[\mathsf{AOwd}])$,

3. $\text{RevEng}^{+}(\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$,

4. $\text{RevEng}^{\mathsf{E}}(\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$.

Proof. We proceed item by item.

Item 1. For $\text{RevEng}^{+}(\text{SP}[\mathsf{AOwd}])$, by Lemma 3.8 it suffices to guess a (polynomially-sized) candidate query and *verify* that it realises the input (note that the decision problem $\text{Verify}^{+}(\text{SP}[\mathsf{AOwd}]) \in \mathbf{coNP}$, as per Proposition 3.4).

Item 2. For $\text{RevEng}^{\mathsf{E}}(\text{SP}[\mathsf{AOwd}])$, the situation is similar (Proposition 3.5 shows that $\text{Verify}^{\mathsf{E}}(\text{SP}[\mathsf{AOwd}]) \in \mathbf{coNP}$.

All previous cases lead to a non-deterministic, polynomial-time algorithm which makes use of an oracle for a problem in **coNP**, so we have that $\text{RevEng}^{+}(\text{SP}[\mathsf{AOwd}])$ and $\text{RevEng}^{\mathsf{E}}(\text{SP}[\mathsf{AOwd}])$ are in Σ_2^p

For $\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}]$, the proofs are very similar, considering that $\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}]$ allows us to guess polynomially-sized queries as well, due to the fact that each filter expression is a set of equalities and inequalities between variables of Ω and constants of D, of which there exist only a polynomial amount. In this case, Lemma 3.9 may be used and similar algorithms derived. □

For the next result, we require a definition. Given a set of mappings Ω, we say Ω is *path-consistent* if and only if for every mapping $\mu \in \Omega$ and node n in $\mathcal{C}(\Omega)$ such that $\mu \in n$ but no children of n contain μ, and for every mapping $\nu \in \Omega$ such that $\nu \in n$ and node m such that m is a child of n in $\mathcal{C}(\Omega)$ and $\nu \in m$, it is not the case that $\mu_n = \nu_n$, where μ_n is the mapping such that $\text{dom}(\mu_n) = \mathsf{scope}_\Omega(n)$ and $\mu_n \sqsubseteq \mu$.

Lemma 3.11. *Given RDF graph D and a set of mappings Ω, if $(D, \Omega) \in \text{RevEng}^{+}$ $(\text{SP}[\mathsf{AOF}])$ then Ω is path-consistent.*

Proof. Assume that $P \in \text{SP}[\mathsf{AOF}]$ and that $\Omega \subseteq [\![P]\!]_D$ and assume, for the sake of contradiction, that Ω is not path-consistent, whereby there are mappings $\mu, \nu \in \Omega$ nodes $n, m \in \mathcal{C}(\Omega)$ such that m is a child of n in $\mathcal{C}(\Omega)$, $\mu \in n$ (i.e. meaning that μ mentions all variables in the scope of n) and both $\nu \in n$ and $\nu \in m$.

The previous, though, implies that we may construct mapping μ^* from μ by adding the variables corresponding to node m, assigning the values given by ν, which implies that μ is not maximal. As this is a contradiction, we conclude that Ω is path-consistent. □

Proposition 3.13. *The following decision problems are in* **NP**:

1. $\textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOFwd}])$,

2. $\textsc{RevEng}^{\mathsf{E}}(\mathrm{SP}[\mathsf{AOFwd}])$.

Proof. **Item 1.** For $\textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOFwd}])$, consider the following **NP** algorithm. On input (D, Ω), guess a tree structure compatible with Ω, and then construct a canonical query $P \in \mathrm{SP}[\mathsf{AOFwd}]$ with said structure. If the pattern tree is well-formed, then return True; otherwise, return False.

If the previous algorithm returns True, we claim that P is such that $\Omega \subseteq [\![P]\!]_D$. For this, consider a mapping $\mu \in \Omega$ and notice that $\mu \in \mathsf{partials}(D, P)$ by construction. We now claim that μ is maximal. For the sake of contradiction, assume that it is not maximal, whereby there is a mapping $\nu \in \mathsf{partials}(D, P)$ such that $\mu \sqsubset \nu$. Consider nodes n, m in P such that $(n, m) \in E(P)$, $n, m \in P_\nu$, $n \in P_\mu$, and $m \notin P_\mu$ (such a combination must exist as $\mu \sqsubset \nu$). As ν is a partial answer, and due to the construction of P there must exist a mapping $\nu^* \in \Omega$ such that $\nu(\mathbf{v}) = \nu^*(\mathbf{v})$ for every variable $v \in \mathsf{scope}_P(m)$, which implies that μ and ν^* witness the fact that Ω is not path-consistent. This is a contradiction, whereby we conclude that $\Omega \subseteq [\![P]\!]_D$.

On the other hand, if the algorithm returns False, then every canonical query (for every possible compatible tree structure) is not well-formed, which implies that there do not exist the required triple patterns which satisfy all mappings in Ω.

We thus conclude that $\textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOFwd}]) \in \mathbf{NP}$.

Item 2. The proof of this case is a relatively simple extension of item 1, and has thus been omitted. $\square$

Proposition 3.14. *The following decision problems are in* **PTIME***:*

1. $\textsc{RevEng}^{+}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOFwd}])$,

2. $\textsc{RevEng}^{\mathsf{E}}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOFwd}])$,

Proof. The algorithm for this problem is similar to that of Proposition 3.13, where guessing the tree structure is not necessary. $\square$

In the case where both positive and negative examples are provided, the strong previous results are not applicable. Indeed, the tightest notion of candidate that we may define here is that if the input is definable, then it will be definable by a polynomially sized compatible pattern tree. We turn to that result now.

Lemma 3.12. *Given an RDF graph D and two sets of mappings $\Omega, \bar{\Omega}$, if $(D, \Omega, \bar{\Omega}) \in \textsc{RevEng}^{\pm}(\mathrm{SP}[\mathsf{AOFwd}])$ then there exists a polynomially-sized pattern tree P such that $\Omega \subseteq [\![P]\!]_D$ and $\bar{\Omega} \cap [\![P]\!]_D = \emptyset$.*

Proof. Assume $(D, \Omega, \bar{\Omega}) \in \text{RevEng}^{\pm}(\text{SP}[\mathsf{AOFwd}])$, whereby there is a filter pattern tree $Q \in \text{SP}[\mathsf{AOFwd}]$ such that $\Omega \subseteq [\![Q]\!]_D$ and $\bar{\Omega} \cap [\![Q]\!]_D = \emptyset$. We will now show that there exists a pattern tree P with size polynomial in the size of the input $(D, \Omega, \bar{\Omega})$, such that $\Omega \subseteq [\![P]\!]_D$ and $\bar{\Omega} \cap [\![P]\!]_D$.

Firstly, let P have the same tree structure as Q, and let P contain the same triple pattern as Q, i.e. $P = (V_P, E_P, r_P, \lambda_P, \delta_P)$ such that $V_P = V_Q$, $E_P = E_Q$, and $\lambda_P = \lambda_Q$. Next, consider a node n in Q and the filter expression $\phi_n = \delta_Q(n)$. Build the following filter expression φ_n:

$$\varphi_n = \underbrace{\left[\bigvee_{\mu \in \Omega_n} \bigwedge_{?\mathtt{X} \in \mathrm{dom}_n(\mu)} ?\mathtt{X} = \mu(?\mathtt{X}) \right]}_{\varphi_n^1} \vee \varphi_n^2$$

where $\Omega_n = \{\mu \in \Omega \mid \mathsf{scope}_Q(n) \subseteq \mathrm{dom}(\mu)\}$ and $\mathrm{dom}_n(\mu) = \mathrm{dom}(\mu) \cap \mathsf{scope}_Q(n)$. Also, let $\text{NonMaximals} = \{\bar{\mu} \in \bar{\Omega} \mid \text{such that } \bar{\mu} \in \mathsf{partials}(D, Q) \text{ but } \bar{\mu} \text{ is not maximal in } \mathsf{partials}(D, Q)\}$. For each such $\bar{\mu} \in \text{NonMaximals}$ let $\max(\bar{\mu})$ be the (unique) mapping such that $\bar{\mu} \sqsubset \max(\bar{\mu})$ and $\max(\bar{\mu}) \in [\![Q]\!]_D$. For node n, let $N_n = \{\max(\bar{\mu}) \mid \bar{\mu} \in \text{NonMaximals} \text{ and } \mathsf{scope}_Q(n) \subseteq \mathrm{dom}(\max(\bar{\mu}))\}$. Define:

$$\varphi_n^2 = \bigvee_{\max(\bar{\mu}) \in N_n} \bigwedge_{?\mathtt{X} \in \mathrm{dom}_n(\max(\bar{\mu}))} ?\mathtt{X} = [\max(\bar{\mu})](?\mathtt{X}).$$

We show $\Omega \subseteq [\![P]\!]_D$. For this, consider $\mu \in \Omega$, and note that, as Q and P have the same tree structure, there is a reduction $P_\mu \trianglelefteq P$ such that $\mathrm{var}(P_\mu) = \mathrm{dom}(\mu)$ and $\mu(t) \in D$ for every triple pattern in P_μ (P has the same triple patterns as Q). Now consider node n in P_μ and note that the filter $\varphi_n = \delta_P(n)$ is such that $\mu \models \varphi_n$.

We show μ is maximal in $\mathsf{partials}(D, P)$. Assume that μ is not maximal in $\mathsf{partials}(D, P)$, whereby there is $\nu \in \mathsf{partials}(D, P)$ such that $\mu \sqsubset \nu$ and ν is maximal in $\mathsf{partials}(D, P)$. We claim this would imply ν is also a partial answer in Q. There is $Q_\nu \trianglelefteq Q$ such that $\mathrm{var}(Q_\nu) = \mathrm{dom}(\nu)$, and for every triple pattern t in Q_ν we have $\nu(t) \in D$ (as Q and P have the same triple patterns). For a node n in Q_ν and for $\phi_n = \delta_P(n)$, consider the following, of which at least one holds:

1. if $\nu \models \varphi_n^1$, then there exists a mapping $\sigma \in \Omega$ such that σ and ν coincide on all variables in $\mathsf{scope}_Q(n)$, whereby, as $\sigma \in [\![Q]\!]_D$, we also have that $\nu \models \delta_Q(n)$.

2. if $\nu \models \varphi_n^2$, then there is $\bar{\sigma} \in \bar{\Omega}$ such that $\max(\bar{\sigma})$ and ν coincide on all variables in $\mathsf{scope}_Q(n)$, whereby, as $\max(\bar{\sigma}) \in [\![Q]\!]_D$, we have $\nu \models \delta_Q(n)$.

Both cases contradict the fact that μ is maximal in $\mathsf{partials}(D, Q)$, and thus it must be the case that μ is maximal in $\mathsf{partials}(D, P)$. We conclude that $\Omega \subseteq [\![P]\!]_D$.

We must now show that $\bar{\Omega} \cap [\![P]\!]_D = \emptyset$. For this, consider a mapping $\bar{\mu} \in \bar{\Omega}$. We claim that $\bar{\mu} \notin [\![P]\!]_D$. There are two possibilities:

1. It may be the case that $\bar{\mu} \notin \mathsf{partials}(D, Q)$. Then there are three further possibilities, of which at least one must hold:

   (a) There does not exist a reduction $P_{\bar{\mu}}$ such that $\mathrm{var}(P_{\bar{\mu}}) = \mathrm{dom}(\bar{\mu})$, in which case $\bar{\mu} \notin [\![P]\!]_D$. In the following items we assume that such a reduction does exist.

   (b) There is a node n in $Q_{\bar{\mu}}$ and a triple pattern t in n such that $\bar{\mu}(t) \notin D$. In this case, as Q and P have the same triple patterns, we also have that $\mu \notin \mathsf{partials}(D, P)$, whereby $\bar{\mu} \notin [\![P]\!]_D$.

   (c) There is a node n in $Q_{\bar{\mu}}$ such that $\bar{\mu} \not\models \phi_n = \delta_Q(n)$. In this case, we claim that $\bar{\mu} \not\models \varphi_n = \delta_P(n)$ also. To see this, for the sake of contradiction, assume that $\bar{\mu} \models \varphi_n$, whereby there are two possibilities, of which at least one must hold:

      i. $\bar{\mu} \models \varphi_n^1$. In this case there is a mapping $\sigma \in \Omega$ such that $\bar{\mu}$ and σ coincide on every variable in $\mathsf{scope}_Q(n)$, whereby $\bar{\mu} \models \phi_n$, which is a contradiction.

      ii. $\bar{\mu} \models \varphi_n^2$. In this case there is a mapping $\max(\bar{\sigma})$ for some $\bar{\sigma} \in \bar{\Omega}$, such that $\mathsf{scope}_P(n) \subseteq \mathrm{dom}(\max(\bar{\sigma}))$, and $\max(\bar{\sigma})$ coincides with $\bar{\mu}$ on every variable in $\mathsf{scope}_P(n)$. As $\max(\bar{\sigma}) \in \mathsf{partials}(D, Q)$ we must have that $\max(\bar{\sigma}) \models \phi_n$, whereby $\bar{\mu} \models \phi_n$, which is a contradiction.

      By contradiction, then we have concluded that $\bar{\mu} \not\models \varphi_n$, and thus that $\bar{\mu} \notin [\![P]\!]_D$.

   In all three cases we have concluded that $\bar{\mu} \notin [\![P]\!]_D$, in particular $\bar{\mu} \notin \mathsf{partials}(D, P)$.

2. It may be the case that $\bar{\mu} \in \mathsf{partials}(D, Q)$ but $\bar{\mu}$ is not maximal. In this case recall that there exists the mapping $\max(\bar{\mu}) \in \mathsf{partials}(D, Q)$ such that $\bar{\mu} \sqsubset \max(\bar{\mu})$ and $\max(\bar{\mu})$ is maximal in $\mathsf{partials}(D, Q)$. Firstly, we note that $\bar{\mu} \in \mathsf{partials}(D, P)$, using arguments analogous to those used earlier in this proof. We claim that $\bar{\mu}$ is not maximal in $\mathsf{partials}(D, P)$.

For this, note that there is a node n in Q such that the reduction $Q_{\bar{\mu}}$ contains the parent of n but not n itself, whereas $\mathsf{scope}_Q(n) \subseteq \mathrm{dom}(\max(\bar{\mu}))$. In this node, but now looking at P, we have that $\varphi_n = \varphi_n^1 \vee \varphi_n^2$ is such that φ_n^2 contains a conjunct which corresponds to $\max(\bar{\mu})$, and thus $\max(\bar{\mu}) \models \varphi_n$. Note then that $\bar{\mu}$ may be extended to $\bar{\mu}'$ such that $\mathrm{dom}(\bar{\mu}') = \mathrm{dom}(\bar{\mu}) \cup \{?\mathtt{X} \mid n = \mathsf{top}_P(n)\}$, where for every variable in $\mathrm{dom}(\bar{\mu})$ we have that $\bar{\mu}'(?\mathtt{X}) = \bar{\mu}(?\mathtt{X})$, and for variables in $\{?\mathtt{X} \mid n = \mathsf{top}_P(n)\}$ we have $\bar{\mu}(?\mathtt{X}) = [\max(\bar{\mu})](?\mathtt{X})$. By construction, then, we have that $\bar{\mu}' \models \varphi_n$, and that $\bar{\mu}'$ satisfies the triple patterns of n as well, whereby $\bar{\mu}' \in \mathsf{partials}(D, P)$, allowing us to conclude that $\bar{\mu}$ is not maximal in $\mathsf{partials}(D, P)$.

The previous implies that $\bar{\Omega} \cap [\![P]\!]_D = \emptyset$. Finally, it is straightforward to see that the size of P is polynomial in the size of the input $(D, \Omega, \bar{\Omega})$. □

Proposition 3.15. *The following decision problems are in* Σ_2^p*:*

1. $\textsc{RevEng}^{\pm}(\mathrm{SP}[\mathsf{AOwd}])$,

2. $\textsc{RevEng}^{\pm}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$,

3. $\textsc{RevEng}^{\pm}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$,

4. $\textsc{RevEng}^{\pm}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$,

Proof. We proceed item by item.

Item 1. For $\textsc{RevEng}^{\pm}(\mathrm{SP}[\mathsf{AOwd}])$, it is possible to guess a polynomially-sized pattern tree P which mentions only variables in Ω and only constants in D, as the number of nodes in P is bounded by the number of variables in Ω, and the number of possible triple patterns in each node is also polynomially bounded by the size of the input). By Proposition 3.6, $\textsc{Verify}^{\pm}(\mathrm{SP}[\mathsf{AOwd}]) \in \mathbf{DP}$, and an oracle for this problem may be used.

Item 2. For $\textsc{RevEng}^{\pm}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$, the proof is similar (using Lemma 3.12) and has been omitted.

Item 3. For $\textsc{RevEng}^{\pm}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$, the result is trivial, as this is a special case of $\textsc{RevEng}^{\pm}(\mathrm{SP}[\mathsf{AOwd}])$.

Item 4. For $\textsc{RevEng}^{\pm}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$, the result is trivial, as this is a special case of $\textsc{RevEng}^{\pm}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$. □

Proposition 3.16. *The decision problem* $\textsc{RevEng}^{\pm}(\mathrm{SP}[\mathsf{AOFwd}])$ *is in* $\mathbf{NP}$.

Proof. Consider the following **NP** algorithm. Given an input $(D, \Omega, \bar{\Omega})$, as in the proof of Proposition 3.13, guess a tree structure compatible with Ω and build the corresponding canonical pattern tree $P = P^F_{\mathsf{can}}(D, \Omega)$. If this pattern tree is not well-formed, reject; otherwise, continue.

We next proceed to check the negative examples in polynomial time. For each negative mapping $\bar{\mu} \in \bar{\Omega}$, if there is no $P_{\bar{\mu}} \trianglelefteq P$ such that $\mathrm{var}(P_{\bar{\mu}}) = \mathrm{dom}(\bar{\mu})$ then $\bar{\mu}$ is not an answer, so skip to next $\bar{\mu} \in \bar{\Omega}$. Otherwise, assume there exists $P_{\bar{\mu}}$. Next, check whether $\bar{\mu}$ is a partial answer. If $\bar{\mu}$ is not a partial answer, then it is not an answer, so skip to next $\bar{\mu} \in \bar{\Omega}$. Otherwise, assume $\bar{\mu}$ is a partial answer.

Next, consider each leaf node n of $P_{\bar{\mu}}$ which has a child node m in P. Notice that there must exist a mapping $\mu' \in \Omega$ such that $\bar{\mu}(\mathbf{v}) = \mu'(\mathbf{v})$ for every variable $\mathbf{v} \in \mathsf{scope}_P(n)$ (due to the construction of the filter expressions). Now, there are two possibilities for μ': (i) if n is a leaf node of $P_{\mu'}$, then we conclude that there exists no expansion of $\bar{\mu}$ into children of n (in other words, there exists no mapping $\bar{\nu} \in \mathsf{partials}(D, P)$ such that $\bar{\mu} \sqsubset \bar{\nu}$ and some child n' of n is in $P_{\bar{\nu}}$), in which case we register this fact and proceed to the next leaf node of $P_{\bar{\mu}}$; (ii) if n is not a leaf node of $P_{\mu'}$, then $\bar{\mu}$ may expand in to the same child as μ', whereby $\bar{\mu}$ is not maximal and is thus not an answer, in which case we proceed to the next $\bar{\mu}$. If $\bar{\mu}$ cannot expand from any of its leaf nodes, then $\bar{\mu}$ is maximal and is therefore an answer, in which case we return False. If all negative examples are not answers, then return True.

If the previous algorithm accepts, notice that P is such that $\Omega \subseteq [\![P]\!]_D$ (see proof of Proposition 3.13). By the check done in polynomial time in the previous paragraph, we also conclude that $\bar{\Omega} \cap [\![P]\!]_D = \emptyset$.

Therefore, $\textsc{RevEng}^{\pm}(\mathrm{SP}[\mathsf{AOFwd}])$ is in **NP**. □

Proposition 3.17. *The decision problem* $\textsc{RevEng}^{\pm}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOFwd}])$ *is in* **NP**.

Proof. This is a trivial corollary of Proposition 3.16. □

3.4.6 Lower bounds on reverse engineering problems

In this section we provide complexity lower bounds for the reverse engineering decision problems, thus closing the exact complexities for all these problems, except for $\textsc{RevEng}^{\pm}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOFwd}])$, a result which is left open, to be addressed in future work.

Recall that for our smallest language, $\mathrm{SP}[\mathsf{A}]$, we have **PTIME** upper bounds for the positive and positive-and-negative examples reverse engineering problems, but only a **coNP** upper bound for the exact variant. Given that the problem of verifying that an SP[A] query exactly fits a set of example mappings is **coNP**-hard, it is not surprising that the corresponding reverse engineering problem is also **coNP**-hard:

Proposition 3.18. $\textsc{RevEng}^{\mathsf{E}}(\mathrm{SP}[\mathsf{A}])$ *is* **coNP**-*hard.*

Proof. We show this via a reduction from the complement of the $3-$COLOURABILITY problem. Given a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$, construct an instance (D_G, Ω_G) as follows:

- Let $D_G = D^0 \cup D^1 \cup D^2$, where:
  - $D^0 = \{(c_i, e, c_j), (c_i, s, c_j) \mid i \in [1,3], j \in [1,3], i \neq j\}$,
  - D^k, $k \in [1,2]$, has a triple (c^k_ℓ, e, c^k_m) for every edge $\{v_\ell, v_m\} \in E$ and a triple $(c^k_\ell, s, c^k_{\ell+1})$ for $\ell \in [1, n-1]$.
- Let Ω_G consist of mappings $\mu_k = [?\mathtt{X}_1 \mapsto c^k_1, \ldots, ?\mathtt{X}_n \mapsto c^k_n]$, $k \in [1,2]$.

It is straightforward to show that G is not 3-colourable if and only if there exists $P \in \mathrm{SP}[\mathsf{A}]$ such that $[\![P]\!]_{D_G} = \Omega_G$.

We now claim that if G is not 3-colourable, then there exists a $P \in \mathrm{SP}[\mathsf{A}]$ such that $[\![P]\!]_{D_G} = \Omega_G$. For this, assume that G is not 3-colourable, whereby for any function $\sigma : \{v_1, \ldots, v_n\} \to \{R, G, B\}$, σ is not a valid colouring for G. Consider, then, the graph pattern $P = \{(?\mathtt{X}_i, e, ?\mathtt{X}_j) \mid i, j \in [1,n], \{v_i, v_j\} \in E\} \cup \{(?\mathtt{X}_i, s, ?\mathtt{X}_{i+1}) \mid i \in [1, n-1]\}$.

Note that $\mu_1 \in [\![P]\!]_{D_G}$ by mapping P to the D^1 component. Similarly, $\mu_2 \in [\![P]\!]_{D_G}$. Note that any other mapping to either D^k will break the ordering required by the $(c^k_\ell, s, c^k_{\ell+1})$ triples. Also, any mapping to the D^0 component will have at least one pair which maps to the same colour, making such a mapping impossible. Therefore, $[\![P]\!]_{D_G} = \{\mu_1, \mu_2\}$.

Conversely, we claim that if there exists a $P \in \mathrm{SP}[\mathsf{A}]$ such that $[\![P]\!]_{D_G} = \Omega_G$ then G is not 3-colourable. For this, assume that there exists a $P \in \mathrm{SP}[\mathsf{A}]$ such that $[\![P]\!]_{D_G} = \Omega_G$ and note that as there is no mapping to the D^0 component, there exist no valid colourings of G. □

We move to lower bounds for SP[AOwd] for tree-like cases.

Proposition 3.19. *The following decision problems are* **coNP**-*hard:*

1. $\textsc{RevEng}^{+}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$,
2. $\textsc{RevEng}^{\mathsf{E}}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$,

3. $\textsc{RevEng}^{+}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$,

4. $\textsc{RevEng}^{\mathsf{E}}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$.

Proof. We proceed item by item.

Item 1. For $\textsc{RevEng}^{+}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$, we reduce from the complement of the 3$-$Colourability problem. Given a graph $G = (V, E)$ with $V = \{v_1, \dots, v_n\}$, construct an instance (D_G, Ω_G) as follows:

- Let D_G be the union of graphs D^0, D^1, D^2, with
  - $D^0 = \{(r, a, c_1)\} \cup \{(c_i, e, c_j), (c_i, s, c_j) \mid i, j \in [1, 3], i \neq j\}$,
  - For $k \in [1, 2]$, $D^k = \{(r^k, a, c^k_1)\} \cup \{(c^k_\ell, e, c^k_m) \mid \{v_\ell, v_m\} \in E\} \cup \{(c^k_\ell, s, c^k_{\ell+1}) \mid \ell \in [1, n-1]\}$.
- Let Ω_G consist of a mapping $\mu_0 = [?R \mapsto r, ?\mathtt{X}_1 \mapsto c_1]$ and $\mu_k = [?R \mapsto r^k, ?\mathtt{X}_1 \mapsto c^k_1, \dots, ?\mathtt{X}_n \mapsto c^k_n]$, for $k \in [1, 2]$.

We will now show that G is not 3-colourable if and only if there exists a SP[AOwd] pattern P with $\Omega_G \subseteq [\![P]\!]_{D_G}$.

Firstly, assume that G is not 3-colourable. In this case, consider the graph pattern $P = P_1 \,\mathbf{OPT}\, P_2$, where $P_1 = \{(?\mathtt{R}, a, ?\mathtt{X}_1)\}$ and $P_2 = \{(?\mathtt{X}_i, e, ?\mathtt{X}_j) \mid \{v_i, v_j\} \in E\} \cup \{(?\mathtt{X}_i, s, ?\mathtt{X}_{i+1}) \mid i \in [1, n-1]\}$. Note that all three mappings in Ω_G are partial answers, and that μ_1, μ_2 are clearly maximal, whereby $\mu_1, \mu_2 \in [\![P]\!]_{D_G}$. For μ_0, note that any subsuming mapping must map into the D^0 component, and that such a mapping would imply the existence of a valid 3-colouring. As such a colouring does not exist, we have that μ_0 is maximal, and thus is also an answer. We then have that $\Omega_G \subseteq [\![P]\!]_{D_G}$.

Next, assume that there exists a graph pattern $Q \in \mathrm{SP}[\mathsf{AOwd}]$ such that $\Omega_G \subseteq [\![Q]\!]_{D_G}$ and note that due to Lemma 3.5, the candidate graph pattern $P_{\mathsf{can}}(D_G, \Omega_G)$ works. Also, note that the graph pattern P defined in the previous paragraph *is* the candidate graph pattern. Thus, we have that μ_0 is a maximal partial answer, whereby there is no subsuming pattern mapping into D^0, and therefore, no valid colouring of G.

Item 2. The same reduction works for the $\textsc{RevEng}^{\mathsf{E}}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$ problem, as the previous proof actually produces an input such that $\Omega_G = [\![P]\!]_{D_G}$ holds.

Item 3. For $\textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$, informally we need each colour c_i to be already matched by each $?\mathtt{X}_\ell$. We can do this by introducing $3n$ more graphs, isomorphic to D^k and disjoint from all others, except that c_ℓ^k (in this copy) is merged with c_i.

More precisely, to show $\overline{3\text{-}\textsc{Colourability}} \leq_m^p \textsc{RevEng}^+(\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$, given an input $G = (V, E)$ to $\overline{3\text{-}\textsc{Colourability}}$, construct an input (D_G, Ω_G) to $\textsc{RevEng}^+(\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$ as follows:

- Let D_G be the union of graphs D^0, D^1, D^2, and $D^{(i,p)}$ for $i \in [1, n]$, $p \in [1, 3]$ with
  - D^0, D^1, and D^2 constructed as in item (1).
  - For each $i \in [1, n]$ and $p \in [1, 3]$, create $D^{(i,p)}$ as follows. Initially set $D^{(i,p)} = D^0 \cup \{(r^{(i,p)}, a, c_1^{(i,p)})\} \cup \{(c_\ell^{(i,p)}, e, c_m^{(i,p)}) \mid \{v_\ell, v_m\} \in E\} \cup \{(c_\ell^{(i,p)}, s, c_{\ell+1}^{(i,p)}) \mid \ell \in [1, n-1]\}$. Now, replace each $c_i^{(i,p)}$ by c_p.
- Let Ω_G consist of the following mappings:
  - μ_0, μ^1, and μ^2 constructed as in item (1).
  - For each $i \in [1, n]$ and $p \in [1, 3]$, initially set $\mu_{(i,p)} = [?R \mapsto r^{(i,p)}, ?\mathtt{X}_1 \mapsto c_1^{(i,p)}, \ldots, ?\mathtt{X}_n \mapsto c_n^{(i,p)}]$. Next, replace $c_i^{(i,p)}$ by c_p.

This construction works in the same way as before, now constructing the P_{can} which corresponds to $\text{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}]$, and invoking Lemma 3.6. The exact version of the problem is analogous. □

Next we consider the general cases of the problems and start with $\textsc{RevEng}^+$ $(\text{SP}[\mathsf{AOwd}])$.

Proposition 3.20. *The following problems are Σ_2^p-hard:*

1. $\textsc{RevEng}^+(\text{SP}[\mathsf{AOwd}])$,
2. $\textsc{RevEng}^\pm(\text{SP}[\mathsf{AOwd}])$,
3. $\textsc{RevEng}^{\mathsf{E}}(\text{SP}[\mathsf{AOwd}])$.

Proof. We proceed item by item.

Item 1. Consider the following problem, which is known to be Σ_2^p-complete:

$\exists\forall$3SAT

Input:	A quantified formula ϕ of the form $\exists\bar{x}\,\forall\bar{y}\,\neg\psi$, where $\bar{x}$ and $\bar{y}$ are tuples of variables and ψ is a conjunction of clauses of the form $(\ell_1 \vee \ell_2 \vee \ell_3)$, where ℓ_1, ℓ_2, ℓ_3 are either variables in $\bar{x} \cup \bar{y}$ or their negations.
Output:	True if ϕ is true, and False otherwise.

We will show that $\exists\forall 3\text{SAT} \leq_m^p \text{REVENG}^+(\text{SP}[\mathsf{AOwd}])$, that is, there is a polynomial-time many-one reduction from $\exists\forall 3\text{SAT}$ to $\text{REVENG}^+(\text{SP}[\mathsf{AOwd}])$.

Consider an instance $\phi = \exists\bar{x}\,\forall\bar{y}\,\neg\psi$ of $\exists\forall 3\text{SAT}$ problem. Without loss of generality we assume that clauses do not contain repeated literals.

We will construct (in polynomial-time) a corresponding instance (D_ϕ, Ω_ϕ) of the problem $\text{REVENG}^+(\text{SP}[\mathsf{AOwd}])$ such that ϕ is valid if and only if there exists a pattern $P \in \text{SP}[\mathsf{AOwd}]$ such that $\Omega_\phi \subseteq [\![P]\!]_{D_\phi}$.

We first construct the RDF graph D_ϕ as the union of the following disjoint parts, where $\gamma \in \psi$ means that a clause γ is a conjunct in ψ, and $\ell \in \gamma$ means that ℓ is a literal in γ:

$$D_\phi = D^0 \;\cup\; D^1 \;\cup\; D^2 \;\cup\; \bigcup_{x\in\bar{x}} D^x \;\cup\; \bigcup_{y\in\bar{y}} D^y \;\cup\; \bigcup_{\gamma\in\psi} D^\gamma$$
$$\cup \bigcup_{\gamma\in\psi,\;\ell\in\gamma,\;\ell\text{ uses }x\in\bar{x}} D^{(\gamma,\ell)} \;\cup\; D^*.$$

All the previous components, except the first and last, have very similar structure. For this reason, we next describe a graph $\hat{D}^s$ and then explain how to obtain D^s from $\hat{D}^s$ for different s in $\{1,2\}\cup\bar{x}\cup\bar{y}\cup\{\gamma \mid \gamma\in\psi\}$ and pairs (γ,ℓ) for literals ℓ in clauses γ. We also describe the D^0 and D^* components.

Let $\hat{D}^s = \hat{D}^s_b \cup \hat{D}^s_w \cup \hat{D}^s_{\bar{x}} \cup \hat{D}^s_u \cup \hat{D}^s_{\bar{y}} \cup \hat{D}^s_\psi \cup \hat{D}^s_C$ where the subparts are as follows:

- $\hat{D}^s_b = \{(A, A, b^s)\}$,
- $\hat{D}^s_w = \{(b^s, W, e^s)\}$,
- $\hat{D}^s_{\bar{x}} = \{(e^s, F_x, f^s_x), (e^s, T_x, t^s_x) \mid x \in \bar{x}\}$,
- $\hat{D}^s_u = \{(b^s, U, d^s)\}$,
- $\hat{D}^s_{\bar{y}} = \{(d^s, R_y, r^s_y) \mid y \in \bar{y}\}$,
- $\hat{D}^s_\psi = \{(d^s, R_\gamma, r^s_\gamma) \mid \gamma \in \psi\}$,
- $\hat{D}^s_C = \{(r^s_\gamma, \mathit{FArg}_i, f^s_x), (r^s_\gamma, \mathit{TArg}_i, t^s_x) \mid \gamma \in \psi$, literal $\ell_i, i \in [1,3]$, in γ uses variable $x \in \bar{x}\}$
  $\cup\, \{(r^s_\gamma, \mathit{Arg}^s_i, r^s_x) \mid \gamma \in \psi$, literal $\ell_i, i \in [1,3]$, in γ uses variable $y \in \bar{y}\}$.

We are ready to describe the parts of D_ϕ:

1. Let $D^0 = \{(A, A, b^0)\}$,

2. Let $D^1 = \hat{D}^1$ and $D^2 = \hat{D}^2$,

3. For each $x \in \bar{x}$, let $D^x = \hat{D}^x_b \cup \hat{D}^x_w \cup \hat{D}^x_{\bar{x}} \cup \hat{D}^x_\psi \cup \hat{D}^x_C$, removing the triples (e^x, F^x_x, f^x_x) and (e^x, T^x_x, t^x_x),

4. For each $y \in \bar{y}$, let $D^y = \hat{D}^y_b \cup \hat{D}^y_u \cup \hat{D}^y_{\bar{y}} \cup \hat{D}^y_\psi \cup \hat{D}^y_C$, removing the triple (d^y, R^y_y, f^y_y),

5. For each $\gamma \in \psi$, let $D^\gamma = \hat{D}^\gamma_b \cup \hat{D}^\gamma_u \cup \hat{D}^\gamma_{\bar{y}} \cup \hat{D}^\gamma_\psi \cup \hat{D}^\gamma_C$, removing the triple $(d^\gamma, R^\gamma_\gamma, f^\gamma_\gamma)$,

6. For each literal ℓ_i which has number i and mentions a variable $x \in \bar{x}$, in each clause γ, let $D^{(\gamma,\ell_i)} = \hat{D}^{(\gamma,\ell_i)}_b \cup \hat{D}^{(\gamma,\ell_i)}_u \cup \hat{D}^{(\gamma,\ell_i)}_{\bar{y}} \cup \hat{D}^{(\gamma,\ell_i)}_\psi \cup \hat{D}^{(\gamma,\ell_i)}_C$, removing the triples $(r^{(\gamma,\ell_i)}_\gamma, \mathit{FArg}^{(\gamma,\ell_i)}_i, f^{(\gamma,\ell_i)}_x)$ and $(r^{(\gamma,\ell_i)}_\gamma, \mathit{TArg}^{(\gamma,\ell_i)}_i, t^{(\gamma,\ell_i)}_x)$,

7. Let D^* consists of the following triples, assuming that each $\gamma \in \psi$ has 7 satisfying assignments $h^\gamma_1, \ldots, h^\gamma_7$:

   - (A, A, b^*),
   - (b^*, U, d^*),
   - (d^*, R_y, f^*_y) and (d^*, R_y, t^*_y) for each $y \in \bar{y}$,
   - $(d^*, R_\gamma, r^*_{(\gamma,1)}), \ldots, (d^*, R_\gamma, r^*_{(\gamma,7)})$ for each $\gamma \in \psi$,
   - $(r^*_{(\gamma,j)}, \mathit{VArg}^{(\gamma,1)}, v^*_{z_1}), \ldots, (r^*_{(\gamma,j)}, \mathit{VArg}^{(\gamma,3)}, v^*_{z_3})$ for satisfying assignments h^γ_j of each $\gamma = \ell_1 \vee \cdots \vee \ell_3$ in ψ, where
     - $z_i \in \bar{x} \cup \bar{y}$,
     - $v^*_{z_i} = \begin{cases} f^*_{z_i} & \text{if } h^\gamma_j(z_i) = \mathsf{False}, \\ t^*_{z_i} & \text{otherwise,} \end{cases}$
     - $\mathit{VArg}^{(\gamma,i)} = \begin{cases} \mathit{FArg}_i & \text{if } z_i \in \bar{x} \text{ and } h^\gamma_j(z_i) = \mathsf{False}, \\ \mathit{TArg}_i & \text{if } z_i \in \bar{x} \text{ and } h^\gamma_j(z_i) = \mathsf{True}, \\ \mathit{Arg}_i & \text{if } z_i \in \bar{y}. \end{cases}$

Having completed the RDF graph, we next define the set Ω_ϕ with the following mappings:

1. $\mu^0 = [?b \mapsto b^0]$,

2. For each $k = 1, 2$, the mapping μ^k assigns

$$\begin{array}{ll} ?b \mapsto b^k, ?e \mapsto e^k, ?d \mapsto d^k, & \\ ?f_x \mapsto f^k_x, ?t_x \mapsto t^k_x, & \text{for each } x \in \bar{x}, \\ ?r_y \mapsto r^k_y, & \text{for each } y \in \bar{y}, \\ ?r_\gamma \mapsto r^k_\gamma, & \text{for each } \gamma \in \psi, \end{array}$$

3. For each $x \in \bar{x}$, $\mu^x = [?b \to b^x, ?e \to e^x]$,

4. For each s in variables $\bar{y}$, clauses in ψ, pairs (ℓ_i, γ) with ℓ_i a literal in a clause γ that uses a variable in $\bar{x}$, or $*$, $\mu^s = [?b \to b^s, ?d \to d^s]$.

We now show that there exists an SP[AOwd] pattern P such that $\Omega_\phi \subseteq [\![P]\!]_{D_\phi}$ if and only if ϕ is true.

For the first direction, assume that ϕ is true, in which case there is an assignment $\sigma : \bar{x} \to \{F, T\}$ such that for every assignment $\delta : \bar{y} \to \{F, T\}$ it is the case that $\neg\psi$ holds. With this, consider the following graph pattern (represented as a pattern tree):

$$\begin{tikzcd}
& \{(A, A, ?\mathtt{b})\} & \\
\{(?\mathtt{b}, W, ?\mathtt{e})\} \arrow[ur] & & \{(?\mathtt{b}, U, ?\mathtt{d})\} \arrow[ul] \\
P_e \arrow[u] & & P_d \arrow[u]
\end{tikzcd}$$

where:

$$\begin{aligned}
P_d = &\{(?\mathtt{d}, R_y, ?\mathtt{r}_y) \mid y \in \bar{y}\} \\
&\cup \{(?\mathtt{d}, R_\gamma, ?\mathtt{r}_\gamma) \mid \gamma \in \psi\} \\
&\cup \{(?\mathtt{r}_\gamma, \mathit{TArg}_i, ?\mathtt{t}_x) \mid \gamma \in \psi, x \in \gamma[i] \text{ such that } \sigma(x) = T\} \\
&\cup \{(?\mathtt{r}_\gamma, \mathit{FArg}_i, ?\mathtt{f}_x) \mid \gamma \in \psi, x \in \gamma[i] \text{ such that } \sigma(x) = F\} \\
&\cup \{(?\mathtt{r}_\gamma, \mathit{Arg}_i, ?\mathtt{r}_y) \mid \gamma \in \psi, y \in \gamma[i]\}.
\end{aligned}$$

With the previous, notice that variables $?\mathtt{f}_x$ have not been mentioned for $x \in \bar{x}$ such that $\sigma(x) = F$, and similarly for $?\mathtt{t}_x$ such that $\sigma(x) = F$. Therefore:

$$\begin{aligned}
P_e = &\{(?\mathtt{e}, T_x, ?\mathtt{t}_x) \mid x \in \bar{x} \text{ such that } \sigma(x) = F\} \\
&\cup \{(?\mathtt{e}, F_x, ?\mathtt{f}_x) \mid x \in \bar{x} \text{ such that } \sigma(x) = T\}.
\end{aligned}$$

We claim that the previous graph pattern P is such that $\Omega \subseteq [\![P]\!]_{D_\phi}$.

- μ^0 is an answer, as it maps the root node of P to the D^0 component of D_ϕ (it is clearly maximal as well).

- For each $k \in [1, 2]$, μ^k is a (clearly maximal) partial answer. To see this, note that μ^k maps the three inner nodes of P to the D^k component. Also, μ^k maps the left leaf, P_e to the $D^k_{\bar{x}}$ sector of the D^k component (note that $D^k_{\bar{x}}$ contains *both* (e^k, F_x, f^k_x) and (e^k, T_x, t^k_x) for each x).

Now consider the right leaf, P_d. μ^k maps $(?\texttt{d}, R_y, ?\texttt{r}_y)$ and $(?\texttt{d}, R_\gamma, ?\texttt{r}_\gamma)$ triples to the $D^k_{\bar{y}}$ and $D^k_{\bar{\psi}}$ subcomponents, respectively. For each $\gamma \in \psi$, and $i \in [1,3]$, let x be the propositional variable mentioned in literal $\gamma[i]$; if $\sigma(x) = T$ then triple $(?\texttt{r}_\gamma, \mathit{TArg}_i, ?\texttt{t}_x)$ maps to $(r^k_\gamma, \mathit{TArg}_i, t^k_x)$; again, both options are present in the D^k_C subcomponent. The situation is similar for the $(?\texttt{r}_\gamma, \mathit{Arg}_i, ?\texttt{r}_y)$ triples. We conclude that $\mu^k \in [\![P]\!]_{D_\phi}$.

- For $x \in \bar{x}$, consider the mapping μ^x. The $?\texttt{b}$ and $?\texttt{e}$ nodes of P are clearly mapped to the D^x component, and hence μ^x is a partial answer. We must now show that μ^x is maximal. For the right inner node, there is clearly no expansion $(\mu^x)'$ which mentions $?\texttt{d}$, as the triple (b^x, U, d^x) does not exist.

  For the left leaf, P_e consider the $D^x_{\bar{x}}$ subcomponent. Note that a triple pattern $(?\texttt{e}, T_x, ?\texttt{t}_x) \in P_e$ maps to a triple $(e^x, T_x, ?)$ in $D^x_{\bar{x}}$. However, the only possibility is (e^x, T_x, t_x), which was explicitly excluded from the subcomponent. Therefore, μ^x is maximal.

- For the rest of the μ^s mappings, except μ^*, an argument similar to that of the previous item applies.

- For μ^*, it is clearly a partial answer, as the $?\texttt{b}$ and $?\texttt{d}$ inner nodes map to the D^* component. We must show that it is maximal. We proceed by contradiction, assuming there is an expansion ν^*, and show that this contradicts the fact that $\forall \bar{y} \neg \psi$ holds.

  Assume that there is a mapping ν^* such that $\mu^* \sqsubset \nu^*$ which is an answer, and note immediately that it does not mention $?\texttt{e}$ as the D^* component does not include (b^*, W, e^*). Therefore, ν^* must define the variables in P_d.

  - For $(?\texttt{d}, R_y, ?\texttt{r}_y)$ triples, we see that ν^* must map either $?\texttt{r}_y \mapsto t^*_y$ or $?\texttt{r}_y \mapsto f^*_y$, thus inducing an assignment $\delta : \bar{y} \to \{F, T\}$.
  - For $(?\texttt{d}, R_\gamma, ?\texttt{r}_\gamma)$ triples, ν^* must map $?\texttt{r}_\gamma \mapsto r^*_{(\gamma,j)}$ for some $j_\gamma \in [1,7]$, thus representing one of the 7 truth-states of γ.
  - For $(?\texttt{r}_\gamma, \mathit{TArg}_i, ?\texttt{t}_x)$ triple patterns, we have that x is mentioned in $\gamma[i]$ and $\sigma(x) = T$. Note that the previous item forces $?\texttt{r}_\gamma \mapsto r^*_{(\gamma,j_\gamma)}$, where $j_\gamma \in [1,7]$ is the truth-state of γ. Let $h^\gamma_{j_\gamma}$ be the satisfying assignment of clause γ associated to truth-state j_γ.

    For the first literal of γ and considering truth-state j_γ, there are two possibilities:

1. $\gamma[1]$ mentions $y \in \bar{y}$, a possibility which will be considered later, as it offers no information on triple pattern $(?\mathtt{r}_\gamma, \mathit{TArg}_i, ?\mathtt{t}_x)$.
2. $\gamma[1]$ mentions $x \in \bar{x}$. There are two further possibilities:
   (a) $h^\gamma_{j_\gamma}(x) = T$, whereby the triple $(r^*_{(\gamma,j_\gamma)}, \mathit{VArg}_1, v^*_{z_1})$ of D^* is such that $z_1 = x$, $\mathit{VArg}_1 = \mathit{TArg}_1$, and $v^*_{z_1} = t^*_x$. Thus, $(r^*_{(\gamma,j_\gamma)}, \mathit{TArg}_1, t^*_x) \in D^*$, implying that ν^* must map $?\mathtt{t}_x \mapsto t^*_x$.
   (b) $h^\gamma_{j_\gamma}(x) = F$, whereby the triple $(r^*_{(\gamma,j_\gamma)}, \mathit{VArg}_1, v^*_{z_1})$ of D^* is such that $z_1 = x$, $\mathit{VArg}_1 = \mathit{FArg}_1$, and $v^*_{z_1} = f^*_x$. Thus, $(r^*_{(\gamma,j_\gamma)}, \mathit{FArg}_1, f^*_x) \in D^*$. As ν^* cannot map $(?\mathtt{r}_\gamma, \mathit{TArg}_1, ?\mathtt{t}_x)$ to this triple, no information is gained.

The same holds for the second and third literal of γ.

– For $(?\mathtt{r}_\gamma, \mathit{FArg}_i, ?\mathtt{f}_x)$ triple patterns, and analogously to the previous item, we have that if, for literal i of γ there is a variable $x \in \bar{x}$ and truth-state j_γ is such that $h^\gamma_{j_\gamma}(x) = F$, then it must be the case that ν^* maps $?\mathtt{f}_x \mapsto f^*_x$; otherwise, no restriction is added.

– For $(?\mathtt{r}_\gamma, \mathit{Arg}_i, ?\mathtt{r}_y)$ triple patterns, we have that y is mentioned in $\gamma[i]$. Note that a previous item forces $?\mathtt{r}_\gamma \mapsto r^*_{(\gamma,j_\gamma)}$, where $j_\gamma \in [1,7]$ is the truth-state of γ. Let $h^\gamma_{j_\gamma}$ be the satisfying assignment of clause γ associated to truth-state j_γ.

For the first literal of γ and considering truth-state j_γ, there are two possibilities:

1. $\gamma[1]$ mentions $x \in \bar{x}$, a possibility which offers no information on triple pattern $(?\mathtt{r}_\gamma, \mathit{Arg}_i, ?\mathtt{r}_y)$.
2. $\gamma[1]$ mentions $y \in \bar{y}$. There are two further possibilities:
   (a) $h^\gamma_{j_\gamma}(y) = T$, whereby the triple $(r^*_{(\gamma,j_\gamma)}, \mathit{VArg}^{(\gamma,1)}_1, v^*_{z_1})$ of D^* is such that $z_1 = y$, $\mathit{VArg}^{(\gamma,1)} = \mathit{Arg}_1$, and $v^*_{z_1} = t^*_y$. Thus, $(r^*_{(\gamma,j_\gamma)}, \mathit{Arg}_1, t^*_y) \in D^*$, implying that ν^* must map $?\mathtt{r}_y \mapsto t^*_y$.
   (b) $h^\gamma_{j_\gamma}(y) = F$, whereby the triple $(r^*_{(\gamma,j_\gamma)}, \mathit{VArg}^{(\gamma,1)}, v^*_{z_1})$ of D^* is such that $z_1 = y$, $\mathit{VArg}^{(\gamma,1)} = \mathit{Arg}_1$, and $v^*_{z_1} = f^*_y$. Thus, $(r^*_{(\gamma,j_\gamma)}, \mathit{Arg}_1, f^*_y) \in D^*$, implying that ν^* must map $?\mathtt{r}_y \mapsto f^*_y$.

Crucially, note that item 2 immediately above implies a restriction on ν^* that *must agree* with the truth assignment induced previously on $\bar{y}$. This is the case if and only if the truth-states j_γ correspond to the values clauses γ would have under assignment $\sigma \cup \delta$. As we have assumed that ν^* exists and satisfies all triple pattern, have conclude that the induced assignment δ is such that every clause γ is in a truth-state, and therefore ψ evaluates to True under $\sigma \cup \delta$, whereby $\neg\psi$ evaluates to False. This contradicts the assumption that σ has the property that *for every* assignment of $\bar{y}$, $\neg\psi$ holds (i.e. evaluates to True), whereby we conclude that ν^* cannot exist, and therefore μ^* is an answer.

We have thus shown that $\Omega_\phi \subseteq [\![P]\!]_{D_\phi}$, and therefore that a realiser exists.

We now consider the other direction, assuming there exists a graph pattern $Q \in$ SP[AOwd] such that $\Omega_\phi \subseteq [\![Q]\!]_{D_\phi}$. We must show that ϕ is true.

As Q is compatible with Ω, we have that Q must have the following structure (where P_b, P_e, and P_d are possibly empty):

$$\begin{tikzcd}
 & \{(A, A, ?\mathtt{b})\} & \\
\{(?\mathtt{b}, W, ?\mathtt{e})\} \arrow[ur] & P_b \arrow[u] & \{(?\mathtt{b}, U, ?\mathtt{d})\} \arrow[ul] \\
P_e \arrow[u] & & P_d \arrow[u]
\end{tikzcd}$$

The triple patterns included above are the only triple patterns which satisfy all mappings in Ω, and therefore must be included. For P_b, note that there are no triple patterns which mention $?\mathtt{b}$, $?\mathtt{t}_x$, etc. (excluding $?\mathtt{e}$ and $?\mathtt{d}$) which hold for both μ^k mappings, whereby P_b is discarded. For P_e, note that the only triple patterns which may exist are of the form $(?\mathtt{e}, T_x, ?\mathtt{t}_x)$ and $(?\mathtt{e}, F_x, ?\mathtt{f}_x)$ (in order for both μ^k mappings to be partial answers), and similarly for P_d. While it will not be shown explicitly, it is the case that $Q = P$. From the arrangement of the $?\mathtt{t}_x$ and $?\mathtt{f}_x$ variables in Q, an assignment $sigma : \bar{x} \to \{F, T\}$ can be defined which serves as a witness that ϕ is valid.

We thus conclude that RevEng$^+$(SP[AOwd]) is Σ_2^p-hard.

Item 2. The previous proof immediately results in the Σ_2^p-hardness of RevEng$^\pm$ (SP[AOwd]), as the reduction corresponding to RevEng$^+$(SP[AOwd]) $\leq_m^p$ RevEng$^\pm$ (SP[AOwd]) may simply set $\bar{\Omega} = \emptyset$.

Item 3. In the previous proof, it is the case that ϕ is valid if and only if there exists $P \in$ SP[AOwd] such that $\Omega_\phi = [\![P]\!]_{D_\phi}$ (and not merely $\Omega_\phi \subseteq [\![P]\!]_{D_\phi}$), whereby the same construction proves the Σ_2^p-hardness of RevEng$^\mathsf{E}$(SP[AOwd]). □

A small modification of the previous proof (as in Proposition 3.20) gives us the same lower bounds for the case of SP[$\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}$]:

Proposition 3.21. *The following problems are Σ_2^p-hard:*

1. $\textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$,
2. $\textsc{RevEng}^{\pm}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$,
3. $\textsc{RevEng}^{\mathsf{E}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$,

Proof. This proof is a generalisation of the previous proof (i.e. the proof of Proposition 3.20) which uses the same techniques as the generalisations of the **coNP** lower bounds presented in the proof of Proposition 3.19, that is, a suitable amount of copies of the graph must be included, and the realiser graph pattern will include all appropriate filter expressions. □

We now have the **NP**-hardness for the problems with arbitrary filter:

Proposition 3.22. *The following problems are* **NP***-hard:*

1. $\textsc{RevEng}^{+}(\mathrm{SP}[\mathsf{AOFwd}])$,
2. $\textsc{RevEng}^{\pm}(\mathrm{SP}[\mathsf{AOFwd}])$,
3. $\textsc{RevEng}^{\mathsf{E}}(\mathrm{SP}[\mathsf{AOFwd}])$,

Proof. We show this by a reduction of the 3SAT problem. Given a formula ϕ of the form $\exists \bar{x}\, \psi$ with ψ a conjunction of clauses $\ell_1 \vee \ell_2 \vee \ell_3$, where each ℓ_i is either a variable from $\bar{x}$ or its negation, we construct an instance (D, Ω) as follows. Let D consists of disjoint parts D^0, D^u, D^w, D^1, D^2, where

- $D^0 = \{(A, A, b^0)\}$,
- $D^u = \{(A, A, b^u), (b^u, U, e^u)\}$,
- $D^w = \{(A, A, b^w), (b^w, W, d^w)\}$,
- D^i, $i = 1, 2$, has triples $(A, A, b^i), (b^i, U, e^i), (b^i, W, d^i)$, triples $(e^i, v^i_x, f^i_x), (e^i, v^i_x, t^i_x)$ for each $x \in \bar{x}$, and triples $(d^i, c^i_\gamma, s^i_1), \ldots, d^i, c^i_\gamma, s^i_3)$ for each clause $\gamma = \ell_1 \vee \cdots \vee \ell_3$ in ψ, where each s^i_j is f^i_x if $\ell_j = \neg x$ and is t^i_x if $\ell_j = x$.

Let Ω consists of mappings

- $\mu^0 = [?\mathtt{b} \mapsto b^0]$,
- $\mu^u = [?\mathtt{b} \mapsto b^u, ?\mathtt{e} \mapsto e^u]$,
- $\mu^w = [?\mathtt{b} \mapsto b^w, ?\mathtt{d} \mapsto d^w]$,

- μ^i, $i = 1, 2$, maps $?\mathtt{b} \mapsto b^i$, $?\mathtt{e} \mapsto e^i$, $?\mathtt{d} \mapsto d^i$, maps $?\mathtt{v}_x \mapsto v^i_x$, $?\mathtt{f}_x \mapsto f^i_x$, $?\mathtt{t}_x \mapsto t^i_x$ for every $x \in \bar{x}$ and maps $?\mathtt{c}_\gamma \mapsto c^i_\gamma$ for each γ in ψ.

It can be seen that ϕ is satisfiable if and only if there exists a pattern P in SP[AOFwd] such that $\Omega \subseteq [\![P]\!]_D$. Moreover, the same reduction works for $\textsc{RevEng}^{\pm}$ (taking $\bar{\Omega} = \emptyset$) and for $\textsc{RevEng}^{+}$. $\square$

The lower bound for $\textsc{RevEng}^{\pm}_{\mathsf{tree}}$(SP[$\mathsf{AOFwd}$]) is left open. Thus, this decision problem may yet be found to be in **PTIME** or ultimately found to be **NP**-hard.

Proposition 3.23. *The following problems are Σ^p_2-hard:*

1. $\textsc{RevEng}^{\pm}_{\mathsf{tree}}$(SP[$\mathsf{AOwd}$]),
2. $\textsc{RevEng}^{\pm}_{\mathsf{tree}}$(SP[$\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}$]).

Proof. For $\textsc{RevEng}^{\pm}_{\mathsf{tree}}$(SP[$\mathsf{AOwd}$]) the proof is similar to that of Proposition 3.20, with the main difference being that now the constructed Ω_ϕ must be tree-like. We proceed to showing that $\exists\forall 3\textsc{Sat} \leq^p_m \textsc{RevEng}^{\pm}_{\mathsf{tree}}$(SP[$\mathsf{AOwd}$]).

Consider an instance $\phi = \exists \bar{x}\, \forall \bar{y}\, \neg\psi$ of the $\exists\forall 3\textsc{Sat}$ problem. Without loss of generality we assume that clauses do not contain repeated literals.

We will construct (in polynomial-time) a corresponding instance $(D_\phi, \Omega_\phi, \bar{\Omega}_\phi)$ of the problem $\textsc{RevEng}^{\pm}_{\mathsf{tree}}$(SP[$\mathsf{AOwd}$]) such that ϕ is valid if and only if there exists a pattern $P \in$ SP[AOwd] such that $\Omega_\phi \subseteq [\![P]\!]_{D_\phi}$ and $\bar{\Omega}_\phi \cap [\![P]\!]_{D_\phi} = \emptyset$.

We first construct the RDF graph D_ϕ as the union of the following disjoint parts, where $\gamma \in \psi$ means that a clause γ is a conjunct in ψ, and $\ell \in \gamma$ means that ℓ is a literal in γ:

$$D_\phi = D^0 \,\cup\, D^1 \,\cup\, D^2 \,\cup\, \bigcup_{x \in \bar{x}} D^x \,\cup\, \bigcup_{y \in \bar{y}} D^y \,\cup\, \bigcup_{\gamma \in \psi} D^\gamma$$
$$\cup \bigcup_{\gamma \in \psi, \ell \in \gamma} D^{(\gamma,\ell)} \,\cup\, D^* \,\cup\, D^\neg \,\cup \bigcup_{i,j \in [1,n+p+q]} D^{(+,i,j)}.$$

Several of the previous components have very similar structure. For this reason, we next describe a graph $\hat{D}^s$ and then explain how to obtain D^s from $\hat{D}^s$ for different s in $\{1,2\} \cup \bar{x} \cup \bar{y} \cup \{\gamma \mid \gamma \in \psi\}$ and pairs (γ, ℓ). We describe the D^0, D^*, and $D^\neg$ components separately.

Define the subcomponents $\hat{D}^s_b$, $\hat{D}^s_{\bar{x}}$, $\hat{D}^s_{\bar{x},2}$, $\hat{D}^s_{\bar{y}}$, $\hat{D}^s_\psi$, $\hat{D}^s_C$, $\hat{D}^s_{C,2}$, and $\hat{D}^s_X$ as follows, letting $\hat{D}^s = \hat{D}^s_b \cup \hat{D}^s_{\bar{x},2} \cup \hat{D}^s_{\bar{y}} \cup \hat{D}^s_\psi \cup \hat{D}^s_{C,2} \cup \hat{D}^s_X$ for ease of notation:

- $\hat{D}^s_b = \{(A, A, b^s)\}$,

- $\hat{D}^s_{\bar{x}} = \{(b^s, F_x, f^s_x), (b^s, T_x, t^s_x) \mid x \in \bar{x}\}$,
- $\hat{D}^s_{\bar{x},2} = \{(b^s, F_x, r^s_x), (b^s, T_x, r^s_x) \mid x \in \bar{x}\}$,
- $\hat{D}^s_{\bar{y}} = \{(b^s, R_y, r^s_y) \mid y \in \bar{y}\}$,
- $\hat{D}^s_{\psi} = \{(b^s, R_\gamma, r^s_\gamma) \mid \gamma \in \psi\}$,
- $\hat{D}^s_C = \{(r^s_\gamma, \mathit{FArg}_i, f^s_x), (r^s_\gamma, \mathit{TArg}_i, t^s_x) \mid \gamma \in \psi$, literal $\ell_i, i \in [1,3]$, in γ uses variable $x \in \bar{x}\}$
  $\cup \{(r^s_\gamma, \mathit{Arg}_i, r^s_y) \mid \gamma \in \psi$, literal $\ell_i, i \in [1,3]$, in γ uses variable $y \in \bar{y}\}$.
- $\hat{D}^s_{C,2} = \{(r^s_\gamma, \mathit{FArg}_i, r^s_x), (r^s_\gamma, \mathit{TArg}_i, r^s_x) \mid \gamma \in \psi$, literal $\ell_i, i \in [1,3]$, in γ uses variable $x \in \bar{x}\}$
  $\cup \{(r^s_\gamma, \mathit{Arg}_i, r^s_y) \mid \gamma \in \psi$, literal $\ell_i, i \in [1,3]$, in γ uses variable $y \in \bar{y}\}$.
- Let $m = n + p + q$ and define the sequence $\bar{\alpha} = \alpha_1, \ldots, \alpha_m$, where:

$$\begin{aligned} \alpha_1, \ldots, \alpha_n &= \gamma_1, \ldots, \gamma_n, \\ \alpha_{n+1}, \ldots, \alpha_{n+p} &= x_1, \ldots, x_p, \\ \alpha_{n+p+1}, \ldots, \alpha_{n+p+q} &= y_1, \ldots, y_q. \end{aligned}$$

  With the previous, for each $i, j \in [1, m]$, let $D^s_X = \{(r^s_{\alpha_i}, X, r^s_{\alpha_j})\}$.

We are ready to describe the parts of D_ϕ:

1. Let $D^0 = \hat{D}^0_b = \{(A, A, b^0)\}$,
2. For $k = 1, 2$, let $D^k = \hat{D}^k$,
3. For each $x \in \bar{x}$, let $D^x = \hat{D}^x$, removing the triples (b^x, F^x_x, r^x_x) and (b^x, T^x_x, r^x_x),
4. For each $y \in \bar{y}$, let $D^y = \hat{D}^y$, removing the triple (b^y, R^y_y, r^y_y),
5. For each $\gamma \in \psi$, let $D^\gamma = \hat{D}^\gamma$, removing the triple $(b^\gamma, R^\gamma_\gamma, r^\gamma_\gamma)$,
6. For each clause $\gamma \in \psi$ and each literal $\ell_i \in \gamma$ in position $i \in [1,3]$, mentioning a variable $x \in \bar{x}$, let $D^{(\gamma,\ell_i)} = \hat{D}^{(\gamma,\ell_i)}$, removing the triples $(r^{(\gamma,\ell_i)}_\gamma, \mathit{FArg}^{(\gamma,\ell_i)}_i, f^{(\gamma,\ell_i)}_x)$ and $(r^{(\gamma,\ell_i)}_\gamma, \mathit{TArg}^{(\gamma,\ell_i)}_i, t^{(\gamma,\ell_i)}_x)$,
7. For each clause $\gamma \in \psi$ and each literal $\ell_i \in \gamma$ in position $i \in [1,3]$, mentioning a variable $y \in \bar{y}$, let $D^{(\gamma,\ell_i)} = \hat{D}^{(\gamma,\ell_i)}$, removing the triple $(r^{(\gamma,\ell_i)}_\gamma, \mathit{Arg}^{(\gamma,\ell_i)}_i, f^{(\gamma,\ell_i)}_x)$,

8. Let D^* consists of the following triples, assuming that each $\gamma \in \psi$ has 7 satisfying assignments $h_1^\gamma, \ldots, h_7^\gamma$:

   - (A, A, b^*),
   - (b^*, T_x^*, r_x^*) and (b^*, F_x^*, r_x^*) for each $x \in \bar{x}$,
   - (b^*, R_y, f_y^*) and (b^*, R_y, t_y^*) for each $y \in \bar{y}$,
   - $(b^*, R_\gamma, r^*_{(\gamma,1)}), \ldots, (b^*, R_\gamma, r^*_{(\gamma,7)})$ for each $\gamma \in \psi$,
   - $(r^*_{(\gamma,j)}, \mathit{VArg}^{(\gamma,1)}, v^*_{z_1}), \ldots, (r^*_{(\gamma,j)}, \mathit{VArg}^{(\gamma,3)}, v^*_{z_3})$ for satisfying assignments h_j^γ of each $\gamma = \ell_1 \vee \cdots \vee \ell_3$ in ψ, where $z_i \in \bar{x} \cup \bar{y}$ and

$$\mathit{VArg}^{(\gamma,i)}, v^*_{z_i} = \begin{cases} \mathit{FArg}_i, f_x^* & \text{if } z_i = x \in \bar{x}, h_j^\gamma(x) = \mathsf{False}, \\ \mathit{TArg}_i, t_x^* & \text{if } z_i = x \in \bar{x}, h_j^\gamma(x) = \mathsf{True}, \\ \mathit{Arg}_i, f_y^* & \text{if } z_i = y \in \bar{y}, h_j^\gamma(y) = \mathsf{False}, \\ \mathit{Arg}_i, t_y^* & \text{if } z_i = y \in \bar{y}, h_j^\gamma(y) = \mathsf{True}, \end{cases}$$

9. Let $D^\neg = \hat{D}_b^\neg \cup \hat{D}_{\bar{x}}^\neg \cup \hat{D}_{\bar{y}}^\neg \cup \hat{D}_{\psi}^\neg \cup \hat{D}_C^\neg \cup \hat{D}_X^\neg$,

10. For $i, j \in [1, m]$, let $D^{(+,i,j)} = \hat{D}_X^{(+,i,j)}$, removing the triple $(r^+_{\alpha_i}, X, r^+_{\alpha_j})$.

Having completed the RDF graph, we next define the set Ω_ϕ with the following mappings:

1. $\mu^0 = [?\mathtt{b} \mapsto b^0]$,

2. For each $k = 1, 2$, the mapping μ^k assigns

$$\begin{array}{ll} ?\mathtt{b} \mapsto b^k, & \\ ?\mathtt{r}_x \mapsto r_x^k, & \text{for each } x \in \bar{x}, \\ ?\mathtt{r}_y \mapsto r_y^k, & \text{for each } y \in \bar{y}, \\ ?\mathtt{r}_\gamma \mapsto r_\gamma^k, & \text{for each } \gamma \in \psi, \end{array}$$

3. For each $x \in \bar{x}$, $\mu^x = [?\mathtt{b} \to b^x]$,

4. For each $y \in \bar{y}$, $\mu^y = [?\mathtt{b} \to b^y]$,

5. For each $\gamma \in \psi$, $\mu^\gamma = [?\mathtt{b} \to b^\gamma]$,

6. For each pair (γ, ℓ_i), $\mu^{(\gamma,\ell_i)} = [?\mathtt{b} \to b^{(\gamma,\ell_i)}]$,

7. For each pair $i, j \in [1, m]$, $\mu^{(+,i,j)} = [?\mathtt{b} \to b^{(+,i,j)}]$,

8. $\mu^* = [?\mathtt{b} \to b^*]$.

Finally, the set of negative examples is $\bar{\Omega}_\phi = \{\bar{\mu}\}$ with $\bar{\mu} = [?\mathsf{b} \mapsto b^\neg]$.

Note that in this case the structure of Ω_ϕ is as follows:

$$\begin{tikzcd}
\{\mu^0, \mu^x, \dots\}^{?\mathsf{b}} \\
\{\mu^1, \mu^2\}^{?\mathtt{t}_x,\dots} \arrow[u]
\end{tikzcd}$$

We must show that ϕ is True if and only if $(D_\phi, \Omega_\phi, \bar{\Omega}_\phi) \in \textsc{RevEng}^{\pm}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$.

For the first direction, assume that ϕ is True, whereby there exists the assignment $\sigma : \bar{x} \to \{F, T\}$ with the usual properties. Construct a realiser P as follows. Firstly, P will have the tree structure:

$$\begin{tikzcd}
\{(A, A, ?\mathsf{b})\} \\
P_1 \arrow[u]
\end{tikzcd}$$

where P_1 consists of the following triple patterns:

- $(?\mathsf{b}, T_x, ?\mathtt{r}_x)$ for each $x \in \bar{x}$ such that $\sigma(x) = T$,
- $(?\mathsf{b}, F_x, ?\mathtt{r}_x)$ for each $x \in \bar{x}$ such that $\sigma(x) = F$,
- $(?\mathsf{b}, R_y, ?\mathtt{r}_y)$ for each $y \in \bar{y}$,
- $(?\mathsf{b}, R_\gamma, ?\mathtt{r}_\gamma)$ for each $\gamma \in \psi$,
- $(?\mathtt{r}_\gamma, \mathit{TArg}_i, ?\mathtt{r}_x)$ for each $\gamma \in \psi$, $i \in [1, 3]$, and $x \in \bar{x}$ such that x is mentioned in the i-th position of γ and $\sigma(x) = T$,
- $(?\mathtt{r}_\gamma, \mathit{FArg}_i, ?\mathtt{r}_x)$ for each $\gamma \in \psi$, $i \in [1, 3]$, and $x \in \bar{x}$ such that x is mentioned in the i-th position of γ and $\sigma(x) = F$,
- $(?\mathtt{r}_\gamma, \mathit{Arg}_i, ?\mathtt{r}_y)$ for each $\gamma \in \psi$, $i \in [1, 3]$, and $y \in \bar{y}$ such that y is mentioned in the i-th position of γ,
- $(?\mathtt{r}_{\alpha_i}, X, ?\mathtt{r}_{\alpha_j})$ for each $i, j \in [1, m]$,

We claim that P is such that $\Omega \subseteq [\![P]\!]_{D_\phi}$ and $\bar{\Omega} \cap [\![P]\!]_{D_\phi} = \emptyset$.

Firstly, note that all mappings $\mu \in \Omega$ are partial answers of P over D_ϕ, by construction, whereby we turn to their maximality.

- μ^b is maximal due to the fact that D^0 only has one triple,
- μ^1 and μ^2 are trivially maximal,

- For a given $x \in \bar{x}$, μ^x is maximal due to the fact that the root node of P, $\{(A, A, ?\mathtt{b})\}$ may only be mapped to the D^x subcomponent. In the second node, P_1, the triple pattern $(?\mathtt{b}, T_x, ?\mathtt{r}_x)$ (or $(?\mathtt{b}, F_x, ?\mathtt{r}_x)$, depending on the value of $\sigma(x)$) cannot be mapped, as D^x is explicitly missing the triple (b^x, T_x, r^x_x),
- The μ^y, μ^γ, $\mu^{(\gamma,\ell_i)}$, and $\mu^{(+,i,j)}$ mappings are maximal for the same reason as the previous item.
- μ^* is maximal for the same reasons as described in the proof of Proposition 3.20: an answer ν^* such that $\mu^* \sqsubset \nu^*$ would contradict the fact that ϕ is True.

Finally, $\bar{\mu}$ must not be an answer. As $\bar{\mu}$ is a partial answer (mapping the root node of P to the $D^\neg$ component), we claim that it is not maximal. This is clearly the case, as $D^\neg$ contains copies of all necessary triples. Therefore,

In the other direction, we now claim that there exists a pattern tree $Q \in \mathrm{SP}[\mathsf{AOwd}]$ such that $\Omega_\phi \subseteq [\![Q]\!]_{D_\phi}$ and $\bar{\Omega}_\phi \cap [\![Q]\!]_{D_\phi} = \emptyset$ and we must prove that ϕ is True.

Due to that fact that Q is compatible with Ω_ϕ, we have that Q must have a root node $r_Q \in V(Q)$ with the triple pattern $(A, A, ?\mathtt{b})$ (this is the only triple pattern that is satisfied by all mappings in Ω_ϕ). Now, we claim that Q has only one child node. For this, assume there exists a node $n \in V(Q)$ such that $(r_Q, n) \in E(Q)$, and a different node $m \in E(Q)$. Therefore, there are two variables $?\mathtt{r}_{\alpha_i}$ and $?\mathtt{r}_{\alpha_j}$ such that $\mathsf{top}_Q(?\mathtt{r}_{\alpha_i}) = n$ and $\mathsf{top}_Q(?\mathtt{r}_{\alpha_j}) = m$. Note that variable $?\mathtt{r}_{\alpha_j}$ corresponding to m is out of scope in n, whereby it may not be mentioned in that node. Therefore, the triple pattern $(?\mathtt{r}_{\alpha_i}, X, ?\mathtt{r}_{\alpha_j})$ is not included in n.

Now, note that the mapping $\mu^{(+,i,j)}$ is a partial answer of Q and note that the subcomponent $D^{(+,i,j)}$ contains all possible triples except $(r^{(+,i,j)}_{\alpha_i}, X, r^{(+,i,j)}_{\alpha_j})$. Therefore, the only valid triple pattern which is not satisfied by $\mu^{(+,i,j)}$ is $(?\mathtt{r}_{\alpha_i}, X, ?\mathtt{r}_{\alpha_j})$, which is not included in n. Therefore, there is a subsuming mapping $\nu^{(+,i,j)}$ which covers node n, which contradicts the fact that $\mu^{(+,i,j)}$ is maximal. We thus conclude that Q has only one node other than the root, and we name this node Q_1.

Using a similar argument, we see that the maximality of μ^y for each $y \in \bar{y}$ implies that triple pattern $(?\mathtt{b}, R_y, ?\mathtt{r}_y)$ must be included in Q, and similarly for all $(?\mathtt{b}, R_\gamma, ?\mathtt{r}_\gamma)$ and $(?\mathtt{r}_\gamma, Arg_i, ?\mathtt{r}_y)$ triple patterns.

For μ^x mappings, we conclude that either $(?\mathtt{b}, T_x, ?\mathtt{r}_x)$, $(?\mathtt{b}, F_x, ?\mathtt{r}_x)$, or both must be present, and $\mu^{(\gamma,\ell_i)}$ mappings similarly show us that either $(?\mathtt{b}, TArg_i, ?\mathtt{r}_x)$, $(?\mathtt{b}, FArg_i, ?\mathtt{r}_x)$, or both must be present.

	SP[A]	SP[AOwd]	SP[AOF$_{\wedge,=,\neq}$wd]	SP[AOFwd]
RevEng$^+$	in **PTIME**	Σ_2^p-c	Σ_2^p-c	**NP**-c
RevEng$^{\mathsf{E}}$	**coNP**-c	Σ_2^p-c	Σ_2^p-c	**NP**-c
RevEng$^\pm$	in **PTIME**	Σ_2^p-c	Σ_2^p-c	**NP**-c
RevEng$^+_{\mathsf{tree}}$	in **PTIME**	**coNP**-c	**coNP**-c	in **PTIME**
RevEng$^{\mathsf{E}}_{\mathsf{tree}}$	**coNP**-c	**coNP**-c	**coNP**-c	in **PTIME**
RevEng$^\pm_{\mathsf{tree}}$	in **PTIME**	Σ_2^p-c	Σ_2^p-c	in **NP**

Table 3.3: Complexity of reverse engineering problems.

We now turn to $\bar{\mu}$, which is not an answer of Q, implying that it is not maximal, whereby there exists a mapping $\bar{\nu}$ such that $\bar{\mu} \sqsubset \bar{\nu}$ and $\nu \in [\![Q]\!]_{D_\phi}$, noting that $\bar{\nu}$ must map Q_1 into the $D^\neg$ subcomponent.

We now claim that, given $x \in \bar{x}$, either $(?\mathtt{b}, T_x, ?\mathtt{r}_x)$ or $(?\mathtt{b}, F_x, ?\mathtt{r}_x)$ are present in Q_1, but not both. For the sake of contradiction, assume that both are included. This is a contradiction as $(?\mathtt{b}, T_x, ?\mathtt{r}_x)$ implies that $\bar{\nu}$ maps $?\mathtt{r}_x \mapsto r_x^\neg$, while $(?\mathtt{b}, F_x, ?\mathtt{r}_x)$ implies that $\bar{\nu}$ maps $?\mathtt{r}_x \mapsto r_x^\neg$, which cannot both hold. Therefore, either $(?\mathtt{b}, T_x, ?\mathtt{r}_x)$ or $(?\mathtt{b}, F_x, ?\mathtt{r}_x)$ are in Q_1, but not both. In fact, exactly one of these triple patterns are included, for each $x \in \bar{x}$. Using a similar argument, we have that for each combination of $\gamma \in \psi$, $i \in [1, 3]$, and $x \in \bar{x}$ such that γ uses x at position i exactly one of $(?\mathtt{b}, \mathit{TArg}_i, ?\mathtt{r}_x)$ and $(?\mathtt{b}, \mathit{FArg}_i, ?\mathtt{r}_x)$ are present. Finally, we may also conclude that for each $x \in \bar{x}$, $(?\mathtt{b}, T_x, ?\mathtt{r}_x)$ is present if and only if $(?\mathtt{b}, \mathit{TArg}_i, ?\mathtt{r}_x)$, and similarly for $(?\mathtt{b}, F_x, ?\mathtt{r}_x)$ and $(?\mathtt{b}, \mathit{FArg}_i, ?\mathtt{r}_x)$.

With the previous, we may define an assignment $\sigma : \bar{x} \to \{F, T\}$ such that for each $x \in \bar{x}$, $\sigma(x) = T$ if and only if $(?\mathtt{b}, T_x, ?\mathtt{r}_x)$ is in Q_1.

Finally, the maximality of μ^* works as usual, allowing us to conclude that ϕ is True.

Item 2. The generalisation to SP[AOF$_{\wedge,=,\neq}$wd] is as usual, and has been omitted. □

Finally, Theorem 3.2 summarises the results of this section:

Theorem 3.2. *The complexity results in Table 3.3 hold.*

3.5 Algorithms for Reverse Engineering

In this section we discuss the algorithms that can be used to solve the reverse engineering problems RevEng$^+_{\mathsf{tree}}$(SP[AOwd]) and RevEng$^+_{\mathsf{tree}}$ (SP[AOF$_{\wedge,=,\neq}$wd]). Although the core ideas for these algorithms were presented in the complexity upper bounds results of Section 3.4.2, we now describe the algorithms and discuss modifications for returning more desirable reverse engineered queries.

Given an input (D, Ω), if Ω is an arbitrary set of mappings (i.e. not necessarily tree-like), then the structure $\mathcal{C}(\Omega)$ (as defined in Section 3.4.2) does not clearly indicate an **OPT** structure for the reverse engineered query, in which case the algorithm must iterate over all possible **OPT** structures and construct a candidate query for each such structure. In what follows, we focus our attention on the tree-like cases ($\textsc{RevEng}^{+}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOwd}])$ and $\textsc{RevEng}^{+}_{\mathsf{tree}}(\mathrm{SP}[\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}])$), where the structure $\mathcal{C}(\Omega)$ of Ω immediately gives the **OPT**-structure, from which we construct the unique candidate query (with the additional requirement of deciding which triple patterns to include in the query) in polynomial time. The complexity lower bound results, however, indicate that even once the candidate query P is constructed, we must check that $\Omega \subseteq [\![P]\!]_D$ actually holds (this is due to the fact that the candidate query has the property that $(D, \Omega) \in \textsc{RevEng}^{+}$ if and only if $\Omega \subseteq [\![P]\!]_D$). However, this clean separation between the polynomial time construction of the candidate query and the **coNP**-complete verification of said query will allow us to use a state-of-the-art SPARQL engine for the final step.

Algorithm 2 outlines a framework for solving the $\textsc{RevEng}^{+}$ $(\mathrm{SP}[\mathsf{AOwd}])$ decision problem. Firstly, if Ω is not consistent then there is no query $P \in \mathrm{SP}[\mathsf{AOF}]$ such that $\Omega \subseteq [\![P]\!]_D$ (see Section 3.2.1). Function `CheckCanon`(D, Ω) simply verifies that the candidate reverse-engineered query P is in $\mathrm{SP}[\mathsf{AOwd}]$ (note that if the candidate query is not in $\mathrm{SP}[\mathsf{AOwd}]$, then we have that the input is not definable). Function `BuildCanon`(D, Ω) uses the structure $\mathcal{C}(\Omega)$ to build the candidate query P, and corresponds to the recursive function $P = P_{\mathsf{can}}(D, \Omega) = P_{\mathsf{can}}(\Omega, D, \Omega)$ which was described in detail in Equation 3.1 of Section 3.4.2. Notice that for each leaf element Λ in the structure $\mathcal{C}(\Omega)$ we set $P_{\mathsf{can}}(\Lambda, D, \Omega) = \mathrm{atype}(D, \Omega_{\mathsf{VarsOf}}(\Lambda))$, which by definition includes *all existing* triple patterns which satisfy the mappings in $\Omega_{\mathsf{VarsOf}(\Lambda)}$. For this reason, we call this the *maximal algorithm.* Finally, once built, the candidate query P must be checked, returning P if $\Omega \subseteq [\![P]\!]_D$ and null otherwise. This final check can be delegated to an external SPARQL engine.

Now we consider a greedy version of the algorithm, which differs from the maximal algorithm only in the construction of the candidate P. Intuitively, for an element $\Lambda \in \mathcal{C}(\Omega)$ it is not necessary for $P_{\mathsf{can}}(\Lambda, D, \Omega)$ to include *all* triple patterns in $\mathrm{atype}(D, \Omega_{\mathsf{VarsOf}(\Lambda)})$—merely *enough* triple patterns from $\mathrm{atype}(D, \Omega_{\mathsf{VarsOf}(\Lambda)})$ to ensure that the positive examples $\mu \in \Omega$ will in fact be answers $\mu \in [\![P]\!]_D$. More precisely, for each element $\Lambda \in \mathcal{C}(\Omega)$ we define a relaxed $P_{\mathsf{can}}^{\text{greedy}}(\Lambda, D, \Omega)$, which must be a subset of atype$(D, \Omega_{\mathsf{VarsOf}(\Lambda)})$ such that (i) every variable $?\mathtt{X} \in \mathsf{VarsOf}(\Lambda)$ is mentioned in at least one triple pattern of $P_{\mathsf{can}}^{\text{greedy}}(\Lambda, D, \Omega)$ (this is a requirement for P to be in SP[AOwd]), and (ii) if Λ' is the parent of Λ in $\mathcal{C}(\Omega)$ (i.e. the unique minimal superset of Λ in $\mathcal{C}(\Omega)$), then for each mapping $\mu \in \Lambda' \setminus \Lambda$, for every ν such that $\mu \sqsubset \nu$ there must exist a triple pattern $t \in P_{\mathsf{can}}^{\text{greedy}}(\Lambda, D, \Omega)$ such that $\nu(t) \notin D$. The previous condition (ii) effectively ensures that μ is a *maximal* partial answer.

We may now implement $\mathtt{BuildCanon}(D, \Omega)$ to find, for each element $\Lambda \in \mathcal{C}(\Omega)$, all triple patterns $t \in$ atype$(D, \Omega_{\mathsf{VarsOf}(\Lambda)})$, adding them to $P_{\mathsf{can}}^{\text{greedy}}(\Lambda, D, \Omega)$ until all variables $?\mathtt{X} \in \mathsf{VarsOf}(\Lambda)$ have been mentioned at least once, and the maximality of each $\mu \in \Lambda' \setminus \Lambda$ has been assured. We call the resulting modified algorithm the *greedy algorithm*.

The greedy algorithm generates an interesting trade-off between the quality of the reverse engineered query and complexity. On one hand, as we only add enough triple patterns to the query to justify the positive examples, the resulting query will be relatively small (at the very least, not larger that the query produced by the maximal algorithm). On the other hand, the process of checking the maximality of a positive example $\mu \in \Omega$ when adding each triple pattern t to $P_{\mathsf{can}}^{\text{greedy}}(\Lambda, D, \Omega)$ takes exponential time, as every possible extension mapping ν such that $\mu \sqsubset \nu$ must be checked (the number of such extensions is exponential in the size of Ω).

Finally, we briefly comment on the modifications required for the decision problem RevEng^{+}(SP[$\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}$]). In this case, the construction in Equation 3.1 can be modified to add a filter expression for each $\Lambda \in \mathcal{C}(\Omega)$. In essence, each $P_{\mathsf{can}}(\Lambda, D, \Omega)$ now consists of a set of triple patterns and a *set of filter comparisons*, which can be of the form $?\mathtt{X} = ?\mathtt{Y}$, $?\mathtt{X} \neq ?\mathtt{Y}$, $?\mathtt{X} = a$, and $?\mathtt{X} \neq a$, for some variables $?\mathtt{X}, ?\mathtt{Y} \in \mathsf{V}$, and some $a \in \mathsf{U} \cup \mathsf{L}$. In this scenario, the atype$(D, \Omega_{\mathsf{VarsOf}(\Lambda)})$ can be generalised to include all filter comparisons R for which $\mu \models R$ for each $\mu \in \Omega_{\mathsf{VarsOf}(\Lambda)}$.

ALGORITHM 2: Outline for deciding $\text{RevEng}^{+}_{\mathsf{tree}}$(SP[$\mathsf{AOwd}$]).

Input: RDF graph D, set of mappings Ω.
Output: A query $P \in$ SP[AOwd] such that $\Omega \subseteq [\![P]\!]_D$ if such a query exists and Ω is tree-like; *null* otherwise.

```
1 if $\Omega$ is not consistent and tree-like then return null;
2 $P \leftarrow$ BuildCanon($D, \Omega$);
3 if CheckCanon($P, \Omega$) = False then return null ;
4 if $\Omega \subseteq [\![P]\!]_D$ then return $P$ else return null ;
```

Chapter 4

SPARQLByE: Reverse Engineering Systems

4.1 Introduction

Having developed the theoretical basis for reverse engineering SPARQL queries, we now return to the initial goal of increasing the usability of RDF database systems. As was discussed previously, an enormous range of data of broad interest is available over Semantic Web-based interfaces, leading to the possibility of a huge increase in the number of active "data queriers". In this context, key obstacles for exploiting the availability of open data, and thus to widening its consumption, are the unfamiliarity of users with the structure of the data, as well as their unfamiliarity with SPARQL.

Although the RDF data model was introduced previously, recall that RDF views data as collections of RDF triples, making heavy use of web identifiers (URIs). This is a fairly low-level representation that is not easy for users to deal with by navigating or browsing the data one item at a time. Standard web Application Programming Interfaces (APIs) usually expose endpoints to the SPARQL query language, which allows users to pose queries that combine and filter information. However, making effective use of SPARQL still requires familiarity both with the precise URIs and with the syntax and semantics of SPARQL operators.

To address these issues, querying RDF data by example has the potential to more closely resemble the natural way in which a human user would approach a querying problem [109, 29, 110]. Querying by example is particularly attractive in an open data setting, since it eliminates the need to understand the structure of data as well as the features of SPARQL needed to express a query. Even users familiar with SPARQL and with a given dataset may prefer to explore the data via example and have the system suggest generalisations.

In this chapter we develop and present a query-by-example system for RDF and SPARQL, which we name SPARQLByE (SPARQL by Example). Its core is based on the set of reverse-engineering algorithms for SPARQL which were developed in the previous chapter. Through this reverse engineering implementation and user interface, we demonstrate how reverse engineering, along with other techniques (such as query relaxation) enables our system to guide users who are unfamiliar with both the dataset and with SPARQL to the desired query and result set. To illustrate a potential querying scenario, consider the following example:

Example 4.1. *Consider* DBpedia*, a public repository of RDF triples which represent knowledge extracted from Wikipedia, and consider a user who would like to extract a list of all Spanish-speaking countries. Although DBpedia has a free public SPARQL endpoint available (see:* `http://dbpedia.org/sparql`*), this interface does not provide much in the way of help for a user who is not familiar with the syntax and semantics of SPARQL queries. Moreover, formulating the appropriate SPARQL query would be challenging even for an experienced SPARQL user, as intimate knowledge of the DBpedia ontology and specific URIs is necessary to express the correct required triples.* □

In contrast, the query-by-example paradigm allows users to specify their information needs using positive and negative examples.

Example 4.2. *Continuing with the previous example, consider now that the user has access to a query-by-example system like SPARQLByE. In this case, the user might indicate Chile, Bolivia, Venezuela, and Spain as positive examples, while indicating Brazil and Angola as negative examples. The system would then be capable of guessing that the user is interested in the following query, and presenting both the query and its results to the user:*

```
SELECT * WHERE {
  ?s  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://dbpedia.org/ontology/Country> .
  ?s  <http://purl.org/dc/terms/subject>
      <http://dbpedia.org/resource/Category:
      Spanish-speaking_countries_and_territories>
}
```

Each URI in the query is of the form `<http://...>`*. In particular,* `<http://dbpedia.org/ontology/Country>` *is the URI for the concept* `Country`*, so in the first triple of the query we are asking to store in the variable* `?s` *all the countries in DBpedia. Moreover, the second triple asks for all the Spanish-speaking countries and territories. Finally, these two triples are combined by means of the period symbol in SPARQL, which essentially represents the* **AND** *operator. Thus, this combination is used to extract the list of all Spanish-speaking countries in DBpedia. Notice that the URIs and triples used in the query are not trivial to remember, even for experienced SPARQL users.* □

Our system, SPARQLByE, allows users to obtain information without any knowledge of SPARQL by supplementing reverse engineering of SPARQL with techniques for guiding the user to further positive and negative examples. In what follows, we first present the architecture and design decisions of the SPARQLByE system, which mainly consists of a Java backend that performs the reverse engineering tasks, based on a set of labelled examples as input, the end objective being to converge towards a reverse engineered query which satisfies the user's expectation. This backend is complemented by a web interface front end, where the user may interact with said set of labelled examples, directing the system towards the desired SPARQL query. Next we take a closer look at the function of its different modules, especially those that work to provide that user with further interaction paths (i.e. providing further positive or negative examples). Finally, we discuss the limitations of the system.

4.1.1 Related work

Query by example systems date back to the early days of relational databases [121, 122], with the initial focus being on a specification using only positive examples. Although [121, 122] took the first steps towards redefining the classical querying paradigm, the system presented in that paper would perhaps be more accurately described as an alternative example-based query language than a reverse engineering system. Nevertheless, the focus on the user interface and user experience for querying showed that, even early on in the history of database systems, there was a pain point to be addressed. Reverse engineering query systems from positive examples has been studied for a number of query languages [119, 110]. The use of positive and negative examples within query learning has been explored in the context of XML queries [36].

Query by example systems for relational data include the JIM system of [29]. Like SPARQLByE, JIM does not simply reverse engineer a query, but allows interaction with a user. However, the underlying reverse-engineering algorithms for SPARQL and relational data are quite different, due to the presence in SPARQL of the **OPTIONAL** operator for extracting optional information. The additional features in SPARQLByE for query-by-example, beyond reverse engineering, are tailored to issues specific to the open data setting, such as the difficulty of mapping entities to URIs. Furthermore, SPARQLByE is designed to connect to a predefined (and customisable) SPARQL endpoint, thus achieving a modular design, allowing it to be added onto existing Semantic Web infrastructure.

Recent systems such as Sapphire [42] also aim to improve the querying experience by striking a balance between keyword search and assisted SPARQL query generation. While Sapphire does not use reverse engineering algorithms under the hood, the interface is similar. Query autocompletion systems also have received attention [117]. This chapter is based on work published in [38], although further work on providing potential positive and negative examples has not been previously published.

4.2 SPARQLByE System Overview

SPARQLByE can be attached to any RDF dataset D. The main input is a set of *annotated mappings* which are specified by the user. An annotated mapping, in this context, is a mapping (see previous chapter for the precise definition) labelled as either a positive example or a negative example. A SPARQL query Q, when evaluated on a RDF dataset D, returns a set of mappings $Q(D)$—the *result set* of Q on D. A result set implicitly defines an annotated mapping set where the mappings in $Q(D)$ are positive examples and the mappings outside of $Q(D)$ are the negative examples. Recall that a set of annotated mappings Ω is *consistent* with a query Q on D if the positive mappings in Ω are in $Q(D)$ and the negative mappings in Ω are all outside of $Q(D)$. The system proceeds by discovering a query Q which is consistent with the annotated examples presented by the user, and to present the full result set $Q(D)$ (the query itself need not be shown). The user may then *refine* the set of annotated mappings in order to improve the result set; this refinement step will be explained in detail below.

Example 4.3. *Consider our running example where a user wants to extract the list of all Spanish-speaking countries in DBpedia. In this case, the user has annotated the following mappings as positive examples:*

```
?s  ↦  Chile            ?s  ↦  Bolivia
?s  ↦  Venezuela        ?s  ↦  Spain
```

Each one of these mappings gives a possible value for the variable `?s`*, for example* `Chile` *and* `Bolivia`*. On the other side, the user has annotated the following mappings as negative examples:*

$$\texttt{?s} \mapsto \texttt{Brazil} \qquad\qquad \texttt{?s} \mapsto \texttt{Angola}$$

These negative examples indicate that variable `?s` *cannot take the value* `Brazil` *or the value* `Angola`*.*

Let Ω be a set consisting of the previous positive and negative examples. Moreover, let Q be the SPARQL query given in Example 4.2, and assume that D is the DBpedia RDF dataset. In this case, we have that Ω is consistent with the query Q on D, as `Chile`*,* `Bolivia`*,* `Venezuela` *and* `Spain` *are answers to the query Q over D, while* `Brazil` *and* `Angola` *are not. On the other hand, consider the following SPARQL query Q' asking for the list of countries in DBpedia:*

```
SELECT * WHERE {
  ?s  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
      <http://dbpedia.org/ontology/Country>
}
```

In this case, we have that Ω is not consistent with Q' on D as `Brazil` *is an answer to Q' over D, as well as* `Angola`*.* □

At the core of SPARQLByE is a set of *reverse-engineering algorithms*, based on Chapter 3, which take a set of annotated mappings and return a SPARQL query that is consistent with the annotated examples [11]. SPARQLByE focuses on SPARQL queries which make use of only the **AND** and **OPTIONAL** operators. Given a set of mappings, a basic step in the system is to reverse engineer a query, evaluate it, and return the result set (or a pointer into a page of it, in case it is too large) to the user. This is the *reverse-engineering step*.

Reverse-engineering steps are interleaved with *example refinement steps*, in which a user responds to the displayed result set by refining the set of annotated mappings. SPARQLByE provides several forms of assistance to a user in the refinement step, including support for translating text descriptions to URIs (needed in constructing positive examples) and for creating pools of *candidate examples*, which the user may add to the labelled set of mappings.

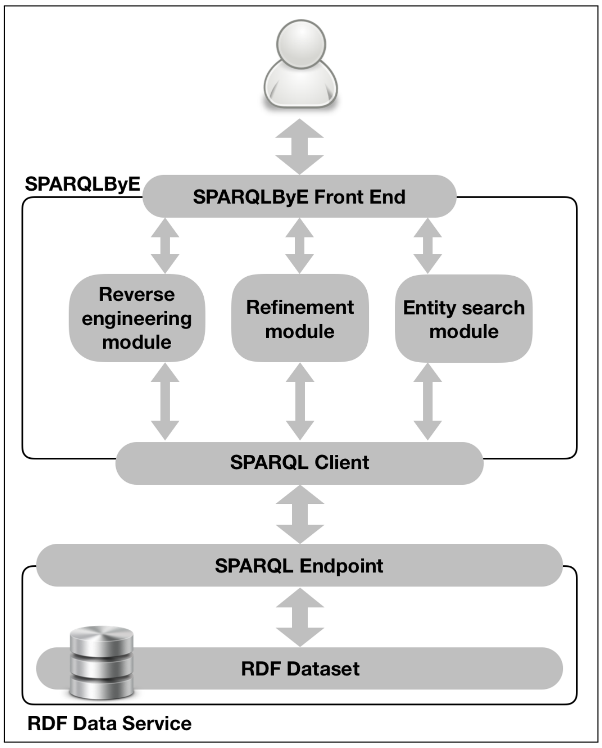


Figure 4.1: SPARQLByE architecture.

Figure 4.1 shows the architecture of SPARQLByE. An HTML-JavaScript front-end is backed by the Java server which is responsible for running the reverse engineering algorithms in the *reverse-engineering* module and obtaining candidate mappings in the *refinement module.* An auxiliary *entity search module* translates keyword searches to URIs.

The two main modules depend on several lower-level ones. In particular, the reverse-engineering algorithms require the execution of auxiliary SPARQL queries, which are delegated to a SPARQL endpoint; this may be either a local or a public endpoint, allowing for flexible setup.

4.3 Reverse Engineering module

Our reverse-engineering approach is based on Chapter 3. We look for well-designed SPARQL queries with **AND** and **OPTIONAL** that are consistent with a set of positive and negative examples, and which satisfy the following natural restrictions: they are *well-designed*, [84], and they they are *tree-like*, i.e. the domains of different mappings (that is, sets of variables) should form a tree-like order, as described in the previous chapter. Roughly speaking, the algorithm proceeds by induction on domains starting with the the smallest ones. At each inductive step it looks to put together a set of *safe patterns* through the **AND** operator, that is, patterns which are satisfied by the positive examples, are not fit by the negative examples, and which are consistent with the absent variables (that is, NULL values) present in positive examples. We make use of a "greedy" version of the algorithm that iteratively adds patterns until it obtains a safe set. This greedy approach has the desirable property of generating relatively small reverse-engineered queries. The algorithm needs to query the data to generate potential patterns, and also to check that a candidate query does correctly fit the data. This is done via issuing SPARQL queries to the endpoint.

As shown in Chapter 3, the core algorithm for reverse engineering well-designed, treelike SPARQL queries must build the *atomic type* for each node of the tree. In what follows, to simplify the description, we assume **AND**-queries, where the tree consists of only one node. The atomic type may be built naïvely with the iteration shown in Algorithm 3.

ALGORITHM 3: Naïve method for generating a reverse-engineered candidate.

```
1 for $t \in (\mathsf{U} \cup \mathsf{VarsOf}(\Omega))^3$ do
2     if $t \in$ atype then
3         Add $t$ to $Q$;
4     end
5 end
```

The main issue with Algorithm 3 is that the generated query will tend to be very large and take very long to compute, as usually many triple patterns satisfy the requirement. Thus, it is necessary to include a greedy aspect to the algorithm.

Another approach, which more closely reflects the inner workings of SPARQLByE, is to loop only until enough triples patterns have been included to justify the positive and negative examples. Algorithm 4 shows the structure of the steps taken.

A final and crucial consideration must be made regarding the step of iterating over the set $(\mathsf{U} \cup \mathsf{VarsOf}(\Omega))^3$, in order to produce candidate triple patterns to be included in the reverse engineered query. This set, of course, is not only very large but must be accessed through the SPARQL endpoint (as U is the set of constants appearing in D). One possibility is to execute the following query to the endpoint:

```
SELECT DISTINCT ?u WHERE {
  { ?u ?c1 ?c2 . }
  UNION
  { ?c3 ?u ?c4 . }
```

ALGORITHM 4: Less naïve method for generating a reverse-engineered candidate.

```
Prepare $\Omega^+$ with positive mappings;
Prepare $\Omega^-$ with negative mappings;
while $Q$ does not mention all variables in $\Omega^+$ do
    Extract next $t$ from $(\mathsf{U} \cup \mathsf{VarsOf}(\Omega))^3$;
    if $t \in$ atype then
        Add $t$ to $Q$;
    end
end
foreach $\mu \in \Omega^-$ do
    if $\mu$ is not a partial answer of $Q$ over $D$ then
        Remove $\mu$ from $\Omega^-$;
    end
end
while $\Omega^-$ is not empty do
    Extract next $t$ from $(\mathsf{U} \cup \mathsf{VarsOf}(\Omega))^3$;
    if $t \in$ atype then
        Add $t$ to $Q$;
    end
    foreach $\mu \in \Omega^-$ do
        if $\mu$ is not a partial answer of $Q$ over $D$ then
            Remove $\mu$ from $\Omega^-$;
        end
    end
end
```

```
    UNION
    { ?c5 ?c6 ?u . }
  }
```

This would be extremely inefficient, however. In order to actually produce a stream of potential triple patterns, the problem is broken down into parts, as follows.

Firstly, assume that $\mathsf{VarsOf}(\Omega) = \{\texttt{?x\_1}, \ldots, \texttt{?x\_m}\}$. Then, consider the set $A = \mathsf{VarsOf}(\Omega) \cup \{C\}$, where C is a symbol which represents constants. We now categorise triple patterns in $(\mathsf{U} \cup \mathsf{V})^3$ into families corresponding to each of the elements in the set A^3. For example, the element $(\texttt{?x\_3}, C, \texttt{?x\_2})\}$ defines the family of all triple patterns of the form $t = (\texttt{?x\_3}, c, \texttt{?x\_2})\}$ for some $c \in \mathsf{U}$.

For what follows, consider $\mu_1, \ldots, \mu_n$ to be the positive examples. For a specific family $F \in A^3$ of triple patterns, we issue an *auxiliary* SPARQL query Q_F to the endpoint, built as follows:

1. Replace each C in the family F by a fresh variable `?c_i`. For example, if the

family is $F = (\texttt{?x}, C, C)$, then we obtain $t_F = (\texttt{?x}, \texttt{?c\_1}, \texttt{?c\_2})$.

2. Build the following set of triple patterns:

$$\{\mu_i(t_F) \mid i \in [1, n]\}\,.$$

which we subsequently interpret as the final auxiliary query Q_F.

The query Q_F returns all triple patterns in the family corresponding to F which also happen to be in the corresponding atomic type $\text{atype}(D, \Omega)$. We show the auxiliary SPARQL query corresponding to one such family as an example.

Example 4.4. *Consider a scenario where the two positive mappings are* $\{\texttt{?x} \mapsto \texttt{Chile}\}$ *and* $\{\texttt{?x} \mapsto \texttt{Bolivia}\}$*. For family* $F = (\texttt{?x}, C, C)$*, we obtain* $t_F = (\texttt{?x}, \texttt{?c\_1}, \texttt{?c\_2})$*, and the following auxiliary SPARQL query* Q_F*:*

```
SELECT ?c WHERE {
  Chile ?c1 ?c2 .
  Bolivia ?c1 ?c2 .
}
```

This auxiliary query has been constructed in such a way that its result mappings will contain every possible combination of constants $c_1, c_2 \in \mathsf{U}$ *such that the triple pattern* $t = (\texttt{?x}, c_1, c_2)$ *is also in the atomic type.* □

Iterating over the answers of the auxiliary query Q_F (and the corresponding auxiliary query for each family) is a much more efficient way of traversing $(\mathsf{U} \cup \mathsf{VarsOf}(\Omega))^3$.

4.4 Example Refinement and other modules

SPARQLByE provides several features to assist users in refining annotated examples.

The top-right-hand side of the user interface houses the Results Panel. Here, the generated query is evaluated and a selection of results is presented (see Section 4.6). A user may use this panel to judge whether the answers to the generated query are satisfactory; in particular, an answer from this panel may be marked, by the user, as a negative example, thus *refining* the set of labelled examples and forcing SPARQLByE to generate a new, more restrictive, query. In this way, the Results Panel also serves as a *pool for potential negative examples* (see Section 4.6 for a more in-depth discussion on how this is achieved).

Below the Results Panel is the Extra Results Panel, whose main purpose is to serve as a *pool for potential positive examples.* This is achieved by generating a relaxed version of the current working query. A selection of the answers of the relaxed query is then presented in the Extra Results Panel. Much work has been done in the area of query relaxation [43]; in our system we implement a relatively simple approach (see discussion below).

The purpose of relaxing queries is to include answers which are not captured by the working query Q, and which may be of interest to the user. Consequently, labelling one of these as a positive example would force the system to generate a new — and possibly more relaxed — working query.

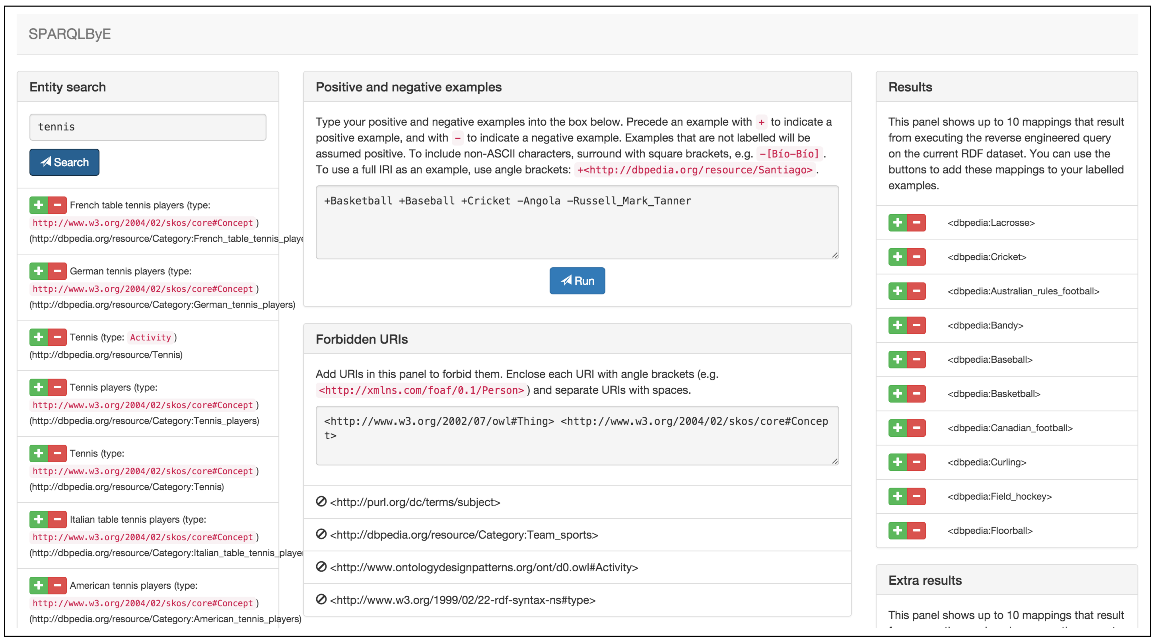
Finally, finding the correct entity identifiers (i.e. URIs) presents one of the main barriers to the formulation of meaningful SPARQL queries. Entity URIs are notoriously hard to remember; even the use of standard URI prefixes do not solve this problem, as several such prefixes exist and must be chosen correctly. The Entity Search Panel of SPARQLByE provides a simple keyword search for users to find appropriate URIs.

Example 4.5. *Figure 4.2 shows the result of inputting* `tennis` *into the entity search module (left-most panel of the interface). The result is a list of entities, where, for each, their* `label` *is presented, along with a* type. *The buttons allow for assigning the URI as a positive or negative example (note that this will only apply if the examples have arity 1).* □

Keyword searches in the entity search box are ultimately used to supply the user with a list of entities (URIs). This module allows for different techniques to be used, ranging from simple text search to using third-party semantic annotator APIs. Currently, the entity search module translates the keyword search to a SPARQL query which asks for a `uri` such that the triple (`uri`, `rdfs:label`, `X`) is in the dataset with label `X` containing each keyword as a substring.

4.5 SPARQL-based backend

SPARQLByE relies on a SPARQL endpoint for its operation. The reverse engineering of SPARQL queries from positive and negative examples is achieved by generating intermediate SPARQL queries which allow the reverse-engineering module to first *produce* a candidate query, and then to *check* said query. The fact that SPARQLByE can operate on a SPARQL endpoint allows for flexible setup including:


SPARQLByE

Entity search

```
tennis
```

French table tennis players (type: `http://www.w3.org/2004/02/skos/core#Concept`)
(http://dbpedia.org/resource/Category:French_table_tennis_playe

German tennis players (type: `http://www.w3.org/2004/02/skos/core#Concept`)
(http://dbpedia.org/resource/Category:German_tennis_players)

Tennis (type: `Activity`)
(http://dbpedia.org/resource/Tennis)

Tennis players (type: `http://www.w3.org/2004/02/skos/core#Concept`)
(http://dbpedia.org/resource/Category:Tennis_players)

Tennis (type: `http://www.w3.org/2004/02/skos/core#Concept`)
(http://dbpedia.org/resource/Category:Tennis)

Italian table tennis players (type: `http://www.w3.org/2004/02/skos/core#Concept`)
(http://dbpedia.org/resource/Category:Italian_table_tennis_playe

American tennis players (type: `http://www.w3.org/2004/02/skos/core#Concept`)
(http://dbpedia.org/resource/Category:American_tennis_players)

Positive and negative examples

Type your positive and negative examples into the box below. Precede an example with `+` to indicate a positive example, and with `–` to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. `–[Bío-Bío]`. To use a full IRI as an example, use angle brackets: `+<http://dbpedia.org/resource/Santiago>`.

```
+Basketball +Baseball +Cricket –Angola –Russell_Mark_Tanner
```

Forbidden URIs

Add URIs in this panel to forbid them. Enclose each URI with angle brackets (e.g. `<http://xmlns.com/foaf/0.1/Person>`) and separate URIs with spaces.

```
<http://www.w3.org/2002/07/owl#Thing> <http://www.w3.org/2004/02/skos/core#Concep
t>
```

⊘ <http://purl.org/dc/terms/subject>

⊘ <http://dbpedia.org/resource/Category:Team_sports>

⊘ <http://www.ontologydesignpatterns.org/ont/d0.owl#Activity>

⊘ <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>

Results

This panel shows up to 10 mappings that result from executing the reverse engineered query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

<dbpedia:Lacrosse>

<dbpedia:Cricket>

<dbpedia:Australian_rules_football>

<dbpedia:Bandy>

<dbpedia:Baseball>

<dbpedia:Basketball>

<dbpedia:Canadian_football>

<dbpedia:Curling>

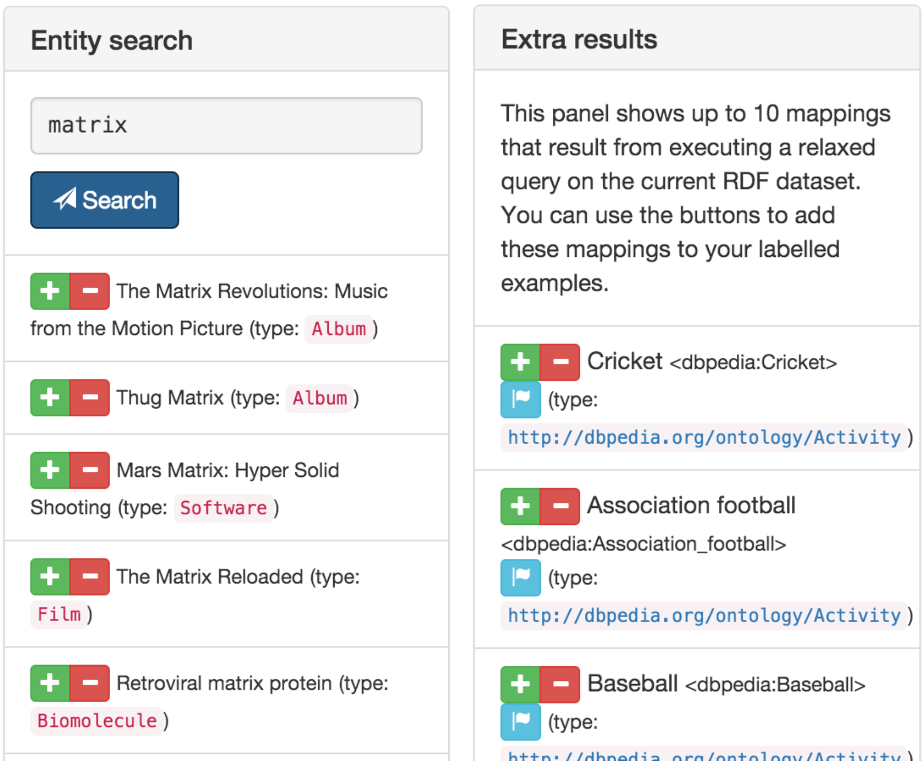
<dbpedia:Field_hockey>

<dbpedia:Floorball>

Extra results

This panel shows up to 10 mappings that result

Figure 4.2: SPARQLByE user interface overview showing the result of adding labelled examples and using the Entity search box.


Entity search

```
matrix
```

The Matrix Revolutions: Music from the Motion Picture (type: `Album`)

Thug Matrix (type: `Album`)

Mars Matrix: Hyper Solid Shooting (type: `Software`)

The Matrix Reloaded (type: `Film`)

Retroviral matrix protein (type: `Biomolecule`)

Extra results

This panel shows up to 10 mappings that result from executing a relaxed query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

Cricket <dbpedia:Cricket> (type: `http://dbpedia.org/ontology/Activity`)

Association football <dbpedia:Association_football> (type: `http://dbpedia.org/ontology/Activity`)

Baseball <dbpedia:Baseball> (type:

Figure 4.3: Close-up of the SPARQLByE Entity Search Panel (left) and Extra Results Panel (right).

- using a local RDF backend for quicker access and custom indices tailored for the query loads that the reverse-engineering module will generate,
- plugging in the SPARQLByE system into an existing SPARQL endpoint, and
- installing SPARQLByE as a third-party client to a public SPARQL endpoint.

4.6 Supplying negative examples

As discussed previously, a central task of the system is to provide potential further examples — in this case negative examples — to the user, in order to restrict the reverse engineered query further. Supplying such a pool of potential negative examples allows the system to provide the user with further interaction alternatives. In SPARQLByE, this is achieved by executing the current reverse engineered query and returning a ranked sample of its results to the user.

As SPARQLByE is based on a SPARQL backend (see previous section), obtaining a ranked sample of results is achieved by performing a transformation of the reverse engineered query, a.k.a. working query, Q and returning the results of the transformed query Q'. The objective is to produce a query Q' which will return a list of results that are sorted in a way that will maximize the probability of the user finding a diverse set of potential negative examples to add to their current example set. Note that simply executing the reverse engineered query Q and presenting the first answers may result in a less useful answer set for the user, as the ordering of the results is undefined, and thus the user may be presented with, for example, an alphabetically ordered list of answers.

Example 4.6 shows a simple implementation of the query transformation that prioritizes answers based on the amount of classes they belong to in the dataset.

Example 4.6. *Consider a scenario where the current reverse engineered query Q is the following:*

```
SELECT ?x WHERE {
?x type Country .
}
```

Under a simple class membership based implementation of the Results Panel, the transformation will results in the following query Q':

```
SELECT ?x (COUNT(?t) as ?count) WHERE {
?x type Country .
?x type ?t
}
GROUP BY ?x
ORDER BY DESC(?count)
```

As such, the transformed query returns the results of the reverse engineered query, prioritizing answers that belong to more classes. □

In Example 4.6, the ranking of query answers is expressed as a transformation on the reversed engineered query, based on the number of classes to which an answer belongs. Another possibility — and the one currently implemented in SPARQLByE — is to rank answers based on their out degree, as in Example 4.7.

Example 4.7. *Consider a scenario where the current reverse engineered query Q is the following:*

```
SELECT ?x WHERE {
?x type Country .
}
```

The transformed query will then be the following query Q':

```
SELECT ?x (COUNT(?t) as ?count) WHERE {
?x type Country .
?x ?p ?t
}
GROUP BY ?x
ORDER BY DESC(?count)
```

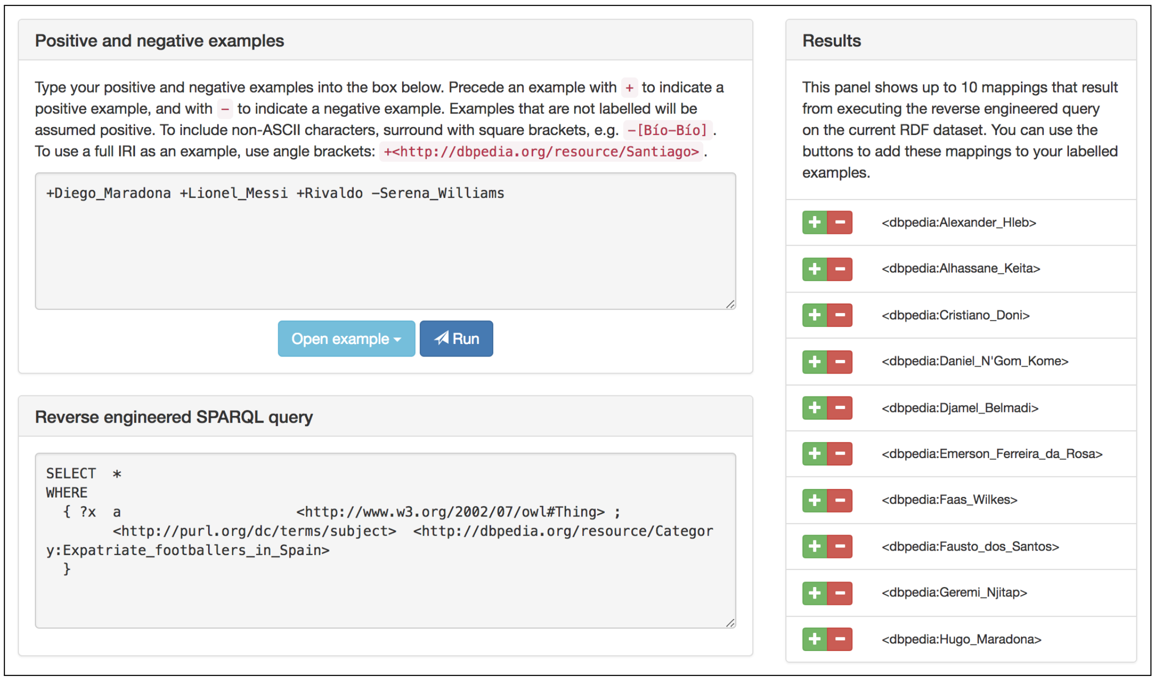
As such, the transformed query returns the results of the reverse engineered query, prioritizing answers that have more outgoing edges. □

We now discuss the advantages and disadvantages of these ranking based implementations. Firstly, the currently implemented ranking (ranking answers according to the number of outgoing edges they have in the dataset) is efficient to implement in the SPARQL-endpoint design architecture. The number of outgoing edges also provides a simple and transparent measure of the importance of and relative visibility of an entity in the dataset. This implementation also has the advantage of being similar in structure to more widely used rankings which are usually based on the popularity of an entity among users (or any other numerical ranking which may be calculated straightforwardly).

In order to compare the trivial and ranked implementations of the Results Panel, we provide an example interaction with the system in the series of Figures 4.4, 4.5, 4.6, 4.7, 4.8, and 4.9. These figures consider a hypothetical user who wishes to explore the set of association football players who are signed with FC Barcelona. We compare a trivial implementation of the Results Panel, that is, an implementation with the trivial transformation $Q' = Q$ (Figures 4.4, 4.5, 4.6, and 4.7), with the outgoing edges based ranking implementation of the Results Panel (Figures 4.8 and 4.9). Initially, the user inputs three positive examples: `Diego_Maradona`, `Lionel_Messi`, and `Rivaldo`, and one negative example: `Serena_Williams` (a tennis player).

Consider the trivial implementation of the Results Panel. Figure 4.4 shows the result of executing the initial input. The system has reverse engineered a query which seeks expatriate footballers in Spain, which is certainly a reasonable conclusion given the examples. The Results Panel consequently shows such a list; the main problem is that the order in which these results are presented is undefined, or more precisely, defined by the storage order of the entities in the intermediate data structures used to answer the working query. In this case, the results appear to be in URI-alphabetical order. Figures 4.5, 4.6, and 4.7 show the progression of accepting the first proposed negative example at each step; while the system eventually converges to a working query which includes the correct constraint, that the entity be a footballer of FC Barcelona, the final query also contains less useful constraints. Perhaps most importantly, the proposed negative examples were not necessarily recognisable football players.

Now consider an out degree ranking based implementation of the Results Panel; specifically, the transformation shown in Example 4.6. This scenario is depicted in the two figures in the series mentioned previously (Figures 4.8 and 4.9). After the initial input is executed, it is immediately clear that the Results Panel is showing more widely known football players (Figure 4.8). As the first entity shown in the Results Panel, David Beckham, does not play for FC Barcelona, adding it as a further negative example should serve to converge on the desired query. Figure 4.9 shows that this negative example has produced the desired query, and the convergence was much faster in this case.



Positive and negative examples

Type your positive and negative examples into the box below. Precede an example with + to indicate a positive example, and with – to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. `–[Bío-Bío]`. To use a full IRI as an example, use angle brackets: `+<http://dbpedia.org/resource/Santiago>`.

```
+Diego_Maradona +Lionel_Messi +Rivaldo -Serena_Williams
```

Reverse engineered SPARQL query

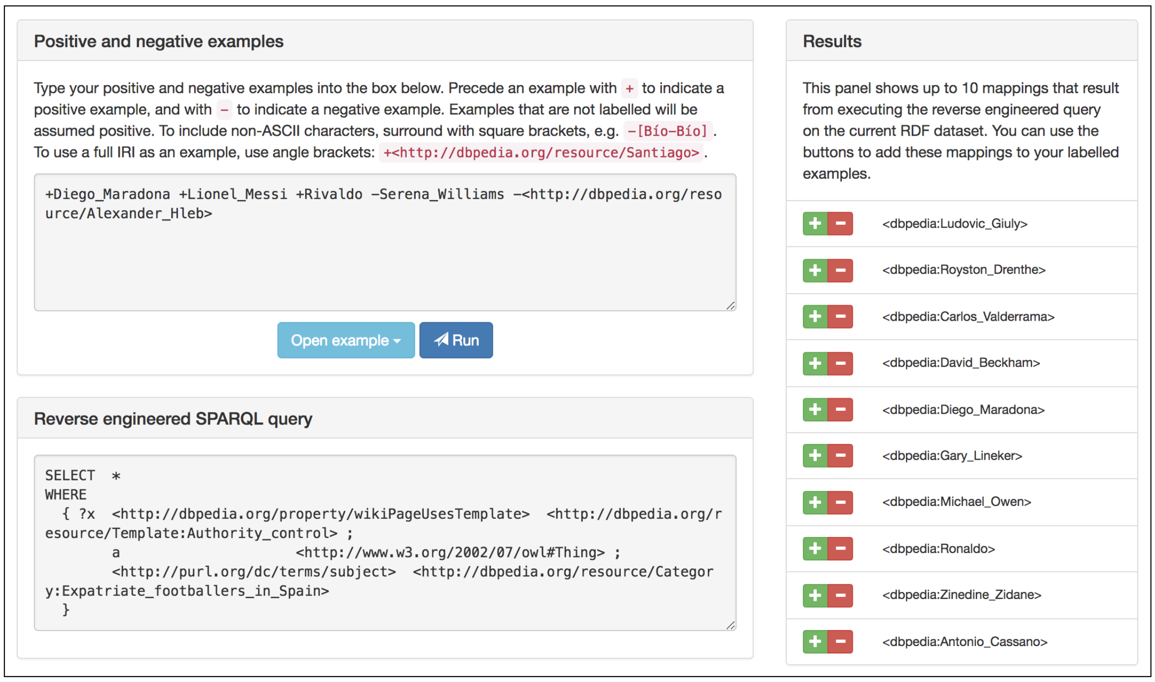
```
SELECT  *
WHERE
  { ?x  a                     <http://www.w3.org/2002/07/owl#Thing> ;
        <http://purl.org/dc/terms/subject>  <http://dbpedia.org/resource/Categor
y:Expatriate_footballers_in_Spain>
  }
```

Results

This panel shows up to 10 mappings that result from executing the reverse engineered query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

<dbpedia:Alexander_Hleb>
<dbpedia:Alhassane_Keita>
<dbpedia:Cristiano_Doni>
<dbpedia:Daniel_N'Gom_Kome>
<dbpedia:Djamel_Belmadi>
<dbpedia:Emerson_Ferreira_da_Rosa>
<dbpedia:Faas_Wilkes>
<dbpedia:Fausto_dos_Santos>
<dbpedia:Geremi_Njitap>
<dbpedia:Hugo_Maradona>

Figure 4.4: Screenshot of SPARQLByE showing the result of a user inputting three known football players as initial positive examples (and a known tennis player as a negative). The Results Panel provides answers to the working query with a trivial implementation (see Section 4.6 for a discussion, and see Figure 4.5 for the next figure in the series).


Positive and negative examples

Type your positive and negative examples into the box below. Precede an example with `+` to indicate a positive example, and with `−` to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. `−[Bío-Bío]`. To use a full IRI as an example, use angle brackets: `+<http://dbpedia.org/resource/Santiago>`.

```
+Diego_Maradona +Lionel_Messi +Rivaldo −Serena_Williams −<http://dbpedia.org/reso
urce/Alexander_Hleb>
```

Reverse engineered SPARQL query

```
SELECT  *
WHERE
  { ?x  <http://dbpedia.org/property/wikiPageUsesTemplate>  <http://dbpedia.org/r
esource/Template:Authority_control> ;
        a                     <http://www.w3.org/2002/07/owl#Thing> ;
        <http://purl.org/dc/terms/subject>  <http://dbpedia.org/resource/Categor
y:Expatriate_footballers_in_Spain>
  }
```

Results

This panel shows up to 10 mappings that result from executing the reverse engineered query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

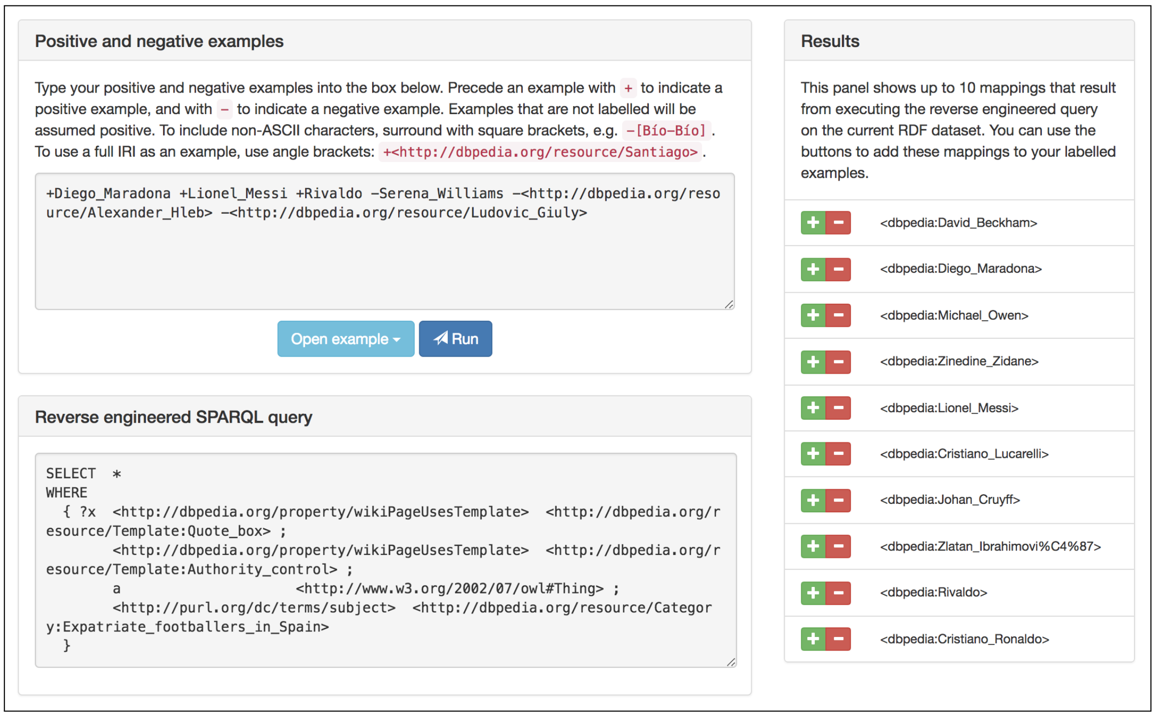
<dbpedia:Ludovic_Giuly>
<dbpedia:Royston_Drenthe>
<dbpedia:Carlos_Valderrama>
<dbpedia:David_Beckham>
<dbpedia:Diego_Maradona>
<dbpedia:Gary_Lineker>
<dbpedia:Michael_Owen>
<dbpedia:Ronaldo>
<dbpedia:Zinedine_Zidane>
<dbpedia:Antonio_Cassano>

Figure 4.5: Screenshot of SPARQLByE showing the progression of a user interacting with a trivial implementation of the Results Panel (see Section 4.6 for a discussion, see Figure 4.4 for the previous figure in the series, and see Figure 4.6 for the next figure in the series).

Although the user interaction shown is anecdotal evidence, it serves to illustrate the fact that a better implementation of the Results Panel may go a long way in assisting the user to navigate and explore the data with SPARQLByE.

The main disadvantage of ranking-based implementation is that it does not easily generalise to n-ary answers, as a answer tuple does not have a clearly defined membership to a class. In the current implementation of the system, for n-ary answers we simply focus on the first variable for the purposes of ranking answers; clearly not a completely satisfactory solution. Another potential disadvantage is in the diversity of the answers that are returned, as two or more highly ranked answers may share the same set of classes, in which case they become redundant as potential negative examples. These problems generate the need to consider alternative implementations of the ranking.

In considering alternative ranking implementations, it may be desirable to provide a diverse set of answers, instead of prioritizing a numerical rank, for example. This may be achieved by obtaining a sample of answers that belong to different classes, or a set of examples that constitutes an anti-chain in the class hierarchy. This hypothetical implementation would ideally increase the probability of providing the user with a


Positive and negative examples

Type your positive and negative examples into the box below. Precede an example with + to indicate a positive example, and with − to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. `−[Bío−Bío]`. To use a full IRI as an example, use angle brackets: `+<http://dbpedia.org/resource/Santiago>`.

```
+Diego_Maradona +Lionel_Messi +Rivaldo −Serena_Williams −<http://dbpedia.org/reso
urce/Alexander_Hleb> −<http://dbpedia.org/resource/Ludovic_Giuly>
```

Reverse engineered SPARQL query

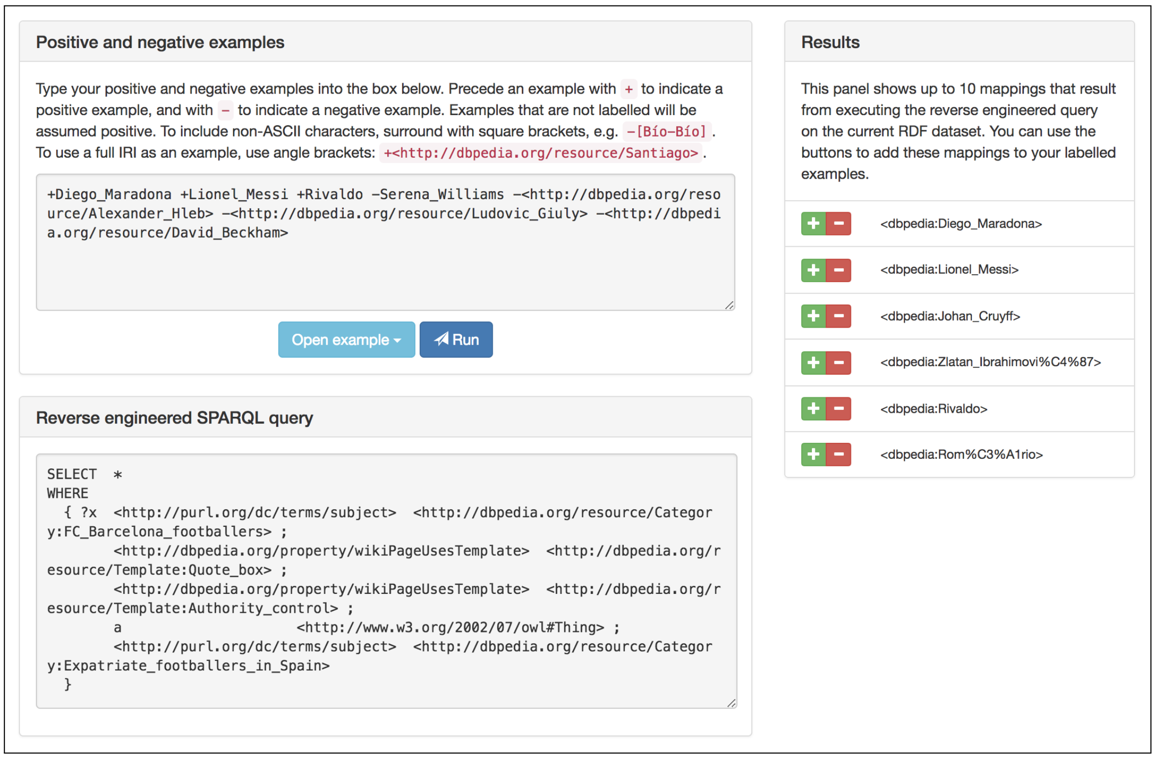
```
SELECT  *
WHERE
  { ?x  <http://dbpedia.org/property/wikiPageUsesTemplate>  <http://dbpedia.org/r
esource/Template:Quote_box> ;
        <http://dbpedia.org/property/wikiPageUsesTemplate>  <http://dbpedia.org/r
esource/Template:Authority_control> ;
        a                     <http://www.w3.org/2002/07/owl#Thing> ;
        <http://purl.org/dc/terms/subject>  <http://dbpedia.org/resource/Categor
y:Expatriate_footballers_in_Spain>
  }
```

Results

This panel shows up to 10 mappings that result from executing the reverse engineered query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

<dbpedia:David_Beckham>
<dbpedia:Diego_Maradona>
<dbpedia:Michael_Owen>
<dbpedia:Zinedine_Zidane>
<dbpedia:Lionel_Messi>
<dbpedia:Cristiano_Lucarelli>
<dbpedia:Johan_Cruyff>
<dbpedia:Zlatan_Ibrahimovi%C4%87>
<dbpedia:Rivaldo>
<dbpedia:Cristiano_Ronaldo>

Figure 4.6: Screenshot of SPARQLByE showing the progression of a user interacting with a trivial implementation of the Results Panel (see Section 4.6 for a discussion, see Figure 4.5 for the previous figure in the series, and see Figure 4.7 for the next figure in the series).



Type your positive and negative examples into the box below. Precede an example with + to indicate a positive example, and with − to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. −[Bío-Bío] . To use a full IRI as an example, use angle brackets: +<http://dbpedia.org/resource/Santiago> .

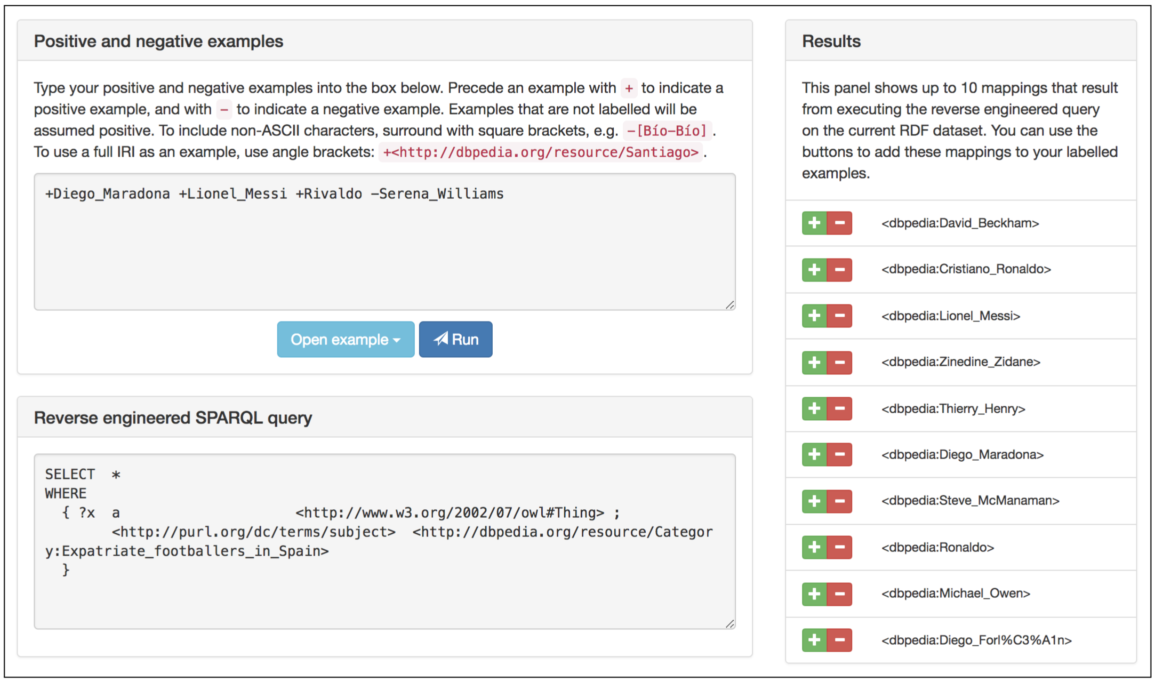
```
+Diego_Maradona +Lionel_Messi +Rivaldo -Serena_Williams -<http://dbpedia.org/reso
urce/Alexander_Hleb> -<http://dbpedia.org/resource/Ludovic_Giuly> -<http://dbpedi
a.org/resource/David_Beckham>
```

```
SELECT  *
WHERE
  { ?x  <http://purl.org/dc/terms/subject>  <http://dbpedia.org/resource/Categor
y:FC_Barcelona_footballers> ;
        <http://dbpedia.org/property/wikiPageUsesTemplate>  <http://dbpedia.org/r
esource/Template:Quote_box> ;
        <http://dbpedia.org/property/wikiPageUsesTemplate>  <http://dbpedia.org/r
esource/Template:Authority_control> ;
        a                     <http://www.w3.org/2002/07/owl#Thing> ;
        <http://purl.org/dc/terms/subject>  <http://dbpedia.org/resource/Categor
y:Expatriate_footballers_in_Spain>
  }
```

This panel shows up to 10 mappings that result from executing the reverse engineered query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

<dbpedia:Diego_Maradona>
<dbpedia:Lionel_Messi>
<dbpedia:Johan_Cruyff>
<dbpedia:Zlatan_Ibrahimovi%C4%87>
<dbpedia:Rivaldo>
<dbpedia:Rom%C3%A1rio>

Figure 4.7: Screenshot of SPARQLByE showing the progression of a user interacting with a trivial implementation of the Results Panel (see Section 4.6 for a discussion, see Figure 4.6 for the previous figure in the series, and see Figure 4.8 for the next figure in the series).



Type your positive and negative examples into the box below. Precede an example with `+` to indicate a positive example, and with `–` to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. `–[Bío–Bío]`. To use a full IRI as an example, use angle brackets: `+<http://dbpedia.org/resource/Santiago>`.

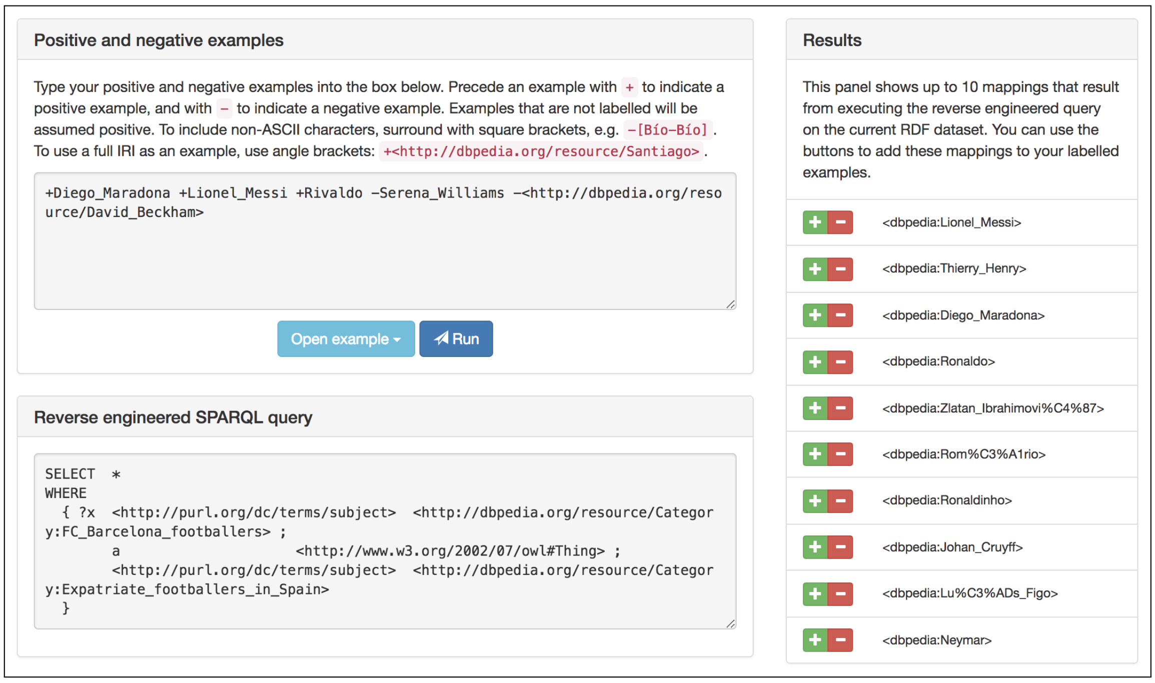
```
+Diego_Maradona +Lionel_Messi +Rivaldo –Serena_Williams
```

```
SELECT  *
WHERE
  { ?x  a                     <http://www.w3.org/2002/07/owl#Thing> ;
        <http://purl.org/dc/terms/subject>  <http://dbpedia.org/resource/Categor
y:Expatriate_footballers_in_Spain>
  }
```

This panel shows up to 10 mappings that result from executing the reverse engineered query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

```
<dbpedia:David_Beckham>
<dbpedia:Cristiano_Ronaldo>
<dbpedia:Lionel_Messi>
<dbpedia:Zinedine_Zidane>
<dbpedia:Thierry_Henry>
<dbpedia:Diego_Maradona>
<dbpedia:Steve_McManaman>
<dbpedia:Ronaldo>
<dbpedia:Michael_Owen>
<dbpedia:Diego_Forl%C3%A1n>
```

Figure 4.8: Screenshot of SPARQLByE showing the result of a user inputting three known football players as initial positive examples (and a known tennis player as a negative). The Results Panel provides answers to the working query with a ranked implementation (see Section 4.6 for a discussion, see Figure 4.7 for the previous figure in the series, and see Figure 4.9 for the next figure in the series).


Positive and negative examples

Type your positive and negative examples into the box below. Precede an example with `+` to indicate a positive example, and with `-` to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. `-[Bío-Bío]`. To use a full IRI as an example, use angle brackets: `+<http://dbpedia.org/resource/Santiago>`.

```
+Diego_Maradona +Lionel_Messi +Rivaldo -Serena_Williams -<http://dbpedia.org/reso
urce/David_Beckham>
```

Reverse engineered SPARQL query

```
SELECT  *
WHERE
  { ?x  <http://purl.org/dc/terms/subject>  <http://dbpedia.org/resource/Categor
y:FC_Barcelona_footballers> ;
        a                     <http://www.w3.org/2002/07/owl#Thing> ;
        <http://purl.org/dc/terms/subject>  <http://dbpedia.org/resource/Categor
y:Expatriate_footballers_in_Spain>
  }
```

Results

This panel shows up to 10 mappings that result from executing the reverse engineered query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

<dbpedia:Lionel_Messi>
<dbpedia:Thierry_Henry>
<dbpedia:Diego_Maradona>
<dbpedia:Ronaldo>
<dbpedia:Zlatan_Ibrahimovi%C4%87>
<dbpedia:Rom%C3%A1rio>
<dbpedia:Ronaldinho>
<dbpedia:Johan_Cruyff>
<dbpedia:Lu%C3%ADs_Figo>
<dbpedia:Neymar>

Figure 4.9: Screenshot of SPARQLByE showing the result of a user inputting three known football players as initial positive examples (and a known tennis player as a negative). The Results Panel provides answers to the working query with a ranked implementation (see Section 4.6 for a discussion, see Figure 4.8 for the previous figure in the series).

distinct sample of answers, but may come at a runtime cost.

Finally, this discussion assumes that it is possible to express this answer ranking as a transformation of the working query Q into a SPARQL query Q', in order to maintain the SPARQL-based system architecture. However, a user interface may also relax this restriction, providing alternative procedures for ranking the answers of the reverse engineered query.

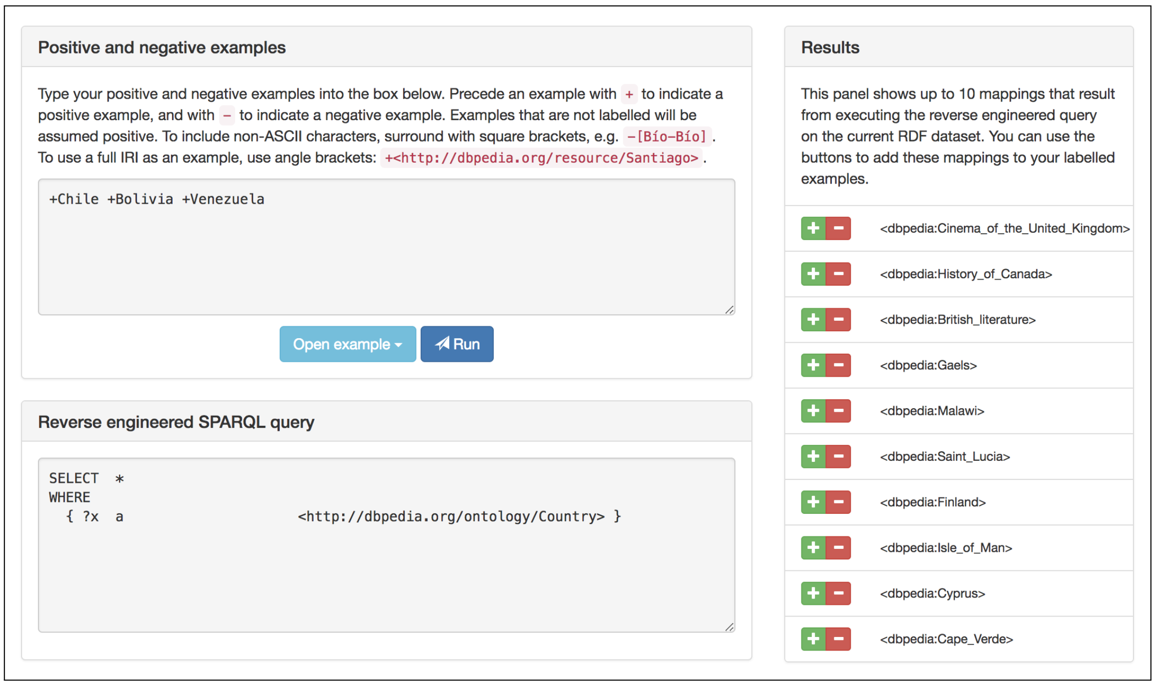
Although not exactly a disadvantage of the ranking-based implementation of the Results Panel, it is noteworthy to comment on the effect the quality of the data has on the results presented to the user. Consider Figure 4.10, where a hypothetical user seeks to explore Spanish-speaking countries. The user inputs three positive examples (`Chile`, `Bolivia`, and `Venezuela`), whereby the system reverse engineers the query shown. The resulting transformed query (which is not shown in the interface, but corresponds to the second query from Example 4.6) is executed and its results are presented to the user in the Results Panel. Interestingly, the first result is not actually a country (`Cinema_of_the_United_Kingdom`). This situation is caused by the fact that the data asserts that the entity `Cinema_of_the_United_Kingdom` is of type `Country`. Any data exploration tool (or query engine, for that matter) will incur in the same errors in cases like these, unless a previous data cleaning procedure is performed.

4.7 Supplying positive examples

The complementary task to that of providing potential negative examples to the user is to provide potential positive examples. For this, SPARQLByE provides a further set of answers in the Extra Results Panel. The intention of this panel is to address the possibility that the working query is too restrictive, and to provide the user an avenue to refine their example set to indicate this issue.

Note that the answers to the working query — that is, the answers presented in the Results Panel, which were discussed in the previous section — are not useful as potential further positive examples, as adding such an answer to the list of positive examples would be redundant (i.e. would not change the fact that the working query satisfies all labelled examples).

As with the Results Panel, the Extra Results Panel is modelled as a transformation of the working query, considering the design of SPARQLByE, which is based on a connection to a SPARQL endpoint. In this case, the transformation does not merely seek to sort the working query's results but to *relax* the working query, in order


Positive and negative examples

Type your positive and negative examples into the box below. Precede an example with + to indicate a positive example, and with − to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. −[Bío-Bío] . To use a full IRI as an example, use angle brackets: +<http://dbpedia.org/resource/Santiago> .

```
+Chile +Bolivia +Venezuela
```

Reverse engineered SPARQL query

```
SELECT  *
WHERE
  { ?x  a                     <http://dbpedia.org/ontology/Country> }
```

Results

This panel shows up to 10 mappings that result from executing the reverse engineered query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

<dbpedia:Cinema_of_the_United_Kingdom>
<dbpedia:History_of_Canada>
<dbpedia:British_literature>
<dbpedia:Gaels>
<dbpedia:Malawi>
<dbpedia:Saint_Lucia>
<dbpedia:Finland>
<dbpedia:Isle_of_Man>
<dbpedia:Cyprus>
<dbpedia:Cape_Verde>

Figure 4.10: Result of a user interaction in SPARQLByE in which the Results Panel uses a class-based criterium to rank the results. Note that the top answer in the Results Panel is not a country. This is due to the quality of the data.

to return a broader set of answers from which the user may select further positive examples. As before, there are several ways to implement such a query relaxation. All of the implementations we consider shall follow the same structure: we relax the working query Q by generating a modified query Q' which returns a superset of the answers to Q.

A basic implementation consists in randomly eliminating one of the triple patterns of the working query. As each triple pattern can be understood as a constraint on the answers, eliminating one produces a relaxed query. In the current implementation of SPARQLByE, however, we consider an ontology based query relaxation method. This consists of randomly selecting a triple of the form $(?\mathtt{x}, \mathtt{type}, \mathtt{C})$ from the working query and relaxing the class C to some randomly chosen superclass B of C.

In order to illustrate the Extra Results Panel, Figure 4.11 shows the progression of a user interaction in which the user seeks to explore the plant kingdom. Initially, the user inputs three positive examples: Spruce, Pine, and Fir, and one negative example: Lion. Upon executing, the system reverse engineers a query which returns conifers, which are actually a subset of the set of plants (the outcome is shown in the upper panel of Figure 4.11). In other words, the example set given resulted in a working query which is too restrictive.

In this scenario, it is not useful for the user to be provided with further negative examples, as these would restrict the working query — and therefore the returned answers — even more; potential positive examples thus become useful. In this case, the Extra Results Panel assists the user in generalising the answer set by running a relaxed query on the database. The relaxed query is not shown, but in this case the ontology based relaxation replaces `Conifer` by its superclass `Plant` (the outcome is shown in the lower panel of Figure 4.11). When the user adds the first answer from the Extra Results Panel as a further positive example, the system correctly obtains a working query that returns all plants.

This ontology based relaxation technique is not free of problems, however, as it is very common for reverse engineered queries to make no mention of the database's ontology, but to restrict on the basis of connections between entities. For working queries which do not contain type restrictions (i.e. do not contain triples of the form $(?\mathtt{x}, \mathtt{type}, \mathtt{C})$), the current implementation falls back to a trivial relaxation approach, that is, randomly eliminating a triple from the query.

As a final comment, the modular design of the SPARQLByE system allows for taking advantage of a large body of work in the area of query relaxation in general [49, 108], and in principle any of a number of query relaxation techniques may be used to provide further positive examples to the user.

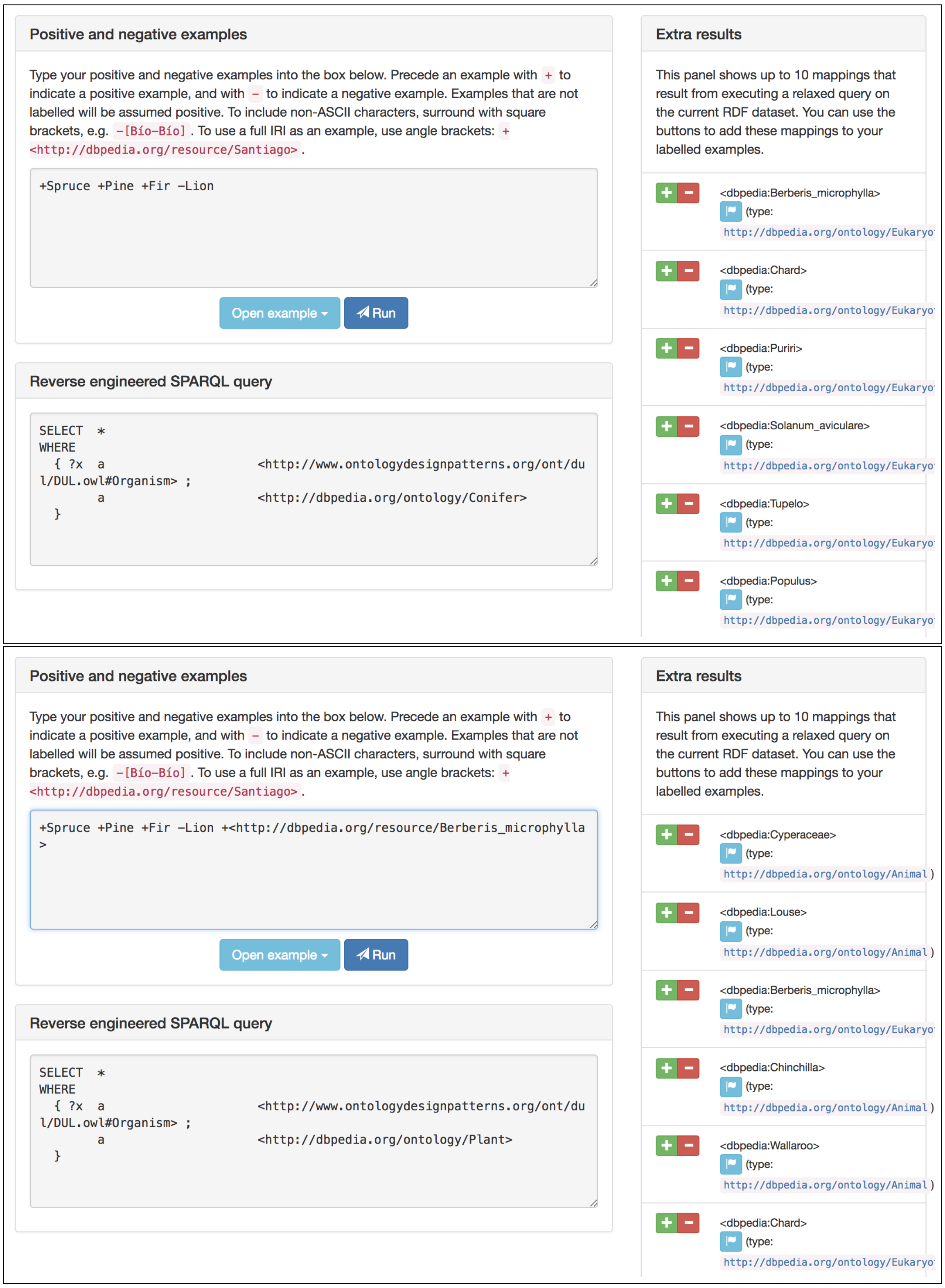
4.8 Use cases and examples

In contrast to a standard SPARQL query interface, where users are simply presented with a text field in which to formulate a query, SPARQLByE allows for a more interactive experience.

An obvious scenario which can showcase the capabilities of SPARQLByE is that of obtaining a query capable of capturing a class of entities.

In this case the mappings are of arity 1. As was shown in [11], a set of examples of arity 1 will always be reverse-engineerable with an **AND**-query (a query which only makes use of the **AND** operator).

Example 4.8. *Consider the case of a user seeking to obtain a list of Spanish-speaking countries. A possible first approach for a user is to input the countries* `Chile`*,* `Bolivia`*, and* `Venezuela` *as positive examples (see Figure 4.13, top frame). In this case, the corresponding reverse-engineered query asks for all entities of type* `Country`*, which is too general.*



Positive and negative examples

Type your positive and negative examples into the box below. Precede an example with + to indicate a positive example, and with − to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. −[Bío−Bío] . To use a full IRI as an example, use angle brackets: + <http://dbpedia.org/resource/Santiago> .

```
+Spruce +Pine +Fir −Lion
```

Reverse engineered SPARQL query

```
SELECT  *
WHERE
  { ?x  a                    <http://www.ontologydesignpatterns.org/ont/du
l/DUL.owl#Organism> ;
        a                    <http://dbpedia.org/ontology/Conifer>
  }
```

Extra results

This panel shows up to 10 mappings that result from executing a relaxed query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

<dbpedia:Berberis_microphylla> (type: http://dbpedia.org/ontology/Eukaryo

<dbpedia:Chard> (type: http://dbpedia.org/ontology/Eukaryo

<dbpedia:Puriri> (type: http://dbpedia.org/ontology/Eukaryo

<dbpedia:Solanum_aviculare> (type: http://dbpedia.org/ontology/Eukaryo

<dbpedia:Tupelo> (type: http://dbpedia.org/ontology/Eukaryo

<dbpedia:Populus> (type: http://dbpedia.org/ontology/Eukaryo

Positive and negative examples

Type your positive and negative examples into the box below. Precede an example with + to indicate a positive example, and with − to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. −[Bío−Bío] . To use a full IRI as an example, use angle brackets: + <http://dbpedia.org/resource/Santiago> .

```
+Spruce +Pine +Fir −Lion +<http://dbpedia.org/resource/Berberis_microphylla
>
```

Reverse engineered SPARQL query

```
SELECT  *
WHERE
  { ?x  a                    <http://www.ontologydesignpatterns.org/ont/du
l/DUL.owl#Organism> ;
        a                    <http://dbpedia.org/ontology/Plant>
  }
```

Extra results

This panel shows up to 10 mappings that result from executing a relaxed query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

<dbpedia:Cyperaceae> (type: http://dbpedia.org/ontology/Animal)

<dbpedia:Louse> (type: http://dbpedia.org/ontology/Animal)

<dbpedia:Berberis_microphylla> (type: http://dbpedia.org/ontology/Eukaryo

<dbpedia:Chinchilla> (type: http://dbpedia.org/ontology/Animal)

<dbpedia:Wallaroo> (type: http://dbpedia.org/ontology/Animal)

<dbpedia:Chard> (type: http://dbpedia.org/ontology/Eukaryo

Figure 4.11: Progression of a user interaction in SPARQLByE in which the Extra Results Panel uses an ontology-based relaxation implementation to provide potential further positive examples.

The right hand panel has the country `Angola`*, though, which can be added next as a negative example. The result is shown in the middle frame of Figure 4.13. In this case, the query is asking for countries in South America. The main problem with this query is its exclusion of the quintessential Spanish-speaking country:* `Spain`*, which may then be added as a positive example.*

The bottom frame of Figure 4.13 shows the final result set and the corresponding reverse engineered query that asks for entities of type `Country` *which also have the subject* `Spanish-speaking_countries_and_territories`*. Notice that although the spirit of the query is intuitive, it would not have been trivial for a user to arrive at the specific URIs and triples mentioned in the query without extensive knowledge of the DBpedia data and types.* □

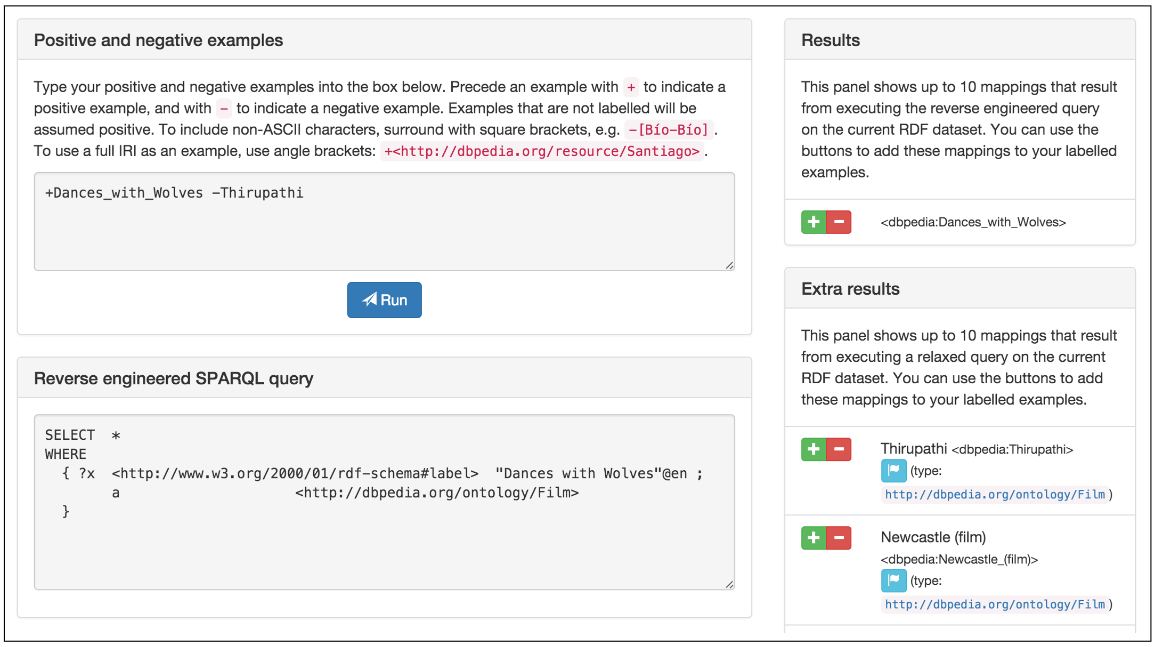
In the previous example, the final positive example (`Spain`) was crucial to obtaining the desired result set. However, it is not always clear what positive example to add. This problem is addressed by the Extra Results Panel, which suggests positives examples.

Example 4.9. *Consider a user who wishes to obtain a list of English-language movies. The user begins with the positive example* `+Dances_with_Wolves`*, and subsequently adds the negative example* `-Thirupathi`*. The resulting query, shown in Figure 4.12, asks for films which have the label* `Dances_with_Wolves`*, which only returns one result. Clearly this is overfitting the example data. The Extra Results Panel, however, has executed a relaxed query which asks for all films, and thus is able to suggest the film* `Newcastle_(film)`*. Adding Newcastle as a positive example gives the desired result set, corresponding to the query;*

```
SELECT * WHERE {
  ?x  <http://purl.org/dc/terms/subject>
      <http://dbpedia.org/resource/Category:English-language_films> .
  ?x  rdf:type  <http://dbpedia.org/ontology/Film>
}
```

□

Finally, a key feature of SPARQLByE is that it is able to reverse engineer SPARQL queries which make use of the **OPTIONAL** operator. In other words, labelled examples may contain unmapped (*null*) values. The following example illustrates this scenario.


Positive and negative examples

Type your positive and negative examples into the box below. Precede an example with `+` to indicate a positive example, and with `-` to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. `-[Bío-Bío]`. To use a full IRI as an example, use angle brackets: `+<http://dbpedia.org/resource/Santiago>`.

```
+Dances_with_Wolves -Thirupathi
```

Reverse engineered SPARQL query

```
SELECT  *
WHERE
  { ?x  <http://www.w3.org/2000/01/rdf-schema#label>  "Dances with Wolves"@en ;
        a                      <http://dbpedia.org/ontology/Film>
  }
```

Results

This panel shows up to 10 mappings that result from executing the reverse engineered query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

<dbpedia:Dances_with_Wolves>

Extra results

This panel shows up to 10 mappings that result from executing a relaxed query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

Thirupathi <dbpedia:Thirupathi> (type: `http://dbpedia.org/ontology/Film`)

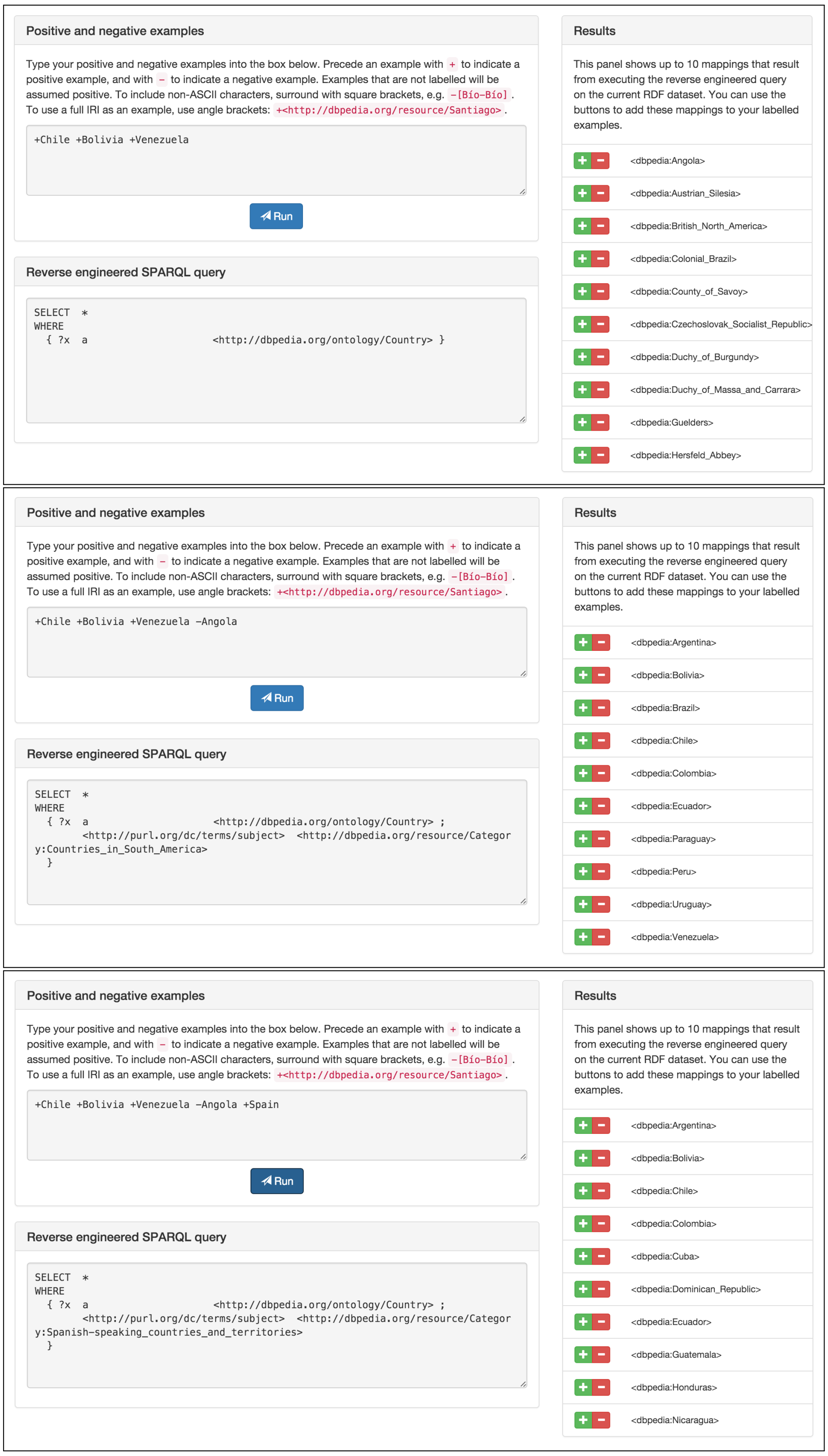
Newcastle (film) <dbpedia:Newcastle_(film)> (type: `http://dbpedia.org/ontology/Film`)

Figure 4.12: SPARQLByE interface after having input two movies as labelled examples. The Extra Results Panel suggests further movies.

Example 4.10. *Consider a user who wishes to obtain a list of authors along with their places of death, the latter being relevant only if they have passed away. If the user uses the positive examples* `{Lewis,Oxford}`, `{Tolkien,Dorset}`, *and* `{Rowling, NULL}`, *and the negative examples* `{11th_Dalai_Lama,Tibet}` *and* `{Abbey_Lincoln, Manhattan}`, *then the resulting reverse-engineered SPARQL query is:*

```
SELECT * WHERE {
  ?x rdf:type <http://dbpedia.org/ontology/Writer> .
  ?x rdf:type <http://xmlns.com/foaf/0.1/Person> .
  ?x rdf:type <http://dbpedia.org/ontology/Artist>
  OPTIONAL {
    ?x <http://dbpedia.org/ontology/deathPlace> ?y
  }
}
```

The previous query thus allows SPARQLByE to present to the user the desired result set. □



Positive and negative examples

Type your positive and negative examples into the box below. Precede an example with + to indicate a positive example, and with – to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. `-[Bío-Bío]`. To use a full IRI as an example, use angle brackets: `+<http://dbpedia.org/resource/Santiago>`.

```
+Chile +Bolivia +Venezuela
```

Run

Reverse engineered SPARQL query

```
SELECT  *
WHERE
  { ?x  a                     <http://dbpedia.org/ontology/Country> }
```

Results

This panel shows up to 10 mappings that result from executing the reverse engineered query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

<dbpedia:Angola>
<dbpedia:Austrian_Silesia>
<dbpedia:British_North_America>
<dbpedia:Colonial_Brazil>
<dbpedia:County_of_Savoy>
<dbpedia:Czechoslovak_Socialist_Republic>
<dbpedia:Duchy_of_Burgundy>
<dbpedia:Duchy_of_Massa_and_Carrara>
<dbpedia:Guelders>
<dbpedia:Hersfeld_Abbey>

Positive and negative examples

Type your positive and negative examples into the box below. Precede an example with + to indicate a positive example, and with – to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. `-[Bío-Bío]`. To use a full IRI as an example, use angle brackets: `+<http://dbpedia.org/resource/Santiago>`.

```
+Chile +Bolivia +Venezuela -Angola
```

Run

Reverse engineered SPARQL query

```
SELECT  *
WHERE
  { ?x  a                     <http://dbpedia.org/ontology/Country> ;
        <http://purl.org/dc/terms/subject>  <http://dbpedia.org/resource/Categor
y:Countries_in_South_America>
  }
```

Results

This panel shows up to 10 mappings that result from executing the reverse engineered query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

<dbpedia:Argentina>
<dbpedia:Bolivia>
<dbpedia:Brazil>
<dbpedia:Chile>
<dbpedia:Colombia>
<dbpedia:Ecuador>
<dbpedia:Paraguay>
<dbpedia:Peru>
<dbpedia:Uruguay>
<dbpedia:Venezuela>

Positive and negative examples

Type your positive and negative examples into the box below. Precede an example with + to indicate a positive example, and with – to indicate a negative example. Examples that are not labelled will be assumed positive. To include non-ASCII characters, surround with square brackets, e.g. `-[Bío-Bío]`. To use a full IRI as an example, use angle brackets: `+<http://dbpedia.org/resource/Santiago>`.

```
+Chile +Bolivia +Venezuela -Angola +Spain
```

Run

Reverse engineered SPARQL query

```
SELECT  *
WHERE
  { ?x  a                     <http://dbpedia.org/ontology/Country> ;
        <http://purl.org/dc/terms/subject>  <http://dbpedia.org/resource/Categor
y:Spanish-speaking_countries_and_territories>
  }
```

Results

This panel shows up to 10 mappings that result from executing the reverse engineered query on the current RDF dataset. You can use the buttons to add these mappings to your labelled examples.

<dbpedia:Argentina>
<dbpedia:Bolivia>
<dbpedia:Chile>
<dbpedia:Colombia>
<dbpedia:Cuba>
<dbpedia:Dominican_Republic>
<dbpedia:Ecuador>
<dbpedia:Guatemala>
<dbpedia:Honduras>
<dbpedia:Nicaragua>

Figure 4.13: SPARQLByE interface converging towards a query that retrieves Spanish-speaking countries.

4.9 Experimental Evaluation

In this section we describe the experimental settings with which we have studied the implementation of the reverse engineering algorithms. More precisely, we first perform a study on synthetically generated inputs, and then we attempt to reverse engineer SPARQL queries in a real-world setting. For the second scenario, as we do not have access to positive examples provided by users, we indirectly obtain these through the use of query logs. All algorithms have been implemented in Java and run on a machine with a 2.3 GHz Intel Core i7 processor and 16 GB of main memory. The SPARQL endpoint was hosted by Virtuoso (open source version) and the connection to the endpoint was implemented using Apache Jena.

4.9.1 Reverse engineering random inputs

To test the efficiency and usefulness of our approach for reverse engineering SPARQL queries, we tested Algorithm 2 over synthetically generated inputs. Although generating a random RDF graph D and a random set of mappings Ω is possible, in practice the pair will usually not be definable. For this reason, we opt to first generate a random query $Q \in \mathrm{SP}[\mathsf{AOwd}]$, and construct the pair (D_Q, Ω_Q) from Q. We now discuss the generation of these inputs.

The experimental setup will focus on investigating the effect of **OPT**-depth, whereby random queries are generated as linear pattern trees, i.e. queries of the form $(P_0\,\mathbf{OPT}(P_1\,\mathbf{OPT}\cdots(P_{n-1}\,\mathbf{OPT}\,P_n)\cdots))$, where each $P_i \in \mathrm{SP}[\mathsf{A}]$. The depth n of each query Q is varied as a parameter. In each node P_i we place a set of triple patterns of the form $t = (s, p, o)$, where $p \in \mathsf{U}$ and $s, o \in \mathsf{U} \cup \mathsf{V}$. We aim to include joins between variables in these queries, and to this end, for each subquery P_i and each variable v which is mentioned in P_i but not in any P_j for $j < i$, we include a triple pattern of the form (u, c, v), where u is a variable mentioned in P_{i-1} and c is a random URI (notice that u is taken from P_{i-1} to obtain a well-designed graph pattern). In total, $\sim$900 random queries were generated, where $\sim$100 queries are generated with each depth $n \in [0, 8]$.

For each random query Q we next generate two distinct inputs (D_1, Ω) and (D_2, Ω) to be submitted to the learning algorithm. First, we generate an RDF graph D_Q by *freezing* the query Q: for each prefix $Q' \trianglelefteq Q$,[1] we convert the set of triple patterns in Q' into a set of triples by replacing the variables with fresh constants.

[1]Recall that such prefixes are formed by replacing a subquery (R **OPT** S) by R, as defined in Section 3.2.2.

Example 4.11. *Assume that* $Q = (?\mathtt{X}, \mathtt{type}, \mathtt{Country})\ \mathbf{OPT}\ (?\mathtt{X}, \mathtt{label}, ?\mathtt{Y})$*. Then for the prefix* $Q_1 = (?\mathtt{X}, \mathtt{type}, \mathtt{Country})$ *of* Q*, we create the triple* $(\mathtt{X\_0}, \mathtt{type}, \mathtt{Country})$ *by replacing variable* $?\mathtt{X}$ *by constant* $\mathtt{X\_0}$*. Moreover, for the prefix* $Q_2 = Q$*, we generate the triples* $(\mathtt{X\_1}, \mathtt{type}, \mathtt{Country})$*,* $(\mathtt{X\_1}, \mathtt{label}, \mathtt{Y\_1})$ *by following the same approach, where* $\mathtt{X\_1}$ *and* $\mathtt{Y\_1}$ *are fresh constants (different from* $\mathtt{X\_0}$*). Thus, we have that* $D_Q = \{(\mathtt{X\_0}, \mathtt{type}, \mathtt{Country}), (\mathtt{X\_1}, \mathtt{type}, \mathtt{Country}), (\mathtt{X\_1}, \mathtt{label}, \mathtt{Y\_1})\}$*.*

The RDF graph D_Q is now used to obtain the full set of answers $\Omega_Q = Q(D_Q)$. For both the inputs (D_1, Ω) and (D_2, Ω) to the algorithm, we extract samples from D_Q and Ω_Q. First, let $\Omega \subseteq \Omega_Q$ be a uniform sample of Ω_Q containing at most 100 mappings. Second, let $D_1 = D_Q$ and $D_2 \subseteq D_Q$ be a uniform sample of D_Q, where each triple in D_Q has a 75% chance of being included in D_2. The first input (D_1, Ω) is the full frozen RDF graph D_Q and a sample of the answers in Ω_Q as positive examples, and it is thus expected to be definable by construction. The second input (D_2, Ω) uses a reduced RDF graph (and the same positive examples), whereby it is possible that this pair will no longer be definable. This random elimination of triples is done in order to generate inputs to the algorithm which are not definable, and thus study the compared performance of the system on both definable and non-definable inputs.

For each random query Q, both pairs (D_1, Ω) and (D_2, Ω) are input into a Java implementation of Algorithm 2, for a total of ~1800 distinct runs of the algorithm. Of these, 1064 definable cases and 761 non-definable cases are reported. Figure 4.14 shows the results of these experiments. The top plot in Figure 4.14 shows the runtime of the algorithm in milliseconds versus the size of the input (defined to be the sum of the number of triples in the RDF graph and the number of mappings in the set of mappings). As this is a logarithmic plot, the exponential dependency is clear, both for the definable and undefinable cases. On the other hand, the bottom plot in Figure 4.14 shows the runtime versus the size of the random query itself, exhibiting a very similar (and exponential) performance behaviour. The average runtime for this set of examples was ~516ms. For the definable examples, the average ratio between the size of the reverse-engineered query P and the original randomly generated query Q was 0.79; this reflects the fact that reverse-engineered queries tend to be relaxed versions of the original query, and thus contain less triple patterns. Similarly, the average difference between the **OPT**-depth of the learned query and the original random query is -1.33, indicating that learned queries tend to use less **OPT** operators. This actually depends on the positive examples, as the **OPT**-depth of the learned query will be exactly the depth of the structure of the set of mappings Ω. Finally, note that merely 32 of the 1064 definable cases learned a query which mentioned a constant that was not originally included in the random query.

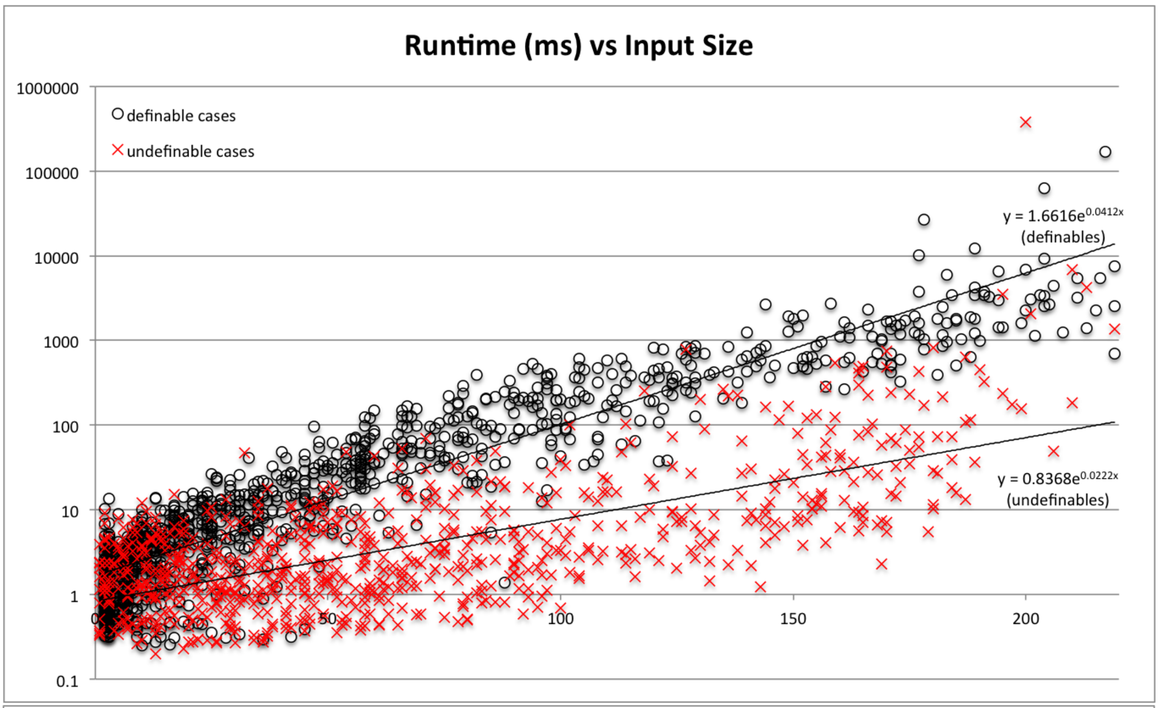
4.9.2 Reverse engineering DBpedia query logs

In order to test our implementation of the reverse engineering algorithm on real-world examples, we turn to the public DBpedia SPARQL endpoint. DBpedia (version 2014) was downloaded and installed locally into the Virtuoso Open Source database manager (version 7.1.0). The DBpedia RDF graph contains over 860 million triples, and the contents is extracted from the Wikipedia and Wikimedia websites. DBpedia provides a public SPARQL endpoint where users may perform SPARQL queries, and the logs for this endpoint are made publicly available as well. To access and study these query logs, we turn to the LSQ project [91], which has extracted and organised the DBpedia query logs.

Although our main goal is to obtain sets of positive examples which can then be input into our reverse engineering algorithm, such sets of positive examples are not available, as no SPARQL reverse engineering or query-by-example systems exists which publish these data. Hence, we use the DBpedia query logs to indirectly obtain positive examples. For a query Q from the query logs, we execute the query on the DBpedia RDF graph, which we denote by D_{Dbpedia}, to obtain $\Omega = [\![Q]\!]_{D_{\text{DBpedia}}}$. This set of mappings represents the full set of examples that the user was supposedly interested in when executing query Q, and a random sample $\Omega' \subseteq \Omega$ of Ω can serve as a hypothetical set of positive examples from said user.

Of the ~740,000 queries in the query logs (queries which were executed on the DBpedia SPARQL endpoint), there are ~220,000 queries which make use of the **OPT** operator but not the **FILTER** or **UNION** operators, and ~34,000 which use both the **OPT** and **FILTER** operators but not **UNION**. To understand the sets of mappings which are to be expected, ~124,000 of the 220,000 queries on D_{DBpedia} were selected, and for each such query Q we obtain the full set of results $[\![Q]\!]_{D_{\text{DBpedia}}}$. Of these sets of mappings, none had a structure whose depth was greater than 1; more precisely, 116 of these queries produced an answer set of depth 1, and the rest produced depth 0 (i.e. they were homogeneous). This suggests that a reverse engineering algorithm may be parametrised to produce queries whose **OPT**-depth is limited to 1.

We next selected ~30,000 queries and replicated the procedure from the previous section for randomly generated queries. That is, each query Q was executed on D_{DBpedia} to obtain $\Omega = [\![Q]\!]_{D_{\text{DBpedia}}}$, and a random sample $\Omega' \subseteq \Omega$ was used as the set of positive examples (in many cases the query has exactly one mapping as an answer,



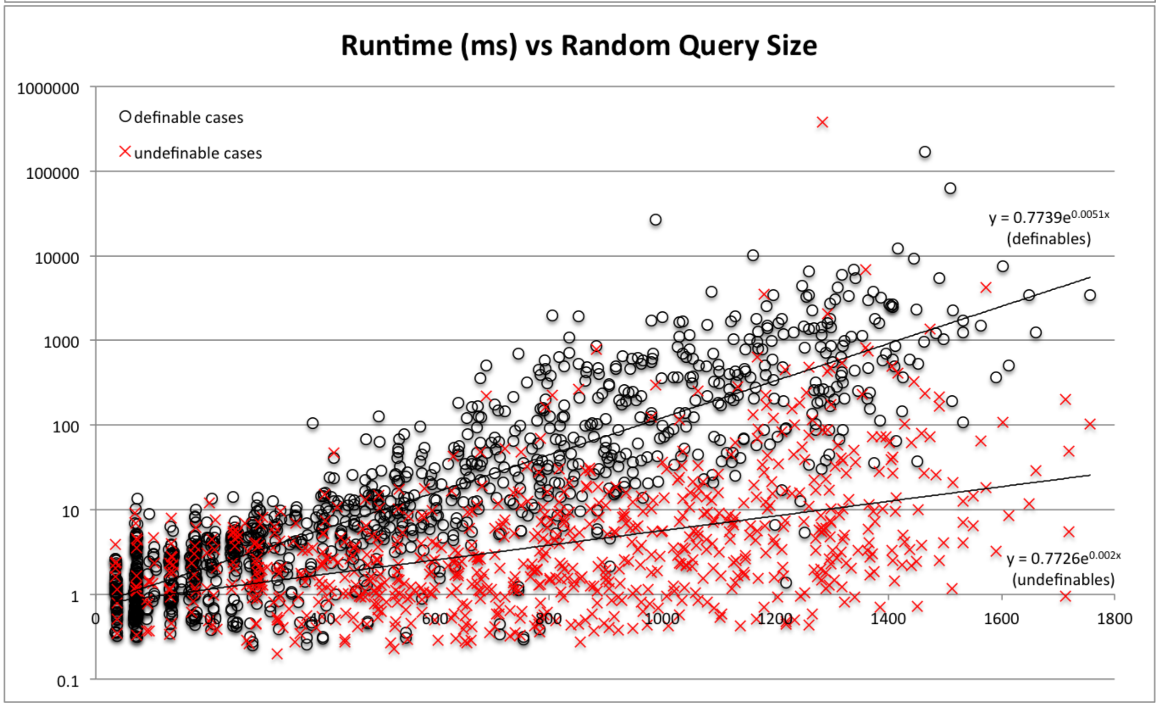


Figure 4.14: Runtimes for ∼2000 randomly generated examples. Top: runtime versus input size; circles (black) represent definable inputs and crosses (red) represent undefinable inputs. Bottom: runtime versus random query size.

in which case we simply use this mapping as the sole positive example). The pair $(D_{\text{DBpedia}}, \Omega')$ then was used as the input. In this experimental setting the average runtime for all queries was 35ms and the average ratio between the learned query size and the original query size was 0.28. These low values can be explained by the fact that many learned queries only have one triple pattern.

To illustrate the behaviour of the algorithm on a slightly more complicated query, we manually prepare the following query Q_2:

```
SELECT * WHERE {
?country type Country .
?country usesTemplate Infobox_country .
OPTIONAL {
?country languages ?language
}
OPTIONAL {
?country2 type Country .
?country wikiLink ?country2 .
?country2 usesTemplate Infobox_country .
?country2 subject Former_Spanish_colonies
}
}
```

The answer set $\Omega = [\![Q_2]\!]_{D_{\text{DBpedia}}}$ has ∼860 results, and its structure is tree-like of depth 1. A sample $\Omega' \subseteq \Omega$ from Ω is extracted and the pair $(D_{\text{DBpedia}}, \Omega')$ are input into the algorithm. The query learned by the greedy algorithm was similar to Q_2, but without the two triple patterns which mention the URI `Infobox_country`. The similarity between learned query and original query is an indication that the algorithm in general gives high quality results.

Finally, we showcase the algorithm for the RevEng$^+$ (SP[$\mathsf{AOF}_{\wedge,=,\neq}\mathsf{wd}$]) decision problem by slightly altering the query Q_2 and adding a single **FILTER** expression, resulting in:

```
SELECT * WHERE {
?country type Country .
OPTIONAL {
?country languages ?language
}
OPTIONAL {
?country2 type Country .
?country2 subject Former_Spanish_colonies .
?country wikiLink ?country2 .
FILTER ( ?country != ?country2 )
}
}
```

The following is the query learned by the algorithm:

```
SELECT * WHERE {
?country type Country .
OPTIONAL {
?country languages ?language
}
OPTIONAL {
?country2 type Country .
?country2 subject Former_Spanish_colonies .
?country wikiLink ?country2 .
?country2 usesTemplate RefList .
FILTER ( ?country != ?country2 )
}
}
```

Note that the algorithm was able to successfully learn the query, although in this case an extra triple pattern has been added.

4.10 System limitations and future work

Although arbitrary tree-like example sets are supported in the algorithms developed in the previous chapter, the meaning of a "negative partial example" (i.e. a negative example with NULLs) is unintuitive for users — it could mean that there is no mapping that matches the example exactly, or no mapping that subsumes the example. In order to stick to an intuitive user-interface, we have restricted SPARQLByE to use only "full negative examples": negative examples must mention all variables included in any positive example. Future work should address possible definitions of partial negative examples.

Another key limitation we have found in practice is the fact that very generic relations commonly justify the positive examples that users provide. For example, for almost any set of arity 1 positive examples (i.e. entities), the triple pattern (`?x`, `type`, `Thing`) [2] will be enough to generate a — useless — reverse engineered query. Thus, it is not enough for the system to greedily generate candidate triple patterns, but must do so in an order that ensures that more informative triple patterns are generated first.

This is not an issue that we currently address from an algorithmic point of view. SPARQLByE does allow user-driven customization of the reverse-engineering algorithm to avoid both over-fitting and over-generalization, though. In practice, users can select a set of "forbidden URIs" (e.g. from the current result set) and the reverse-engineering algorithm will then avoid generating a query containing these.

Another limitation of SPARQLByE lies in the fact that the queries returned merely satisfy the restrictions, but do not necessarily do so in the most interesting fashion. Triple patterns should not necessarily be generated in a random order (or, more precisely, whatever order is used by the SPARQL query engine upon executing the auxiliary queries), but perhaps should maintain a ranking of more popular relations, or more informative relations. We currently address this issue with the "forbidden URIs" facility, but being an ex post addition to the system, it will probably not address the underlying problem in every scenario.

Another interesting approach to the problem of returning increasingly more useful results to the user is the use of ensemble methods, whereby more than one query is reverse engineered with different objectives (e.g. a highly specific query, a more general query, an minimally sized query, etc). The resulting ensemble of queries can be used to provide results to the user according to some aggregation heuristic such as voting. These techniques are common in the context of machine learning and could enhance query reverse engineering systems as well.

The fragment of SPARQL that has been considered in the algorithm design of the previous chapter is also a limitation of the system, especially when considering the variety of features SPARQL includes that cannot be reverse engineered in the current setup. Examples include projection, unions, blank nodes, path expressions, negations, and more. In particular, restricting the reverse engineering to queries which do not use projection has the consequence that only resulting entities which are directly connected in the RDF graph may be considered.

[2] In full: (`?x`, `rdf:type`, `http://www.w3.org/2002/07/owl#Thing`)

As such, future work should prioritise expanding the fragment of SPARQL that is reverse engineer-able. While full inclusion of projection is desirable, an intermediate approach could be to offer the possibility of including unbounded variables into positive and negative example mappings. Such an addition would permit reverse engineering a restricted form of projection, where the number of existential variables in the reverse engineered query is fixed by the input. This addition would be similar to the role blank nodes fill by allowing a querier to indicate that an entity must exist which satisfies the restrictions of the query, but without requiring said querier to specify which entity.

More graph-native features of SPARQL such as path expressions should also be incorporated, along with many other features of the SPARQL language, including the UNION and negation, for example. In these operators lies a wealth of expressiveness of the SPARQL query language that users routinely use, and thus may expect to be able to reverse engineer. Recent analyses on real-world SPARQL queries [30, 74] show that real world SPARQL queries tend to be relatively simple, although in [30] Bonifati et. al. observe that 15% of them make use of the projection operator, showing a clear need for its incorporation into any comprehensive reverse engineering system.

Directions for future advancement will surely draw from the extensive literature in the field, where the state of the art continues to advance with results on reverse engineering aggregate queries [104] and select-project-join queries [114], for example.

Lastly, another essential limitation of our approach lies in the data itself (which, as an input, cannot be chosen). If the data lacks the relations or the associations to justify a set of positive examples, our reverse engineering approach will not find an appropriate query. For example, if the user seeks cars whose engines are designed in Japan, it is likely that DBpedia will not have a category for such cars, and no set of positive and negative examples will satisfy the user. To address this issue it may be necessary to complement reverse engineering techniques with data completion approaches, which we discuss in the next chapter.

A screencast of the system is available at: `www.cs.ox.ac.uk/michael.benedikt/sparqlbye.ogg`.

Chapter 5

Semantic Embeddings for RDF

5.1 Introduction

A growing corpus of human knowledge is being encoded in the form of *knowledge graphs*, that is, in the form of (s, p, o) triples which represent simple subject-predicate-object sentences. In essence, these triple-based formats consist of binary relational data, where an (s, o) pair is effectively declared to be related by the p binary relation. Although more complex—or higher arity—information is difficult to express in this way, the simple syntax enjoyed by these triple-based data models has proved highly useful for a wide range of applications and has thus been widely adopted. Today, the linked open data cloud consists of hundreds of interlinked datasets, including a handful of very large knowledge graphs such as Freebase [25] and DBpedia [19], which contain billions of triples and millions of entities.

Modern knowledge graphs have reached a level of maturity that allows them to be used in applications such as web search, where graph data provides structured information that complements hyperlink answers, artificial intelligence question answering systems such as Watson and voice-based assistants, and semantic web query engines based on SPARQL. However, there are still important issues to be resolved. Even the largest knowledge graphs are extremely incomplete (i.e. many true facts have yet to be encoded as triples) and prone to errors (often triples encoding incorrect facts are included). The main tasks of link prediction, which consists of predicting new triples, and triplet classification, which seeks to assign a probability that a certain triple—be it new or existing—is true or not, look to address these shortcomings. In this context, there is renewed interest in machine learning over (binary) relational data. The techniques used vary from rule-based learning [51] to tensor factorisation, neural network-based approaches, etc. (for a review on the area, see [79]). In this work we focus on latent feature-based techniques, which are also known as graph embeddings.

Graph embeddings correspond to latent feature statistical relational learning models in that they assume the existence of a set of n latent features—random variables—that account for the predictions we desire (triplet classification or otherwise). As such, these latent features provide machine learning-friendly representations of graph data, which can then be used as input to further machine learning tools such as neural networks or logistic regression for classification. The input to a graph embedding model, as with other statistical relational learning techniques, is a knowledge graph, and the output is a mapping which associates to each entity in the graph the set of n real-valued latent features. The name *graph embedding* refers to the geometrical interpretation given to the obtained latent features: the entities are interpreted as points in an n-dimensional real coordinate space, and thus are said to have been embedded into said space. On the other hand, the relations of the knowledge graph are varyingly represented as translational vectors [31], matrix transformations [81], etc.

Graph embedding models are usually trained via the minimisation of a global cost function, which can be expressed as the sum of a cost assigned to each triple in the knowledge graph. The minimisation itself is achieved via stochastic gradient descent or similar methods. One of the greatest advantages of graph embedding models, and especially translational models (loosely defined as those models which represent relations as one or more translational vectors, as opposed to transformation matrices), is that they are able to perform efficiently for very large graphs and with high accuracy. Current graph embedding models achieve high performance on basic learning tasks such as link prediction, e.g. ranking the correct entity among the top ten candidates almost 95% of the time [80].

Despite the relatively positive scenario described above, one of the greatest limitations of current graph embedding models is that they consider a very simple model of the data, ignoring the rich semantics associated with triples in more expressive data models developed by the semantic web community, such as RDF and the associated ontology languages. As mentioned previously, a knowledge graph is considered as nothing more than a set of triples, with no special semantics attached to any particular triple. In contrast, modern RDF data are often accompanied by rich ontological information which encodes a wealth of metadata, including type hierarchies and other constraints. Current graph embedding models are entirely agnostic to such metadata, and thus ontological triples are usually manually removed before training (or, if they are included, simply interpreted as plain data triples). One of the consequences of this is that current graph embedding models do not directly incorporate constraints that humans would consider obvious (e.g. understanding that a human cannot be friends with a building). A huge potential exists in the rich ontologies that inform modern knowledge graphs, and the objective of this work is to explore ways in which such ontologies can become first class citizens of graph embedding models.

More specifically, we consider the relatively unexplored problem of graph embedding on ontology-rich knowledge graphs, where the ontology is specified in a standard language such as RDFS (more expressive languages such as OWL2 have been developed, but will not be considered in this study). Drawing on the geometrical interpretation that graph embeddings give to their latent features (namely, that entities are embedded as points in a real coordinate space), we explore the case where RDFS classes are correspondingly modelled as sets of points (i.e. volumes) in the same coordinate space, and relations are embedded as sets of pairs of points. This generalisation of the basic geometrical interpretation allows for a natural expression of ontological constraints in the global cost function. The result is a model we name EmbedS, which is able to model RDFS ontological constraints as first-class citizens. Along with providing the precise definitions for the EmbedS cost function, we provide experimental results that show EmbedS to be comparable to state-of-the-art graph embedding techniques when measured on traditional benchmark knowledge graphs, while performing well on a new ontology-rich dataset we have prepared for the purposes of studying the new enriched geometrical interpretation that EmbedS provides.

The results presented in this chapter showcase the potential of extending current graph embedding research to include ontological information, and will hopefully encourage further development of the area. The specific contributions to research in the field are as follows:

1. The definition of a translation-based graph embedding model which specifically embeds graph entities, classes, and relationships as points, sets of points, and sets of pairs of points respectively, thus providing a geometrical interpretation to the embedding space.

2. An extensive experimental evaluation of said model, showing performance that is able to compete with state-of-the-art embedding models, while offering the possibility of a new mode of triple classification based on the geometrical interpretation of the embedding space.

In this chapter we first introduce necessary mathematical notation, preliminary definitions, and we discuss related work. We then develop and introduce the EmbedS model, its global cost function, and the geometrical interpretation induced on embedded entities and relations. Next, we explain the experimental setting and evaluation metrics, including a discussion of an ontology-rich benchmark dataset we have prepared for testing the EmbedS model, including a discussion of results obtained. This chapter is based on work published in [39].

5.2 Preliminaries and related work

5.2.1 Definitions and notation

Graph embedding models are usually defined on a *knowledge graph*, which is essentially a set of triples, similar to the RDF data model, but lacking specialised features of the latter, such as the precise definition of IRIs, literals, and the semantics associated with certain keyword IRIs. In this section we will introduce necessary definitions, noting the similarities and discrepancies with the RDF data model. We will then introduce needed concepts such as graph embeddings, and finally we will discuss related work.

Let E and P be two mutually disjoint and countably infinite sets of *entities* and *relations*, respectively. A knowledge graph is then defined as a finite set of triples of the form $(s, p, o) \in \mathsf{E} \times \mathsf{P} \times \mathsf{E}$. As such, a knowledge graph can be interpreted as a directed graph with labelled edges, or as a set of binary relations. As in this work will seek to allow graph embeddings to operate on more expressive graph data, it is important to note the differences with the full RDF data model. RDF considers entities and relations to be indistinguishable members of a larger set of IRIs: $\mathsf{E} \subseteq \mathsf{I}$ and $\mathsf{P} \subseteq \mathsf{I}$, without enforcing their disjointness (thus, a relation is simultaneously considered to be a node in the graph). Also, RDF defines an infinitely countable set of *literals* L, disjoint from I, and finally RDF triples are drawn from the more general set $\mathsf{I} \times \mathsf{I} \times (\mathsf{I} \cup \mathsf{L})$. Note that we have omitted the discussion of blank nodes, which are placeholder entities that do not refer to any particular real-world concept, but merely indicate the *existence* of an entity.

The base RDF data model described provides very little semantics other than the basic interpretation of the triple as a fact. To enrich an RDF dataset, several ontology languages have been developed, of which RDFS is one of the simplest. RDFS defines the core keyword IRIs shown in Table 5.1, which are assigned special semantics in order for richer knowledge—including basic type systems—to be encoded into RDF

Keyword	**Abbr.**	**IRI**
type	`type`	`rdf:type`
subclass	`sc`	`rdfs:subClassOf`
disjoint	`dsjnt`	`owl:disjointWith`
subproperty	`sp`	`rdfs:subPropertyOf`
domain	`dom`	`rdfs:domain`
range	`range`	`rdfs:range`

Table 5.1: RDFS core keywords and their abbreviations used in this chapter.

format. Note that the prefixes used in Table 5.1 are themselves abbreviations. Actually, `rdf:` expands to `http://www.w3.org/1999/02/22-rdf-syntax-ns#` and `rdfs:` expands to `http://www.w3.org/2000/01/rdf-schema#`.

In what follows, we will consider, in addition to the sets E and P, and additional set C of *classes*, also infinitely countable and disjoint from the previous two. Furthermore, we define six special relations $\texttt{type}, \texttt{sc}, \texttt{sp}, \texttt{dom}, \texttt{range}, \texttt{dsjnt} \in \mathsf{P}$. We will only consider triples $t = (s, p, o)$ for which one of the following hold:

- $(s, p, o) \in \mathsf{E} \times \mathsf{P} \times \mathsf{E}$ (called data triples),
- $(s, p, o) \in \mathsf{E} \times \{\texttt{type}\} \times \mathsf{C}$ (called type triples). Type triples are used to indicate that an entity belongs to a class, e.g. $(\texttt{ATuring}, \texttt{type}, \texttt{Person})$.
- $(s, p, o) \in \mathsf{C} \times \{\texttt{sc}\} \times \mathsf{C}$ (called subclass triples). Subclass triples are used to indicate a containment relation among classes, e.g. $(\texttt{Actor}, \texttt{sc}, \texttt{Person})$
- $(s, p, o) \in \mathsf{C} \times \{\texttt{dsjnt}\} \times \mathsf{C}$ (called disjointness triples). Disjointness triples denote disjointness among classes, e.g. $(\texttt{Person}, \texttt{dsjnt}, \texttt{Building})$.
- $(s, p, o) \in \mathsf{P} \times \{\texttt{sp}\} \times \mathsf{P}$ (called subproperty triples). Subproperty triples denote relations between properties. For example, if the property `directorOf` is in triples such as $(\texttt{Spielberg}, \texttt{directorOf}, \texttt{Avatar})$, it could be indicated that $(\texttt{directorOf}, \texttt{sp}, \texttt{workedOn})$.
- $(s, p, o) \in \mathsf{P} \times \{\texttt{dom}\} \times \mathsf{C}$ (called domain triples). Domain triples are used to indicate the class of entities which may be the subject of a triple using a certain property, e.g. $(\texttt{directorOf}, \texttt{dom}, \texttt{Person})$.
- $(s, p, o) \in \mathsf{P} \times \{\texttt{range}\} \times \mathsf{C}$ (called range triples). Range triples are used to indicate the class of entities which may be the object of a triple using a certain property, e.g. $(\texttt{bornIn}, \texttt{range}, \texttt{Place})$.

A *data graph* D is defined as a finite set of triples such that for every triple $t \in D$, t is a data triple or a type triple (thus, type triples are considered to be a form of data triple as well). An *ontology* S (which corresponds to a simplified version of an RDFS ontology) is a finite set of triples such that for every triple $t \in S$, t is not a data triple and t is not a type triple. Finally, an ontology-rich graph is a pair $I = (D, S)$ which consists of a data graph and an ontology (we will often abuse notation and consider $I = D \cup S$).

Example 5.1. *Consider the following data graph D:*

$$D = \{(\texttt{anne}, \texttt{type}, \texttt{Woman}), (\texttt{john}, \texttt{type}, \texttt{Man}), (\texttt{john}, \texttt{knows}, \texttt{anne})\}.$$

The previous information about people and their relationships can be enriched with the following ontology S:

$$S = \{(\texttt{Woman}, \texttt{sc}, \texttt{Person}), (\texttt{Man}, \texttt{sc}, \texttt{Person})\}.$$

The full dataset is then $I = D \cup S$. □

Having introduced the syntax of the simplified version of RDFS that will be used, we now turn to its semantics. Crucially, RDFS allows for inferencing new triples using a series of inference rules. For example, the following is an inference rule for RDFS, for some $x \in \mathsf{E}$ and $B, C \in \mathsf{C}$:

$$\frac{\begin{array}{c}(x, \texttt{type}, C)\\(C, \texttt{sc}, B)\end{array}}{(x, \texttt{type}, B)}$$

Which is read as follows: given a dataset $I = D \cup S$ which contains two triples of the form $(x, \texttt{type}, C)$ and $(C, \texttt{sc}, B)$ (recall that $x \in \mathsf{E}$ and $B, C \in \mathsf{C}$), the triple $(x, \texttt{type}, B)$ may be *inferred* to hold. This particular inference rule is one of many such rules [13], and we denote a general inference rule with $t_1, \ldots, t_n \to t$. Informally, inference rules work in the following way: given an ontology-rich graph $I = D \cup S$ such that there exists an inference rule $t_1, \ldots, t_n \to t$ and an assignment σ of the variables of the rule to URIs such that $\sigma(t_1), \ldots, \sigma(t_n) \in I$, then the triple $\sigma(t)$ is added to I, defining the *inferred* graph $I' = I \cup \{t\}$. The fully inferred graph is then the result of applying all possible inference rules—and adding all possible inferred triples—until a fixed point is reached [13]. In query engines that recognise RDFS ontologies, inferred triples are considered true in the same sense as explicit triples.

Example 5.2. *Consider the ontology-rich graph $I = D \cup S$ from the previous example. Here, we have the following applicable inference rule:*

$$\frac{\begin{array}{c}(x, \texttt{type}, B)\\(B, \texttt{sc}, C)\end{array}}{(x, \texttt{type}, C),}$$

with the following assignment: $\sigma(x) = \texttt{anne}$*,* $\sigma(B) = \texttt{Woman}$*,* $\sigma(C) = \texttt{Person}$*, giving the following instantiation of the rule:*

$$\frac{\begin{array}{c}(\texttt{anne}, \texttt{type}, \texttt{Woman})\\(\texttt{Woman}, \texttt{sc}, \texttt{Person})\end{array}}{(\texttt{anne}, \texttt{type}, \texttt{Person}),}$$

We therefore define the inferred graph $I' = I \cup \{(\texttt{anne}, \texttt{type}, \texttt{Person})\}$*. In this way, the semantics of RDFS allow us to conclude facts which are not explicitly included in the dataset.* □

RDFS and ontology languages in general are thus a powerful tool for representing knowledge about how data interacts. They can be used to specify type hierarchies and other restrictions on data. It is the goal of this work for such metadata to be considered in the modelling phase of graph embeddings.

5.2.2 Graph embeddings and related work

Graph embedding models and, in general, statistical relational learning models, vary greatly in the techniques used in order to obtain machine learning-compatible representations of knowledge graphs. In general, however, the main problem of knowledge base completion gives a common direction to these techniques: constructing statistical models of the data that allow for link prediction (given an incomplete fact such as $(\texttt{Tarantino}, \texttt{inspiredBy}, ?)$ and return the entity that would mostly likely complete the triple) and triple classification (given a triple, assign a probability that it is true).

Informally, the problem statement is usually as follows: given a knowledge graph D, define a series of latent variables $x_1, \ldots, x_n$ and an objective function $f(x_1, \ldots, x_n; D)$ such that the minimisation of $f(x_1, \ldots, x_n; D)$ is such that given a triple t the probability $P(t, X_1, \ldots, X_n; D)$ of said triple is maximised when t is *true*. It is of course difficult to define the trueness of a triple, especially when confronted to new triples extracted from sources of unknown accuracy (as is the case of web-based extraction of triples, for example). For this reason, statistical models are usually trained with a standard division of the dataset into training, validation, and testing subsets. The variables $x_1, \ldots, x_n$ mentioned are known as *parameters*, and their values are the subject to successive updates during the training phase, which essentially consists in minimising the objective function. Only the training subset is considered for updating the parameters. The final assignment of values to the parameters defined is called a solution, and its generalisation power is evaluated on the validation and testing subsets.

The objective function f can also depend on constants which are not modified during training, but whose value must also be chosen a priori; these constants are known as *hyperparameters*. A whole sub-field of machine learning is dedicated to researching how such hyperparameters should be assigned values [5, 22, 40].

The most expressive models involve tensor or matrix factorisation techniques, although these are also the models with the highest complexity, measured in the number of parameters that must be trained. A well-known example of this is RESCAL [81], which explains triples using pairwise interactions of the latent features of entities. Thus, the *cost* associated to a triple x_{ijk} has the following form:

$$\mathsf{cost}(x_{ijk}) = \mathbf{e}_i^\top \mathbf{W}_k \mathbf{e}_j.$$

Other highly expressive models use techniques such as matrix factorisation [65], neural tensor networks [97], and multilayer perceptrons [77]. The latter, also known as word2vec, was strictly a word embedding model, but had as an interesting and unintended consequence a translational property among the latent representation of words: simple binary relations between words could be captured when interpreting the latent representation—*embedding*—of the relation as a *translational vector*. For example, after training on a textual corpus, researchers found that incomplete sentences such as $(\texttt{Madrid}, \texttt{capitalOf}, \texttt{Spain})$ had the property that the vector $\mathbf{e}_{\texttt{Madrid}} + \mathbf{e}_{\texttt{capitalOf}}$ was nearest to $\mathbf{e}_{\texttt{Spain}}$ (where $\mathbf{e}_{\mathsf{x}}$ indicates the embedding assigned to entity $x \in \mathsf{E}$).

The previous translational property spawned a renewed interest in distance-based models that incorporated a *geometrical interpretation* for the latent representations of entities. The first model in this sub-field was TransE [31], and was quickly followed by refinements such as TransH [113], TransR [71], and others. In TransE, the cost of a triple x_{ijk} is as follows:

$$\mathsf{cost}(x_{ijk}) = \mathbf{dist}(\mathbf{e}_i + \mathbf{p}_k, \mathbf{e}_j),$$

where **dist** is the standard euclidean distance function. Notice that TransE now embeds a relation $p_k \in \mathsf{P}$ as an n-dimensional vector $\mathbf{p}_k$ which acts as a translational vector. While less expressive, translation-based models prove more scalable, as their model complexity is lower and a simpler cost structure allows for more efficient training.

The model proposed in this work draws from research in translation-based models. However, our problem scenario focused on ontology-rich knowledge graphs, which contain specific semantics for keyword triples. In this space, work has been done in combining deep neural networks with logic rules [62] and recently, with the design of semantic loss functions [116]. In [34], type constraints are considered by removing type-constraint-violating triples, which improves training speed and link prediction performance. They do not incorporate ontological information into the model as a first class citizen, however, and neither does the geometrical interpretation of the model adapt to reflect the presence of this metadata.

More generally, tools such as Markov logic networks have been used to generate interest probability graphs in knowledge graphs such as WordNet [98].

5.3 The EmbedS model

Consider an RDF dataset $I = D \cup S$, and let $E_I \subseteq \mathsf{E}$, $C_I \subseteq \mathsf{C}$, and $P_I \subseteq \mathsf{P}$ be the sets of entities, classes, and properties that appear in I, respectively. Define for each entity $e_i \in E_I$ an n-dimensional vector of parameters (i.e. variables) $\mathbf{e}_i = (e_{i1}, \ldots, e_{in})$; for each class $c_i \in C_I$ define an n-dimensional vector of parameters $\mathbf{c}_i = (c_{i1}, \ldots, c_{in})$ and a parameter ρ_i; and for each property $p_i \in P_I$ define two n-dimensional vectors of parameters $\mathbf{p}_i^\alpha = (p_{i1}^\alpha, \ldots, p_{in}^\alpha)$ and $\mathbf{p}_i^\beta = (p_{i1}^\beta, \ldots, p_{in}^\beta)$ and a parameter σ_i. We have thus defined a total of $|E_I| \cdot n + |C_I| \cdot (n+1) + |P_I| \cdot (2n+1)$ parameters for our model.

Assuming that $|E_I| = N_E$, $|C_I| = N_C$, and $|P_I| = N_P$, the cost $\mathcal{L}$ will be a function of all the variables previously defined:

$$\begin{aligned}\mathcal{L} = \mathcal{L}(\ & \mathbf{e}_1, \ldots, \mathbf{e}_{N_E}; \mathbf{c}_1, \ldots, \mathbf{c}_{N_C}, \rho_1, \ldots, \rho_{N_C}; \\ & \qquad \mathbf{p}_1^\alpha, \ldots, \mathbf{p}_{N_P}^\alpha, \mathbf{p}_1^\beta, \ldots, \mathbf{p}_{N_P}^\beta, \sigma_1, \ldots, \sigma_{N_P}\ ).\end{aligned}$$

We first provide an overview as to the objective of the model and how it will be trained. Having defined the set of parameters and the global cost function which depends on these parameters, training will proceed via stochastic gradient descent of the objective function, as shown in Algorithm 5. The final state of the parameters after training is called the solution.

For what follows, we first define a distance function **dist** which assigns to every pair of n-dimensional vectors $\mathbf{x} = (x_1, \ldots, x_n)$, and $\mathbf{y} = (y_1, \ldots, y_n)$ a non-negative real value $\mathbf{dist}(\mathbf{x}, \mathbf{y})$, with the standard distance function properties[1]. In what follows,

[1](a) $\mathbf{dist}(\mathbf{x}, \mathbf{y}) \geq 0$, (b) $\mathbf{dist}(\mathbf{x}, \mathbf{y}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{y}$, (c) $\mathbf{dist}(\mathbf{x}, \mathbf{y}) = \mathbf{dist}(\mathbf{y}, \mathbf{x})$, and (d) $\mathbf{dist}(\mathbf{x}, \mathbf{z}) \leq \mathbf{dist}(\mathbf{x}, \mathbf{y}) + \mathbf{dist}(\mathbf{y}, \mathbf{z})$.

ALGORITHM 5: Pseudo-code for training and embedding model.

```
Input: Training graph $I_t$.
Output: Graph embedding solution.
1 for $i = 1$ to $N_{epochs}$ do
2     Calculate cost $\mathcal{L}$ of current solution;
3     Calculate gradient $\nabla\mathcal{L}$;
4     Update parameters in direction of gradient;
5 end
```

we choose to set $\mathbf{dist}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \sum_{i=1}^{i=n}(x_i - y_i)^2$, that is, the L_2 norm of $\mathbf{x} - \mathbf{y}$. The L_1 norm is also commonly used in the context of graph embeddings. However, in this case the geometrical interpretation is central to the design of the model, whereby changing the distance function may have unintended side effects. We also define an activation function **act**, for which we choose the rectifier function, $\mathbf{act}(x) = \max(0, x)$.

We now turn to the precise form of the cost function $\mathcal{L}$. For the entire dataset, define $\mathcal{L}^I = \sum_{t \in I} \mathcal{L}_t$, where the cost of each triple $t = (e_i, p_k, e_j) \in I$ (note that $e_i, e_j \in E_I \cup C_I \cup P_I$ and $p_k \in P_I$), is defined as follows:

$$\mathcal{L}_t = \mathbf{act}\Big(\mathbf{dist}(\mathbf{e}_i, \mathbf{p}_k^{\alpha}) + \mathbf{dist}(\mathbf{e}_j, \mathbf{p}_k^{\beta}) - \sigma_k\Big).$$

The geometrical interpretation of this cost is provided in Section 5.3.1, and is visualized in Figure 5.1. Next, we define a cost term for each possible RDFS assertion. For each $t \in S$:

1. If $t = (e_i, \texttt{type}, c_j) \in S$, where $e_i \in E_I$ and $c_j \in C_I$, define:

$$\mathcal{L}_t^S = \mathbf{act}\Big(\mathbf{dist}(\mathbf{e}_i, \mathbf{c}_j) - \rho_j\Big),$$

2. If $t = (c_i, \texttt{sc}, c_j) \in S$, where $c_i, c_j \in C_I$, define:

$$\mathcal{L}_t^S = \mathbf{act}\Big(\mathbf{dist}(\mathbf{c}_i, \mathbf{c}_j) - (\rho_j - \rho_i)\Big),$$

3. If $t = (p_i, \texttt{sp}, p_j) \in S$, where $p_i, p_j \in P_I$, define:

$$\mathcal{L}_t^S = \mathbf{act}\Big(\mathbf{dist}(\mathbf{p}_i^{\alpha}, \mathbf{p}_j^{\alpha}) + \mathbf{dist}(\mathbf{p}_i^{\beta}, \mathbf{p}_j^{\beta}) - (\sigma_j - \sigma_i)\Big),$$

4. If $t = (p_i, \mathrm{dom}, c_j) \in S$, where $p_i \in P_I$ and $c_j \in C_I$, define:

$$\mathcal{L}_t^S = \mathbf{act}\Big(\mathbf{dist}(\mathbf{p}_i^{\alpha}, \mathbf{c}_j) - \sigma_i\Big),$$

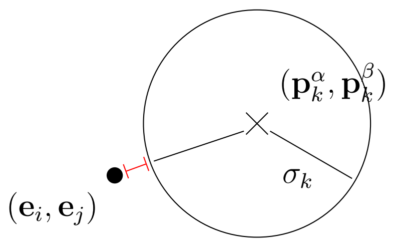

$(\mathbf{p}_k^\alpha, \mathbf{p}_k^\beta)$
σ_k
$(\mathbf{e}_i, \mathbf{e}_j)$

Figure 5.1: Cost of a triple $(e_i, p_k, e_j) \in I$. The circle represents a $2n$-sphere of radius σ_k centred at $(\mathbf{p}_k^\alpha, \mathbf{p}_k^\beta)$. The pair (e_i, e_j) is embedded as the $2n$-dimensional point $(\mathbf{e}_i, \mathbf{e}_j)$. The (red) error line shows the cost.

5. If $t = (p_i, \texttt{range}, c_j) \in S$, where $p_i \in P_I$ and $c_j \in C_I$, define:

$$\mathcal{L}_t^S = \mathbf{act}\Big(\mathbf{dist}(\mathbf{p}_i^\beta, \mathbf{c}_j) - \sigma_i\Big).$$

Finally, we sum, for every triple $t \in S$, the cost of t depending on the RDFS relation it mentions. In that way, we define the cost term $\mathcal{L}^S = \sum_{t \in S} \mathcal{L}_t^S$, and thus define the final cost to be $\mathcal{L} = \mathcal{L}^I + \lambda \mathcal{L}^S$, where λ is a hyperparameter that assigns a larger weight to the ontological triples (and thus must its value must be fixed before training).

5.3.1 Geometrical interpretation

We now give a geometrical interpretation to the model definition. By embedding entities as single n-dimensional vectors we are modelling them as *points* in the n-dimensional euclidean space. Classes, on the other hand, are modelled as *regions* of the euclidean space, being embedded as a vector and a radius, representing n-spheres. This allows for the following geometrical interpretation: if an entity is embedded within the region defined by the embedding of a class, then it is interpreted to be of that type, and vice-versa (see Figure 5.2). Finally, properties, insofar as they represent binary relations, are modelled as $2n$-spheres which constitute a set of *pairs of points*. Thus, each relation $p_k \in P_I$ has an embedding which consists of two n-dimensional vectors $\mathbf{p}_k^\alpha$ and $\mathbf{p}_k^\beta$ and a radius ρ_k. The corresponding geometrical interpretation is analogous to the previous case: in $2n$-space, a pair (e_i, e_j) is interpreted to be related by a relation p_k if the $2n$-point $(\mathbf{e}_i, \mathbf{e}_j)$ is in the region defined by the $2n$-sphere centered at $(\mathbf{p}_k^\alpha, \mathbf{p}_k^\beta)$ with radius σ_k (see Figure 5.1).

The main advantage conferred by this ontology-aware geometrical interpretation is that RDFS classes and ontological assertions are now first-class citizens of the model. By modelling classes as regions of the euclidean space, for example, certain

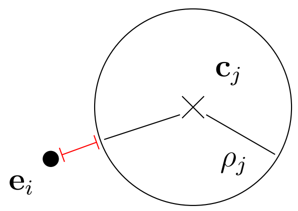


Figure 5.2: Cost (red error line) of $t = (e_i, \texttt{type}, c_j)$, where class c_j is embedded as an n-sphere at $\mathbf{c}_j$ with radius ρ_j.

properties are obtained for free, such as type containment transitivity: if after training the entity `alanTuring` is (correctly) embedded within the class `Researcher`, and said class is (correctly) embedded fully contained within the region corresponding to class `Person`, then the transitive fact that `alanTuring` is a `Person` will be provided for free. In this way, the embedding space will presumably encode ontological information geometrically.

5.3.2 Negative examples

Cost-based graph embedding models must include negative examples in order to avoid trivial solutions from becoming valid. In particular, when trained with only positive examples, cost-based models will tend to optimise the global cost function by assigning the value zero to every parameter in the model, thus obtaining a trivial solution which does, in fact, optimise the cost function. The challenge, then, is to generate a suitable set of negative examples (triples) in the context of incomplete data, where it is not known whether missing triples are missing because they are false or merely because they have not been found to be true yet.

The simplest mechanism for generating negative triples would be to generate entirely random triples $\tilde{t} = (e_i, p_k, e_j)$ with the only restriction being that $\tilde{t} \notin I$. However, this recipe in most cases results in negative triples that are not useful for training: as the universe $\mathsf{E}_I \times \mathsf{P}_I \times \mathsf{E}_I$ is extremely large, the sampled triples will usually be nonsensical (e.g. $(\texttt{Jamaica}, \texttt{birthPlace}, \texttt{Saturday})$). Although a nonsensical triple is technically a valid negative example, training can be performed much more efficiently if negative examples delineate the *border* between true and false facts, that is, negative examples should be as similar as possible to positive triples in order to serve to discriminate between true and false information. As such, translation-based models often include negative examples in the form of *corrupt triples*. During a training epoch, every triple $t = (e_i, p_k, e_j) \in I$ is assigned a randomly generated corrupt triple t' where one of the following hold (i) $t' = (e_{i'}, p_k, e_j)$ for $i \neq i'$, (ii) $t' = (e_i, p_{k'}, e_j)$ for

$k \neq k'$, or (iii) $t' = (e_i, p_k, e_{j'})$ for $j \neq j'$. The previous ensures that the negative example will not be completely random, but rather derived from a positive triple. This configuration of negative example generation is known as *uniform sampling.* In the literature, the corruption of the relation—alternative (ii)—is not usually performed; when it is, the scenario is called uniform sampling *with relation corruption.*

Generating negative examples via triple corruption partially corrects the situation, but not entirely. In fact, research has been done into more refined negative sampling configurations. We now consider one of these configurations.

Bernoulli—or *bern*—sampling was designed as an alternative way of generating negative examples at training time [113]. While uniform sampling generates, for each triple $t = (e_i, p_k, e_j)$ in the training dataset a corresponding negative triple $t' = (e_{i'}, p_k, e_{j'})$ such that either $i' \neq i$ or $j' \neq j$ (but not both), Bernoulli sampling attempts to generate negative triples that more closely model information that is incorrect (as opposed to merely senseless).

The procedure is as follows. Firstly, before training time, for each relation $p_k \in P_I$ the ratio of subjects to objects is calculated as the average number of subjects related to each object through p_k:

$$\text{sto}(p_k) = \frac{\sum_{e_j \in \mathsf{E}_I \in \text{arange}(p_k)} |\{e_i \in \mathsf{E}_I \mid (e_i, p_k, e_j) \in I\}|}{|\text{arange}(p_k)|},$$

where $\text{arange}(p_k) = \{e_j \in \mathsf{E}_I \mid \exists e \in \mathsf{E}_I : (e, p_k, e_j)\}$. Analogously, the ratio of objects to subjects is calculated as the average number of objects related to each subject through p_k:

$$\text{ots}(p_k) = \frac{\sum_{e_i \in \text{adom}(p_k)} |\{e_j \in \mathsf{E}_I \mid (e_i, p_k, e_j) \in I\}|}{|\text{adom}(p_k)|},$$

where $\text{adom}(p_k) = \{e_i \in \mathsf{E}_I \mid \exists e \in \mathsf{E}_I : (e_i, p_k, e)\}$.

The following value is then obtained for p_k:

$$\text{bernratio}(p_k) = \frac{\text{sto}(p_k)}{\text{sto}(p_k) + \text{ots}(p_k)},$$

$\text{bernratio}(p_k)$ thus will be used as follows: given a triple $t = (e_i, p_k, e_j)$ in the training dataset, t will be corrupted on the left with probability $p = \text{bernratio}(p_k)$ (thus generating $t' = (e_{i'}, p_k, e_j)$) and on the right with probability $1 - p$.

5.4 Experimental evaluation

In this section we provide preliminary experimental results showing that EmbedS can perform at state-of-the-art levels on a standard benchmark dataset, while providing a new complementary triple classification method based on the geometrical interpretation which shows encouraging results. An exhaustive experimental evaluation will be left for future work. The model was implemented in the Python Theano framework, and builds on the open source code provided by the authors of [31].

5.4.1 Datasets

We use the following datasets for experimental evaluations:

wn18 Dataset extracted from WordNet. This dataset consists of 151,442 triples, 40,943 entities, and 18 relations.

dbpedia32k RDF dataset extracted from DBpedia, consisting of 340,827 triples, 32,657 entities, and 296 relations.

The first dataset has become a standard benchmark in the graph embedding field [31], and allows for an apples to apples comparison between our model and existing work. However, EmbedS has been designed for a fundamentally different problem setting: that of embedding in *ontology-rich* RDF data. In order to test the performance of our model, we have prepared a dataset, named 'dbpedia32k' which includes an RDFS ontology.

We now describe the construction of the dbpedia32k dataset. It initially draws from three distinct downloads, which are freely available: the 'DBpedia Ontology' file, which contains ontology triples, the 'Instance Types' file, which contains data triples of the form $(a, \texttt{type}, C)$ for some entity a and some class C, and the 'Mappingbased Objects' file, which contains general data triples, representing facts on entities present in Wikipedia articles. From the Mappingbased Objects file first define a relation to be useful if it appears in at least 1000 triples. We define an entity to be useful if it is mentioned at least 10 times in the file. A uniform sample of useful entities is built, keeping only triples which mention useful relations and these sampled entities only. Finally, we recalculate useful entities (in the filtered dataset) and remove triples which mention non-useful entities. We thus obtain the final set of Mappingbased Objects triples to be used. We complete the dataset by extracting, from Instance Types, triples $(a, \texttt{type}, C)$ where a is mentioned in the previous dataset, and similarly we extract relevant ontology triples from DBpedia Ontology. The resulting complete dataset is split uniformly into three subsets for training, validation, and testing, with proportions 0.8, 0.1, and 0.1, respectively.

5.4.2 Link prediction performance

To measure the performance of the model, we use the standard *mean rank* and *hits@10* metrics, each in its two standard flavours: *raw*, *filtered*. We now describe these metrics in detail.

Ranking-based performance metrics are designed to answer the question: if the model were asked to rank the best entities to complete a partial triple (i.e. to predict a link), how well would it rank the correct answer?

More precisely, for every triple $t = (s, p, o)$ in the testing dataset, a *left rank* $\mathrm{lr}(t)$ is calculated by removing the subject of the triple—thus obtaining an incomplete triple $t_l = (?, p, o)$—and subsequently, testing every possible entity $s' \in \mathsf{E}$, calculating the corresponding cost $\mathsf{cost}(t_{s'})$ of the candidate triple $t_{s'} = (s', p, o)$. The candidate triples $t_{s'}$ are then sorted in ascending order of cost. The *raw* left rank of triple t is then defined as its position in the sorted list (the first position being defined as rank 1).

For the *filtered* left rank, the sorted list of triples t' is first filtered, that is, removing all triples $t_{s'}$ for which $t_{s'} \in I$, that is, which happen to be true. An analogous definition holds for the right rank of triple t. Finally, the raw (filtered) mean rank of the testing dataset is the mean of the raw (filtered) left ranks and raw (filtered) right ranks of each testing triple. With the previous, the following is defined:

- **hits@10 left** is defined as the proportion of testing triples t such that $\mathrm{lr}(t) \leq 10$, and analogously for **hits@10 right**.
- The **mean rank left (right)** is defined as the mean of the left (right) ranks of each testing triple t.

5.4.3 Hyperparameter tuning

Selection of hyperparameters of the model is achieved via random search. Although more sophisticated methods have been proposed [5], random search has been shown to be competitive with these systems [22].

More precisely, for each dataset-model combination (e.g. dbpedia32k with EmbedS), a suitable bounding box for the hyperparameters of the model is chosen, and training is performed over 500 epochs for 1,000 different random assignments of hyperparameter values. The final model selected is that which maximizes the estimated mean reciprocal rank for the validation dataset.

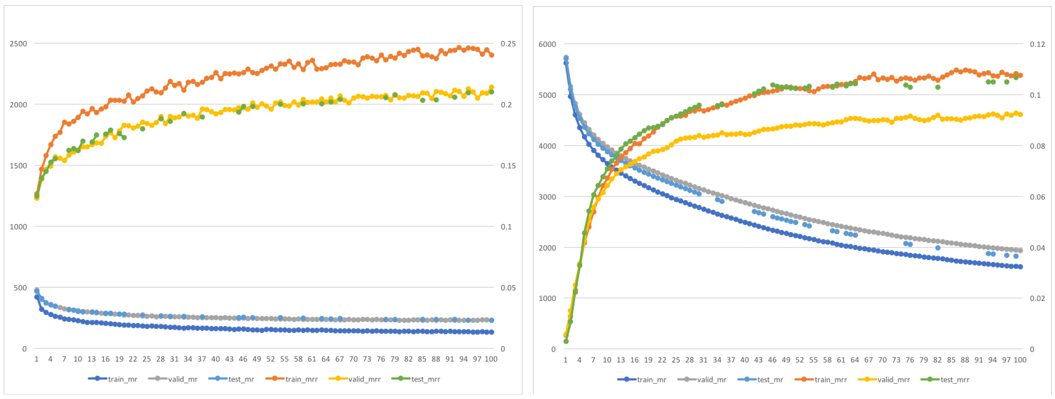


Figure 5.3: Learning process over 1000 epochs with sgd updates (left) and adagrad updates (right). Partial evaluation is done every 10 epochs.

5.4.4 Gradient descent

Optimisation of the global cost function is generally achieved via stochastic gradient descent. However, there are known issues with this approach. Notably, stochastic gradient descent maintains a constant step size which can be detrimental in two scenarios: (i) a relatively large step size may produce a quick convergence at the beginning of training at the expense of an unstable progress near the end of training, where small adjustments of the parameters are needed, or (ii) a relatively small step size, which would be optimal for the later stages, at the expense of a slow convergence at earlier stages.

In response to this, several different update techniques have been developed, including adagrad [41]. We have implemented a version of EmbedS which uses adagrad updates.

Figure 5.3 shows that the adoption of adagrad degraded the performance of the model, going from 52.0% hits@10f to 23.3%. It must be noted, however, that adagrad would probably require a different set of hyperparameters to achieve best performance; towards this hypothesis, the experiment was repeated with larger learning rates (lrparam set to 10.0 and lremb set to 0.01), obtaining 35.4%.

5.4.5 Bern sampling

Figure 5.4 shows the result of implementing Bernoulli sampling on EmbedS. The horizontal axis corresponds to time: a partial evaluation of link prediction performance is calculated on the training, validation, and testing datasets every ten epochs (therefore, although training was executed over a total of 1000 epochs, the plot shows 100 partial evaluations performed). Performance was slighty degraded on the benchmark dataset fb15k.

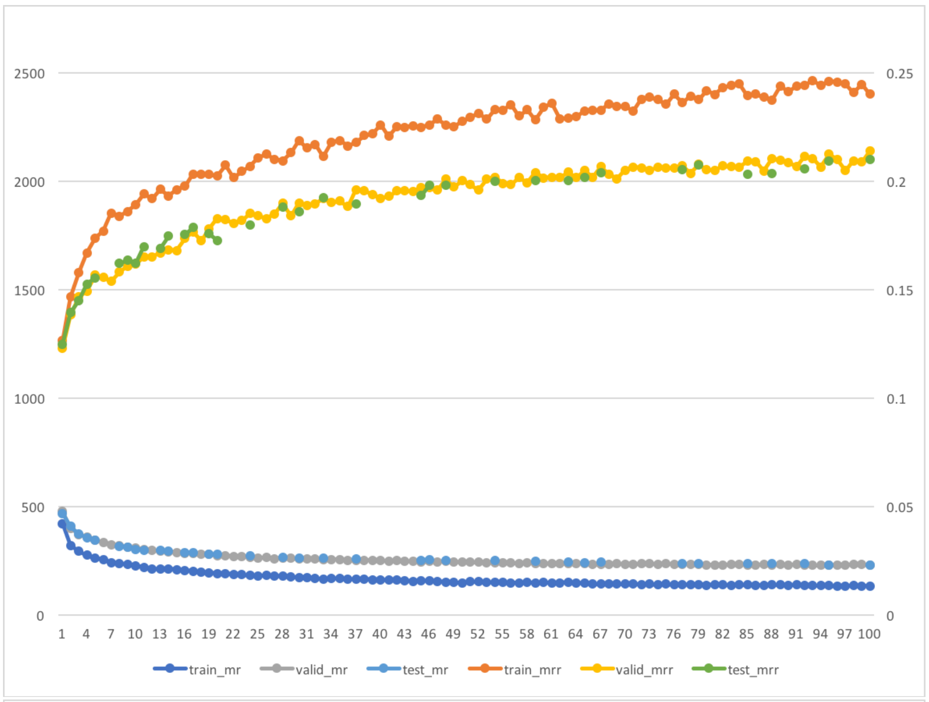
5.4.6 Optimising triple classification

As EmbedS allows for a geometrical interpretation of triples, we also evaluate a triple classification performance. For each triple in the dataset, and an equal amount of randomly generated false triples (i.e. triples not in the dataset), if $t = (e_i, p_k, e_j)$, the binary classification consists in asking whether the $2n$-dimensional point $(\mathbf{e}_i, \mathbf{e}_j)$ is contained in the sphere centred at $(\mathbf{p}_k^\alpha, \mathbf{p}_k^\beta)$ with radius σ_k or not. Precision and recall values are obtained for this test.

5.4.7 Optimal results

EmbedS was trained on the wn18 dataset, optimizing for best validation (filtered) hits@10 value, obtaining 94.9%, which is comparable to state-of-the-art models such as HolE [80]. HolE is clearly superior in the mean reciprocal rank metric, however, with a value of 0.938, compared to 0.560 for EmbedS. It must be noted, though, that these values correspond to *harmonic* mean ranks of 1.07 for HolE and 1.79 for EmbedS. If EmbedS is now instructed to optimise the geometrical interpretation, we achieve a precision of 84.2% and a recall of 83.9%, corresponding to an f-measure of 84.0%.

On the dbpedia_v2 dataset, we find that EmbedS achieves a performance on hits@10 of 22.7% and a mean reciprocal rank of 0.133 (corresponding to a harmonic mean rank of 7.52). TransE, on the other hand, performs at 11.6% hits@10 and 0.054 mean reciprocal rank (corresponding to a harmonic mean rank of 18.52).



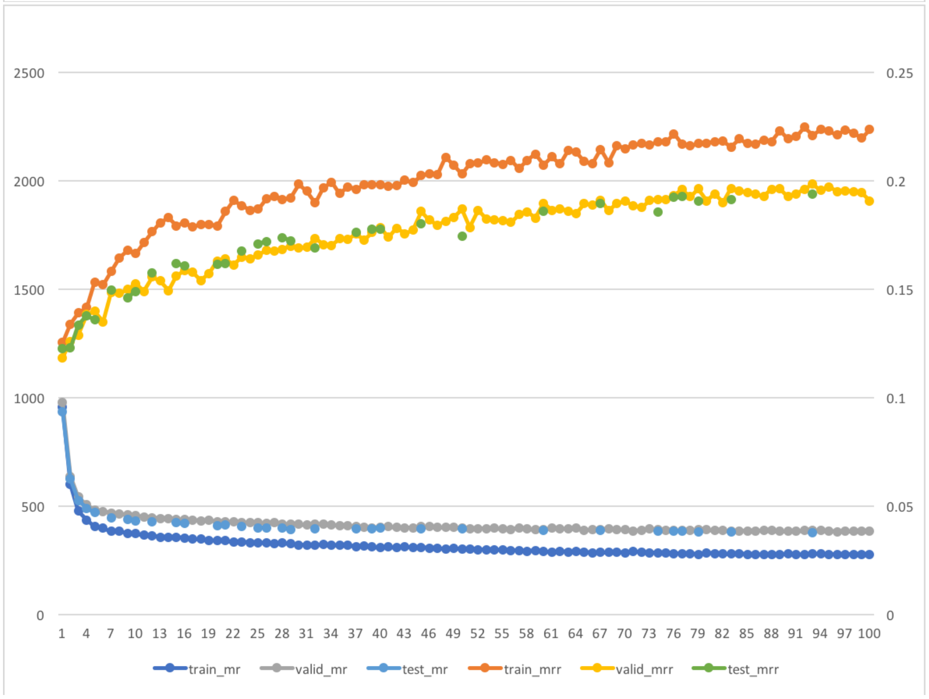


Figure 5.4: Learning process over 1000 epochs with uniform negative example sampling (top) and Bernoulli negative example sampling (bottom). Partial evaluation is done every 10 epochs.

Chapter 6

Conclusion

Wider adoption of linked open data applications depends on several factors, one of them being the development of the theory and tools that ease the exploration and querying of unknown databases. A key obstacle is the unfamiliarity of users with the structure of data, as well as the structure of the classical querying paradigm itself, in which the user must be capable of formulating queries in a query formal language, such as SPARQL. In this context, it is valuable to study extensions, or outright alternatives, to this classical querying paradigm. Querying by example and reverse engineering in particular are areas of research that address this concern.

In this thesis we have explored the problem of increasing the usability of knowledge graph database systems with learning techniques, including the study of definability problems, reverse engineering problems, and machine learning based methods for knowledge base completion.

To summarise the contributions of this thesis, we have studied the computational complexity of the first-order logic definability problem (Chapter 2), FO-Def, and the generalised version, BP-Pairs, which have been found to be **GI**-complete, thus closing two open problems in the database area. A fundamental corollary of these results is that FO-Def can be solved efficiently if the graph isomorphism problem can be solved efficiently. The incremental approach taken by the polynomial-time algorithm for FO-Def with an oracle for the graph isomorphism problem may prove applicable to other scenarios as well, and may open avenues of further research.

In the context of Semantic Web systems (Chapter 3), the reverse engineering problem has been studied for various fragments of the SPARQL query language. Our study of the complexity of the problem indicates that restricting the examples so that the associate lattice is tree-like has significant benefits to the complexity of reverse engineering. Based on this study we developed a reverse engineering procedure that proceeds by building a single candidate query and checking its correctness via a call to a SPARQL query engine. We have examined the performance of this algorithm both on synthetically generated and real-world query examples, profiling both performance and quality of the results. In particular, we found exponential dependency of the runtime on the input size, which was to be expected given our complexity results. However, an important component of the complexity of the algorithms originates from the final checking step of the candidate query. Hence, it is in principle possible to present this candidate query to the user quickly, and to proceed with the final check in the background.

Next, in order to explore the usability of database systems and the query-by-example paradigm, we have demonstrated a system for querying RDF data "by example" (Chapter 4). On the one hand, a user does not have to write SPARQL queries; indeed, the user does not even have to understand SPARQL. On the other hand, the system exploits the power of the SPARQL language for expressing natural user queries, by inducting a SPARQL query from user examples "under the hood". This user interface serves as a proof-of-concept for potential end user systems that allow audiences without technical knowledge of formal query languages to use interactive learning techniques to converge on precise queries and, thus, explore graph-based knowledge bases. However, we found that an important limitation was the quality and/or the completeness of the data. For example, a user who wishes to retrieve a list of all entities of a highly specific, or otherwise obscure, class, will not find query engineering to meet their needs, simply because the data does not contain enough explicit information. Addressing this issue was the main motivation for the last part of the thesis.

Finally, we have presented the new problem of training graph embeddings on ontology-rich datasets (Chapter 5). Graph embeddings are a standard technique for generating statistical models of data that allow for predicting new links in existing data, a concept also known as knowledge base completion. In this way, a graph embedding approach has the potential of becoming an essential ingredient in the toolbox of providing more navigable databases. We proposed a graph embedding model which considers RDFS classes and other ontological information as first-class citizens, providing a geometrical interpretation for triples and for ontology assertions. Experimental results show that the model can perform at state-of-the-art levels on standard benchmark datasets, while on ontology-rich datasets it is also able to provide an alternative form of triple classification which takes advantage of the geometrical interpretation.

In order for the geometrical interpretation of this model to be used to its maximum capacity, it will be necessary to perform an exhaustive random search including all hyperparameters, with the objective of finding a configuration which allows for high performance results both on the standard ranking-based metrics and in the geometry-based triple classification test. There may well be an inherent trade-off which restricts this possibility, in which case the use-cases of the model may be more restricted, but still valuable, as the model would serve as a unified framework where upon choosing the performance metric to be prioritised (i.e. ranking based link prediction versus geometry-based triple classification) the same model can be optimised accordingly.

The theoretical results on computational complexity and algorithms, the implementation of query-by-example user interfaces, and the design of machine learning based formalisms for knowledge base completion, represent steps forward in addressing the issues that the field faces. However, there are several tasks outstanding, such as extending our query reverse engineering complexity bounds and algorithms to larger fragments of the SPARQL query language, in order to capture a broader subset of its full expressivity. Applying query reverse engineering techniques to other newer graph based data models and query languages will serve to extend the query by example paradigm and explore its usefulness in different use cases and application domains. It will also be valuable to propose frameworks in which the machine learning based knowledge graph completion concepts can be merged with query by example systems in order to provide users with a consistent interface for exploring both explicitly stated and learned data. The ongoing goal of increasing the efficiency of accessing data in databases, and making this technology available and usable by wider audiences is one that is far from closed, and remains very much as exciting as it was decades ago, when query languages and databases were still in their infancy.

Bibliography

[1] Scott Aaronson, Greg Kuperberg, and Christopher Granade. Complexity Zoo. `https://complexityzoo.uwaterloo.ca`, 2005.

[2] 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*. Springer, 2007.

[3] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[4] Azza Abouzied, Dana Angluin, Christos H. Papadimitriou, Joseph M. Hellerstein, and Avi Silberschatz. Learning and verifying quantified boolean queries by example. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 49–60. ACM, 2013.

[5] Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3981–3989, 2016.

[6] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980.

[7] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[8] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.

[9] Timos Antonopoulos, Frank Neven, and Frédéric Servais. Definability problems for graph query languages. In Tan et al. [103], pages 141–152.

[10] Marcelo Arenas and Gonzalo I. Diaz. The exact complexity of the first-order logic definability problem. *ACM Trans. Database Syst.*, 41(2):13, 2016.

[11] Marcelo Arenas, Gonzalo I. Diaz, and Egor V. Kostylev. Reverse engineering SPARQL queries. In Jacqueline Bourdeau, Jim Hendler, Roger Nkambou, Ian Horrocks, and Ben Y. Zhao, editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 239–249. ACM, 2016.

[12] Marcelo Arenas, Bernardo Cuenca Grau, Evgeny Kharlamov, Sarunas Marciuska, and Dmitriy Zheleznyakov. Faceted search over rdf-based knowledge graphs. *J. Web Sem.*, 37-38:55–74, 2016.

[13] Marcelo Arenas, Claudio Gutierrez, and Jorge Pérez. Foundations of RDF databases. In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutierrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors, *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures*, volume 5689 of *Lecture Notes in Computer Science*, pages 158–204. Springer, 2009.

[14] Marcelo Arenas and Jorge Pérez. Querying semantic web data with SPARQL. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 305–316. ACM, 2011.

[15] Marcelo Arenas and Martín Ugarte, editors. *18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium*, volume 31 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[16] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[17] Vikraman Arvind and Jacobo Torán. Isomorphism testing: Perspective and open problems. *Bulletin of the EATCS*, 86:66–84, 2005.

[18] DBpedia Association. DBpedia. `https://wiki.dbpedia.org/`, 2007.

[19] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In Aberer et al. [2], pages 722–735.

[20] László Babai. Graph isomorphism in quasipolynomial time. *CoRR*, abs/1512.03547, 2015.

[21] François Bancilhon. On the completeness of query languages for relational data bases. In Józef Winkowski, editor, *Mathematical Foundations of Computer Science 1978, Proceedings, 7th Symposium, Zakopane, Poland, September 4-8, 1978*, volume 64 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 1978.

[22] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[23] Alexander Bilke and Felix Naumann. Schema matching using duplicates. In Karl Aberer, Michael J. Franklin, and Shojiro Nishio, editors, *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 69–80. IEEE Computer Society, 2005.

[24] 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.

[25] Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In Jason Tsong-Li Wang, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pages 1247–1250. ACM, 2008.

[26] Angela Bonifati, Radu Ciucanu, and Aurélien Lemay. Learning path queries on graph databases. In Gustavo Alonso, Floris Geerts, Lucian Popa, Pablo Barceló,

Jens Teubner, Martín Ugarte, Jan Van den Bussche, and Jan Paredaens, editors, *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 109–120. OpenProceedings.org, 2015.

[27] Angela Bonifati, Radu Ciucanu, Aurélien Lemay, and Slawek Staworko. A paradigm for learning queries on big data. In Rada Chirkova and Jun Yang, editors, *Proceedings of the First International Workshop on Bringing the Value of "Big Data" to Users, Data4U@VLDB 2014, Hangzhou, China, September 1, 2014*, page 7. ACM, 2014.

[28] Angela Bonifati, Radu Ciucanu, and Slawek Staworko. Interactive inference of join queries. In Sihem Amer-Yahia, Vassilis Christophides, Anastasios Kementsietsidis, Minos N. Garofalakis, Stratos Idreos, and Vincent Leroy, editors, *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014.*, pages 451–462. OpenProceedings.org, 2014.

[29] Angela Bonifati, Radu Ciucanu, and Slawek Staworko. Interactive join query inference with JIM. *PVLDB*, 7(13):1541–1544, 2014.

[30] Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large SPARQL query logs. *PVLDB*, 11(2):149–161, 2017.

[31] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Burges et al. [32], pages 2787–2795.

[32] Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors. *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, 2013.

[33] Ashok K. Chandra and David Harel. Computable queries for relational data bases. *J. Comput. Syst. Sci.*, 21(2):156–178, 1980.

[34] Kai-Wei Chang, Wen-tau Yih, Bishan Yang, and Christopher Meek. Typed tensor decomposition of knowledge bases for relation extraction. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014*

Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, pages 1568–1579. ACL, 2014.

[35] Sara Cohen and Yaacov Y. Weiss. Certain and possible xpath answers. In Tan et al. [103], pages 237–248.

[36] Sara Cohen and Yaacov Y. Weiss. Learning tree patterns from example graphs. In Arenas and Ugarte [15], pages 127–143.

[37] Jan Van den Bussche. Applications of alfred tarski's ideas in database theory. In Laurent Fribourg, editor, *Computer Science Logic, 15th International Workshop, CSL 2001. 10th Annual Conference of the EACSL, Paris, France, September 10-13, 2001, Proceedings*, volume 2142 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2001.

[38] Gonzalo I. Diaz, Marcelo Arenas, and Michael Benedikt. SPARQLByE: Querying RDF data by example. *PVLDB*, 9(13):1533–1536, 2016.

[39] Gonzalo I. Diaz, Achille Fokoue, and Mohammad Sadoghi. Embeds: Scalable, ontology-aware graph embeddings. In Michael H. Böhlen, Reinhard Pichler, Norman May, Erhard Rahm, Shan-Hung Wu, and Katja Hose, editors, *Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018.*, pages 433–436. OpenProceedings.org, 2018.

[40] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3460–3468. AAAI Press, 2015.

[41] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[42] Ahmed El-Roby, Khaled Ammar, Ashraf Aboulnaga, and Jimmy Lin. Sapphire: Querying RDF data made simple. *PVLDB*, 9(13):1481–1484, 2016.

[43] Shady Elbassuoni, Maya Ramanath, and Gerhard Weikum. Query relaxation for entity-relationship search. In Grigoris Antoniou, Marko Grobelnik, Elena Paslaru Bontas Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Z. Pan, editors, *The Semanic Web: Research and Applications - 8th Extended Semantic Web Conference, ESWC 2011, Heraklion, Crete, Greece, May 29 - June 2, 2011, Proceedings, Part II*, volume 6644 of *Lecture Notes in Computer Science*, pages 62–76. Springer, 2011.

[44] Herbert B. Enderton. *A mathematical introduction to logic.* Academic Press, 1972.

[45] Lee Feigenbaum, Ivan Herman, Tonya Hongsermeier, Eric Neumann, and Susie Stephens. The semantic web in action. *Scientific American*, 297:90–97, 2007.

[46] Flavio Antonio Ferrarotti, Alejandra Lorena Paoletti, and José M. Turull Torres. First-order types and redundant relations in relational databases. In Carlos A. Heuser and Günther Pernul, editors, *Advances in Conceptual Modeling - Challenging Perspectives, ER 2009 Workshops CoMoL, ETheCoM, FP-UML, MOST-ONISW, QoIS, RIGiM, SeCoGIS, Gramado, Brazil, November 9-12, 2009. Proceedings*, volume 5833 of *Lecture Notes in Computer Science*, pages 65–74. Springer, 2009.

[47] George H. L. Fletcher, Marc Gyssens, Jan Paredaens, and Dirk Van Gucht. On the expressive power of the relational algebra on finite sets of relation pairs. *IEEE Trans. Knowl. Data Eng.*, 21(6):939–942, 2009.

[48] George H. L. Fletcher, Marc Gyssens, Jan Paredaens, Dirk Van Gucht, and Yuqing Wu. Structural characterizations of the navigational expressiveness of relation algebras on a tree. *J. Comput. Syst. Sci.*, 82(2):229–259, 2016.

[49] Géraud Fokou, Stéphane Jean, Allel HadjAli, and Mickaël Baron. RDF query relaxation strategies based on failure causes. In Harald Sack, Eva Blomqvist, Mathieu d'Aquin, Chiara Ghidini, Simone Paolo Ponzetto, and Christoph Lange, editors, *The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings*, volume 9678 of *Lecture Notes in Computer Science*, pages 439–454. Springer, 2016.

[50] Wikimedia Foundation. Wikidata. `https://www.wikidata.org/`, 2012.

[51] 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.

[52] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[53] Georg Gottlob and Pierre Senellart. Schema mapping discovery from data instances. *J. ACM*, 57(2):6:1–6:37, 2010.

[54] Martin Grohe. From polynomial time queries to graph structure theory. *Commun. ACM*, 54(6):104–112, 2011.

[55] Martin Grohe. Fixed-point definability and polynomial time on graphs with excluded minors. *J. ACM*, 59(5):27:1–27:64, 2012.

[56] RDF Working Group. RDF 1.1 Concepts and Abstract Syntax. https://www.w3.org/TR/rdf11-concepts/.

[57] Dirk Van Gucht. On the expressive power of the extended relational algebra for the unnormalized relational model. In Moshe Y. Vardi, editor, *Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 23-25, 1987, San Diego, California, USA*, pages 302–312. ACM, 1987.

[58] Dirk Van Gucht. Bp-completeness. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 265–266. Springer US, 2009.

[59] Marc Gyssens, Jan Paredaens, and Dirk Van Gucht. A uniform approach toward handling atomic and structured information in the nested relational database model. *J. ACM*, 36(4):790–825, 1989.

[60] Marc Gyssens, Jan Paredaens, Dirk Van Gucht, and George H. L. Fletcher. Structural characterizations of the semantics of xpath as navigation tool on a document. In Stijn Vansummeren, editor, *Proceedings of the Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 26-28, 2006, Chicago, Illinois, USA*, pages 318–327. ACM, 2006.

[61] Lane A. Hemaspaandra. Lowness: A yardstick for np-p. *SIGACT News*, 24(2):10–14, April 1993.

[62] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard H. Hovy, and Eric P. Xing. Harnessing deep neural networks with logic rules. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016.

[63] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. Making database systems usable. In Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 13–24. ACM, 2007.

[64] Peter Jeavons, David A. Cohen, and Marc Gyssens. How to determine the expressive power of constraints. *Constraints*, 4(2):113–131, 1999.

[65] Xueyan Jiang, Volker Tresp, Yi Huang, and Maximilian Nickel. Link prediction in multi-relational graphs using additive models. In Marco de Gemmis, Tommaso Di Noia, Pasquale Lops, Thomas Lukasiewicz, and Giovanni Semeraro, editors, *Proceedings of the International Workshop on Semantic Technologies meet Recommender Systems & Big Data, Boston, USA, November 11, 2012*, volume 919 of *CEUR Workshop Proceedings*, pages 1–12. CEUR-WS.org, 2012.

[66] Minsuk Kahng, Shamkant B. Navathe, John T. Stasko, and Duen Horng (Polo) Chau. Interactive browsing and navigation in relational databases. *PVLDB*, 9(12):1017–1028, 2016.

[67] Esther Kaufmann and Abraham Bernstein. How useful are natural language interfaces to the semantic web for casual end-users? In Aberer et al. [2], pages 281–294.

[68] Johannes Köbler, Uwe Schöning, and Jacobo Toran. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser, 2012.

[69] 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.

[70] Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. Static analysis and optimization of semantic web queries. *ACM Trans. Database Syst.*, 38(4):25:1–25:45, 2013.

[71] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. 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 2181–2187. AAAI Press, 2015.

[72] Anna Lubiw. Some np-complete problems similar to graph isomorphism. *SIAM J. Comput.*, 10(1):11–21, 1981.

[73] Frederick Maier. A primer on RDF and OWL. In Pascal Hitzler, Aldo Gangemi, Krzysztof Janowicz, Adila Krisnadhi, and Valentina Presutti, editors, *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*, volume 25 of *Studies on the Semantic Web*, pages 337–361. IOS Press, 2016.

[74] Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior, and Adrian Bielefeldt. Getting the most out of wikidata: Semantic technology usage in wikipedia's knowledge graph. In Denny Vrandecic, Kalina Bontcheva, Mari Carmen Suárez-Figueroa, Valentina Presutti, Irene Celino, Marta Sabou, Lucie-Aimée Kaffee, and Elena Simperl, editors, *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8-12, 2018, Proceedings, Part II*, volume 11137 of *Lecture Notes in Computer Science*, pages 376–394. Springer, 2018.

[75] John P. McCrae. Linked Open Data Cloud. `https://lod-cloud.net/`, 2018.

[76] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014.

[77] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[78] Jacob Morgan. A simple explanation of 'the internet of things'. `forbes.com/sites/jacobmorgan/2014/05/13/simple-explanation-internet-things-that-anyone-can-understand/`, 2014.

[79] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[80] Maximilian Nickel, Lorenzo Rosasco, and Tomaso A. Poggio. Holographic embeddings of knowledge graphs. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, pages 1955–1961. AAAI Press, 2016.

[81] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 809–816. Omnipress, 2011.

[82] Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of facets (and some facets of complexity). *J. Comput. Syst. Sci.*, 28(2):244–259, 1984.

[83] Jan Paredaens. On the expressive power of the relational algebra. *Inf. Process. Lett.*, 7(2):107–111, 1978.

[84] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, 2009.

[85] François Picalausa and Stijn Vansummeren. What are real SPARQL queries like? In Roberto De Virgilio, Fausto Giunchiglia, and Letizia Tanca, editors, *Proceedings of the International Workshop on Semantic Web Information Management, SWIM 2011, Athens, Greece, June 12, 2011*, page 7. ACM, 2011.

[86] Reinhard Pichler and Sebastian Skritek. Containment and equivalence of well-designed SPARQL. In Richard Hull and Martin Grohe, editors, *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 39–50. ACM, 2014.

[87] Adolfo Piperno. Search space contraction in canonical labeling of graphs (preliminary version). *CoRR*, abs/0804.4881, 2008.

[88] Li Qian, Michael J. Cafarella, and H. V. Jagadish. Sample-driven schema mapping. In K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 73–84. ACM, 2012.

[89] Ronald C. Read and Derek G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1(4):339–363, 1977.

[90] Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R. Mittal, and Fatma Özcan. ATHENA: an ontology-driven system for natural language querying over relational data stores. *PVLDB*, 9(12):1209–1220, 2016.

[91] Muhammad Saleem, Muhammad Intizar Ali, Aidan Hogan, Qaiser Mehmood, and Axel-Cyrille Ngonga Ngomo. LSQ: the linked SPARQL queries dataset. 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 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, volume 9367 of *Lecture Notes in Computer Science*, pages 261–269. Springer, 2015.

[92] Anish Das Sarma, Aditya G. Parameswaran, Hector Garcia-Molina, and Jennifer Widom. Synthesizing view definitions from data. In Segoufin [96], pages 89–103.

[93] Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of SPARQL query optimization. In Segoufin [96], pages 4–33.

[94] Uwe Schöning. A low and a high hierarchy within NP. *J. Comput. Syst. Sci.*, 27(1):14–28, 1983.

[95] Uwe Schöning. Graph isomorphism is in the low hierarchy. *J. Comput. Syst. Sci.*, 37(3):312–323, 1988.

[96] Luc Segoufin, editor. *Database Theory - ICDT 2010, 13th International Conference, Lausanne, Switzerland, March 23-25, 2010, Proceedings*, ACM International Conference Proceeding Series. ACM, 2010.

[97] Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. Reasoning with neural tensor networks for knowledge base completion. In Burges et al. [32], pages 926–934.

[98] Lubomir Stanchev. Creating a probabilistic graph for wordnet using markov logic network. In Rajendra Akerkar, Michel Plantié, Sylvie Ranwez, Sébastien Harispe, Anne Laurent, Patrice Bellot, Jacky Montmain, and François Trousset, editors, *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics, WIMS 2016, Nîmes, France, June 13-15, 2016*, pages 7:1–7:12. ACM, 2016.

[99] Slawek Staworko and Piotr Wieczorek. Learning twig and path queries. In Alin Deutsch, editor, *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, pages 140–154. ACM, 2012.

[100] Slawek Staworko and Piotr Wieczorek. Characterizing XML twig queries with examples. In Arenas and Ugarte [15], pages 144–160.

[101] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

[102] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A large ontology from wikipedia and wordnet. *J. Web Sem.*, 6(3):203–217, 2008.

[103] Wang-Chiew Tan, Giovanna Guerrini, Barbara Catania, and Anastasios Gounaris, editors. *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18-22, 2013*. ACM, 2013.

[104] Wei Chit Tan, Meihui Zhang, Hazem Elmeleegy, and Divesh Srivastava. Reverse engineering aggregation queries. *PVLDB*, 10(11):1394–1405, 2017.

[105] Balder ten Cate and Víctor Dalmau. The product homomorphism problem and applications. In Arenas and Ugarte [15], pages 161–176.

[106] Balder ten Cate, Víctor Dalmau, and Phokion G. Kolaitis. Learning schema mappings. *ACM Trans. Database Syst.*, 38(4):28:1–28:31, 2013.

[107] Jacobo Torán and Fabian Wagner. The complexity of planar graph isomorphism. *Bulletin of the EATCS*, 97:60–82, 2009.

[108] Ana I. Torre-Bastida, Esther Villar-Rodriguez, Miren Nekane Bilbao, and Javier Del Ser. Intelligent SPARQL endpoints: Optimizing execution performance by automatic query relaxation and queue scheduling. In Jesús Carretero, Javier García Blas, Ryan K. L. Ko, Peter Mueller, and Koji Nakano, editors, *Algorithms and Architectures for Parallel Processing - 16th International Conference, ICA3PP 2016, Granada, Spain, December 14-16, 2016, Proceedings*, volume 10048 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2016.

[109] Quoc Trung Tran, Chee-Yong Chan, and Srinivasan Parthasarathy. Query by output. In Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 535–548. ACM, 2009.

[110] Quoc Trung Tran, Chee Yong Chan, and Srinivasan Parthasarathy. Query reverse engineering. *VLDB J.*, 23(5):721–746, 2014.

[111] Denny Vrandecic. Wikidata: a new platform for collaborative data collection. In Alain Mille, Fabien L. Gandon, Jacques Misselis, Michael Rabinovich, and Steffen Staab, editors, *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, pages 1063–1064. ACM, 2012.

[112] W3C SPARQL Working Group. SPARQL 1.1 Overview W3C Recommendation 21 March 2013. https://www.w3.org/TR/sparql11-overview/.

[113] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 1112–1119. AAAI Press, 2014.

[114] Yaacov Y. Weiss and Sara Cohen. Reverse engineering spj-queries from examples. In Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts, editors, *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 151–166. ACM, 2017.

[115] Ross Willard. Testing expressibility is hard. In David Cohen, editor, *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, volume 6308 of *Lecture Notes in Computer Science*, pages 9–23. Springer, 2010.

[116] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 5498–5507. JMLR.org, 2018.

[117] Peipei Yi, Byron Choi, Sourav S. Bhowmick, and Jianliang Xu. Autog: A visual query autocompletion framework for graph databases. *PVLDB*, 9(13):1505–1508, 2016.

[118] V.N. Zemlyachenko, N.M. Korneenko, and R.I. Tyshkevich. Graph isomorphism problem. *Journal of Soviet Mathematics*, 29(4):1426–1481, 1985.

[119] Meihui Zhang, Hazem Elmeleegy, Cecilia M. Procopiuc, and Divesh Srivastava. Reverse engineering complex join queries. In Kenneth A. Ross, Divesh Srivastava, and Dimitris Papadias, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, pages 809–820. ACM, 2013.

[120] Xiaowang Zhang, Jan Van den Bussche, and François Picalausa. On the satisfiability problem for SPARQL patterns. *J. Artif. Intell. Res.*, 56:403–428, 2016.

[121] Moshé M. Zloof. Query by example. In *American Federation of Information Processing Societies: 1975 National Computer Conference, 19-22 May 1975, Anaheim, CA, USA*, volume 44 of *AFIPS Conference Proceedings*, pages 431–438. AFIPS Press, 1975.

[122] Moshé M. Zloof. Query-by-example: the invocation and definition of tables and forms. In Douglas S. Kerr, editor, *Proceedings of the International Conference on Very Large Data Bases, September 22-24, 1975, Framingham, Massachusetts, USA.*, pages 1–24. ACM, 1975.