SPARQL Rewriting for Query Mediation over Mapped Ontologies

Konstantinos Makris^{*}, Nektarios Gioldasis^{*}, Nikos Bikakis^{**}, Stavros Christodoulakis^{*}

*Laboratory of Distributed Multimedia Information Systems & Applications (MUSIC/TUC), Technical University of Crete, Greece {makris, nektarios, stavros}@cced.tuc.gr

**Knowledge & Database Systems Laboratory, National Technical University of Athens, Greece bikakis@dblab.ntua.gr

Abstract

In the recent years establishing interoperability and supporting data integration has become a major research challenge for the web of data. Uniform information access of heterogeneous sources is of major importance for Semantic Web applications and end users. We describe a methodology for SPARQL query mediation over federated OWL/ RDF knowledge bases. The query mediation exploits mappings between semantically related entities of the global ontology and the local ontologies. A very rich set of mappings, based on Description Logic semantics, is supported. The SPARQL queries that are posed over the global ontology are decomposed, rewritten, and then submitted to the federated sources. The rewritten SPARQL queries are locally evaluated and the results are returned to the mediator. We describe the formal modeling of executable mappings (i.e. mappings that can be used in SPARQL query rewriting), as well as the theoretic and implementation aspects of SPARQL query rewriting. Finally we describe the implementation of a system supporting the mediation process.

Contents

1	Introduction	3
2	Related Work 2.1 Ontology mapping 2.2 SPARQL query rewriting	4 4 5
3	Ontology mapping model3.1Motivating example3.2Abstract syntax and semantics3.3Ontology mapping types3.4Mapping representation	6 6 11 14 16
4	The SPARQL query language4.1Syntax of SPARQL and RDF4.2Semantics of SPARQL graph pattern expressions	16 16 17
5	SPARQL query rewriting overview	20
6	Data Triple Pattern rewriting6.1Rewriting by triple pattern's subject part6.2Rewriting by triple pattern's object part6.3Rewriting by triple pattern's predicate part6.4Combination examples	 22 22 24 30 34
7	Schema Triple Pattern rewriting7.1 Rewriting by triple pattern's subject part7.2 Rewriting by triple pattern's object part	39 43 48
8	Graph pattern rewriting	53
9	Implementation	63
10) Conclusion	64
A	Semantics of property relationshipsA.1 Equivalence/subsumption between properties in OWLA.2 Equivalence/subsumption between properties in our framework	69 69 69
В	Data triple pattern rewriting proofs B.1 Proof of Lemma 6.1 B.2 Proof of Lemma 6.2 B.3 Proof of Lemma 6.3	71 72 73 78

1 Introduction

Data access from distributed autonomous web resources needs to take into account the data semantics at the conceptual level. Assuming that the resources are organized and accessed with the same model and language, a straightforward approach to semantic interoperability is to adhere to a common conceptualization (i.e. a global ontological conceptualization). However, in real-world environments, institutions often do not adhere to common standards. Attempts to find an agreement for a common conceptualization often results in semantically weak minimum consensus schemes (e.g. the Dublin Core [27]) or models with extensive and complex semantics (e.g. the CIDOC/CRM [11]). Moreover, it is not often feasible for cooperating institutions to agree on a certain model or apply an existing standard because they often already have their own proprietary conceptualizations. In this environment, query mediation over mapped ontologies has become a major research challenge since it allows uniform semantic information retrieval and at the same time permits diversification on individual conceptualizations followed by distributed federated information sources.

A mediator architecture is a common approach in information integration systems [43]. Mediated query systems represent a uniform data access solution by providing a single point for querying access to various data sources. A mediator contains a global query processor which is used to send sub-queries to local data sources. The local query results are then combined and returned back to the query processor. Its main benefit is that the query formulation process becomes independent of the mediated data sources requiring from endusers to be aware only of their own conceptualization of the knowledge domain.

In this paper, we describe a mediator based methodology and system for integrating information from federated OWL/RDF knowledge bases. The mediator uses mappings between the OWL [3] ontology of the mediator (global ontology) and the federated site ontologies (local ontologies). SPARQL [33] queries posed over the mediator, are decomposed and rewritten in order to be submitted over the federated sites. The SPARQL queries are locally evaluated and the results are returned to the mediator site. In this paper we focus on the following research issues:

- determination of the different mapping types, which can be used in SPARQL query rewriting
- modeling of the mappings between the global ontology and the local ontologies
- rewriting of the SPARQL queries posed over the global ontology in terms of the local ontologies

Regarding the task of mapping modeling, the focus of this work is on the semantics and syntax of the ontology executable mappings. For identifying and describing such mappings, we define a formal grammar for mapping definition.

Based on these mappings we provide a complete set of graph pattern rewriting functions which are materialized as algorithms in the process of query rewriting for local ontologies. These functions are generic and can be used for SPARQL query rewriting over any overlapping ontology set. We show that the provided functions are semantics preserving, in the sense that each rewriting step that we perform (in order to rewrite the initial query) preserves the mapping semantics.

Contribution. The main contributions of this paper are summarized as follows:

• A model for the expression of mappings between OWL ontologies in the context of SPARQL query rewriting. This mapping model consists of a formal grammar for the mapping definition and a formal specification of the mappings semantics.

- A generic formal methodology for the SPARQL query rewriting process, based on a set of mappings between OWL ontologies.
- A system implementation of the proposed method.

Outline. The rest of the paper is organized as follows: The related work is discussed in Section 2. The mapping model which has been developed in order to express the mappings between the OWL ontologies is described in Section 3. An introduction to the SPARQL query language is presented in Section 4. The SPARQL query rewriting process is described comprehensively in Sections 5, 6, 7 and 8. The implementation of the system that supports the query rewriting is discussed in Section 9. Finally, Section 10 concludes our work.

2 Related Work

In the Semantic Web environment a number of ontology based mediator architectures have been proposed in the literature [42], [28]. In the following sections we present the most relevant research to the issues discussed in this paper.

2.1 Ontology mapping

Ontology mapping is the task of relating the vocabulary of two ontologies by defining a set of correspondences. The correspondences between different entities of the two ontologies are typically expressed using some axioms described in a specific mapping language. The discovery and specification of mappings between two ontologies, is a process which can be achieved in three ways:

- *Manually*: defined by an expert who has a very good understanding of the ontologies to be mapped.
- *Automatically*: using various matching algorithms and techniques which compute similarity measures between different ontology terms.
- *Semi-automatically*: using various matching algorithms and techniques, as well as user feedback.

Many strategies and tools that produce automatically or semi-automatically mappings have been proposed and have their performance analyzed ([17], [23], [39], [9], [18], [12], [30]). Although the automatic or semi-automatic techniques and strategies provide satisfactory results, it is unlikely that the quality of mappings that they produce will be comparable with manually specified mappings. The manual approach for defining mappings is a painful process, although, it can provide declarative and expressive correspondences by exploiting the knowledge of an expert for the two mapped ontologies in many different ways.

In addition to the mapping discovery, the mapping representation is a very important issue for an application that implements a mediation scenario. A set of criteria that should be taken into consideration, in order to decide which language/format should be used for the mapping representation include the following (based on [17]):

- Web compatibility
- Language independence
- Simplicity

- Expressiveness
- Purpose independence
- Executability
- Mediation task

Although, many languages (OWL [3], C-OWL [7], SWRL [20], the Alignment Format [14], MAFRA [24], EDOAL [35], [16], OMWG mapping language [36], etc.) have been proposed for the task of mapping representation, only a few combine the previous criteria. A comparison of some of these languages and formats for mapping specification is available in [17]. An interesting approach for defining ontology mappings using an extension of SPARQL has also been presented in [32]. It is declarative by exploiting CONSTRUCT queries, although, it can be used for the specific task of instance transformation.

In this paper, we do not focus on the discovery of the mappings between two ontologies. We are only interested in the specification and the representation of the kinds of mappings between OWL ontologies which can be exploited by a query mediation system in order to perform SPARQL query rewriting. In our knowledge, only [15], examines the problem of describing such mapping types but not directly, since it describes which mapping types cannot be used in the rewriting process. In contrast, we present in this paper concrete mappings that can be used for SPARQL query rewriting.

2.2 SPARQL query rewriting

Within the Semantic Web community, the process of SPARQL query rewriting is gaining attention in order to perform various tasks such as query optimization, query decomposition, query translation, Description Logic inference, data integration and instance transformation.

SPARQL query optimization focuses on rewriting techniques that minimize the evaluation complexity ([38], [5], [40], [19], [31]). On the other hand, in the field of Description Logic inference, SPARQL query rewriting is basically used for performing reasoning tasks. Two recent approaches [22] and [21] perform SPARQL query rewriting by using inference rules, in order to query effectively OWL/RDF knowledge bases.

SPARQL query decomposition, SPARQL query translation and instance transformation are fundamental tasks in information integration systems. Benslimane et. al [4] proposed recently a system that performs SPARQL query decomposition in order to query distributed heterogeneous information sources. After the decomposition, the resulted SPARQL subqueries are translated into SQL sub-queries but no algorithm or details are provided in the paper. In contrast, Quilitz et. al [34] proposed SPARQL query decomposition, in order to overcome the large overhead in network traffic produced by the SPARQL implementations that load all the RDF graphs mentioned in a query to the local machine.

In the field of SPARQL query translation, two recent approaches [13] and [8] perform complete SPARQL query translation into SQL queries, preserving the SPARQL semantics. Similarly with SPARQL-to-SQL proposed methods, Bikakis et. al [6] present a framework and a system which performs SPARQL query translation into XQuery queries, in order to achieve data integration by querying XML data through SPARQL queries.

Regarding instance transformation, Euzenat et al. [15] proposed the use of SPARQL CONSTRUCT statements. This approach is also followed by the commercial TopBraid Composer provided by TopQuadrant¹. SPARQL CONSTRUCT statements have been used in the past for dynamically generating views over RDF graphs [37].

 $^{^1 \}rm http://composing-the-semantic-web.blogspot.com/2006/09/ontology-mapping-with-sparql-construct.html$

Up to now, limited studies have been made in the field of query rewriting related to posing a SPARQL query over different RDF datasets. Akahani et. al [1] proposed a theoretical perspective of approximate query rewriting for submitting queries to multiple ontologies. In their approach no specific context (e.g. using SPARQL) is defined and no specific algorithms for the query rewriting process are provided.

An approach which comes closer to ours, with some of its parts based on a preliminary description of our work [25], has been presented recently by Correndo et al. [10]. They present a SPARQL query rewriting methodology for achieving RDF data mediation over linked data. Correndo et al. use transformations between RDF structures (i.e. graphs) in order to define the mappings between two ontologies. This choice seems to restrict the mappings expressivity and also the supported query types. Queries containing IRIs inside **FILTER** expressions cannot be handled, while the mapping definition seems to be a painful procedure. In contrast to our proposal, mappings produced by an ontology matching [17] system, need post-processing in order to assist the mapping discovery.

3 Ontology mapping model

In order for SPARQL queries posed over a global ontology to be rewritten in terms of a local ontology, mappings between the global and local ontologies should be specified. In this section we present a model for the expression of mappings between OWL ontologies in the context of SPARQL query rewriting. More specifically, in Section 3.1 we present a motivating example. In Section 3.2 and Section 3.3 we present the supported mapping types used for the query rewriting process, as well as their abstract syntax and semantics. Finally, the mapping representation is discussed in Section 3.4.

3.1 Motivating example

In this section we present a motivating example for elicitating the ontology mapping requirements of the mediator framework. Since our query rewriting methodology is generic, we will be discussing for mappings between a source and a target ontology rather than between global and local ontologies. The mapping types presented in this section have been selected among others because they can be used for the rewriting of a SPARQL query.

In Figure 1, we show the structure of two different ontologies. The source ontology describes a store² that sells various products including books and cd's and the target ontology describes a bookstore³. The rounded corner boxes represent the classes. They are followed by their properties (object and datatype). The rectangle boxes at the bottom of the figure represent individuals. The arrows represent the relationships between these basic OWL constructs.

In order to map a source ontology to a target ontology, various relationship types like equivalence and subsumption can be used. For example, in Figure 2 we present an equivalence relationship between ontology constructs and in Figure 3 and we present a subsumption relationship. The source ontology class Book seems to be equivalent with the target ontology class Textbook, as these two classes seem to describe individuals of the same type. Similarly, the source ontology class Product seems to subsume the target ontology class Textbook, as the class Product seems to describe various types of individuals and not only Textbook individuals.

²Store ontology namespace: src = http://www.ontologies.com/SourceOntology.owl#

 $^{^{3}}$ Bookstore ontology namespace: trg = http://www.ontologies.com/TargetOntology.owl#

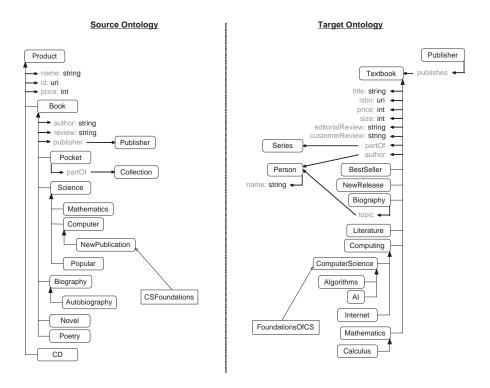


Figure 1: Semantically Overlapping Ontologies.

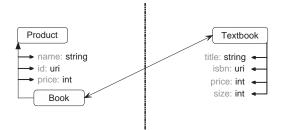


Figure 2: A class mapping using an equivalence relationship.

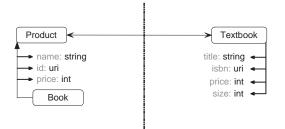


Figure 3: A class mapping using a subsumption relationship.

A source ontology class can be mapped to an expression between target ontology classes. The expression may involve union and intersection operations between classes. For example, in Figure 4 the class Science is mapped to the union of classes ComputerScience and Mathematics, since it seems to describe both ComputerScience and Mathematics individuals. Similarly, in Figure 5 the class Popular is mapped to the intersection of the class BestSeller with the union of classes ComputerScience and Mathematics. This mapping emerges from the fact that the class Popular seems to describe ComputerScience and Mathematics and Mathematics individuals which are also of type BestSeller.

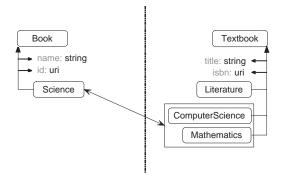


Figure 4: A class mapping using a union operation between classes.

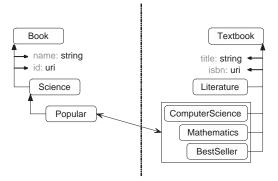


Figure 5: A class mapping using union and intersection operations between classes.

In addition, using expressions it is possible to restrict a class on some property values in order to form a correspondence. For example, in Figure 6 the class Pocket is mapped to the class Textbook restricted on its size property values, since the class Pocket seems to describe Textbook individuals having a specific value for the property size (e.g. less than or equal to 14). Similarly, in Figure 7 the class Autobiography is mapped to the class Biography restricted to the values of the properties author and topic. This mapping emerges from the fact that the class Autobiography seems to describe Biography individuals having the same value for these two properties.

Similarly with classes, an individual from the source ontology can be mapped to an individual from the target ontology (see Figure 8). In this case, only the equivalence relationship can be taken into consideration since the subsumption relationship is used mainly with sets.

Accordingly, an object/datatype property from the source ontology can be mapped to

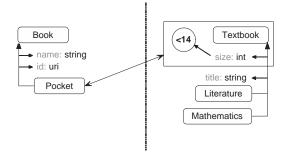


Figure 6: A class mapping using a property restriction.

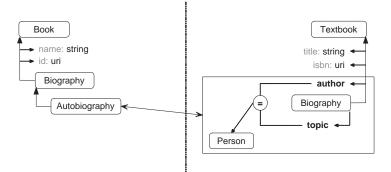


Figure 7: A class mapping using a property restrictions.

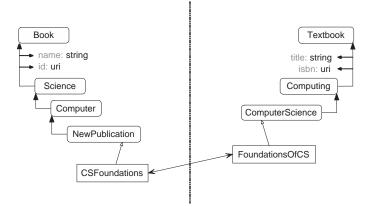


Figure 8: A mapping between two individuals.

an object/datatype property from the target ontology (see Figure 9).

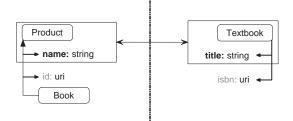


Figure 9: A mapping between two datatype properties.

Domain and range restrictions can be useful for mappings between properties in order to restrict the individuals that participate in these two sets. For example, in Figure 10 the object property partOf from the source ontology is mapped to the object property partOf from the target restricted on its domain values. More specifically, the domain of the property partOf from the target ontology (i.e. class Textbook) is restricted on its size property values in order to match with the domain of the property partOf from the source ontology.

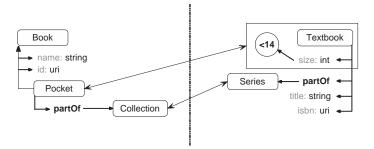


Figure 10: An object property mapping using a domain restriction.

In addition, an object property from the source ontology can be mapped to the inverse of an object property from the target ontology. For example, in Figure 11 the object property **publisher** is mapped to the inverse of the object property **publishes**, since the binary relations described by the property **publisher** correspond with the inverse binary relations described by the property **publishes**. Taking a closer look, we observe that the domain of the property **publisher** corresponds with the range of the property **publishes**, and similarly the range of the property **publisher** corresponds with the domain of the property **publishes**.

Finally, a source ontology property can be mapped to an expression between target ontology properties. The expression may involve union, intersection and composition operations between properties. For example, in Figure 12 the datatype property review is mapped to the union of the datatype properties editorialReview and customerReview, since the binary relations described by the property review correspond with the binary relations described by the properties editorialReview and customerReview.

Similarly, in Figure 13 the datatype property author from the source ontology is mapped to the composition of the object property author with the datatype property name from the target ontology. This mapping emerges from the fact that the binary relations described by the datatype property author from the source ontology correspond with the binary relations

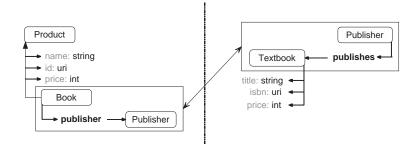


Figure 11: An object property mapping using an inverse operation.

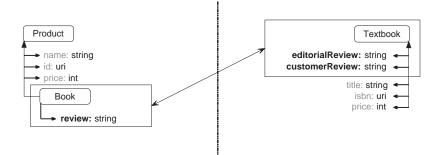


Figure 12: A datatype property mapping using a union operation.

provided by connecting the Textbook individuals to the name property values of the class People.

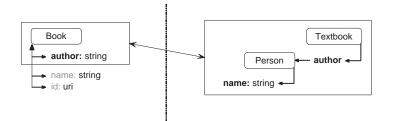


Figure 13: A datatype property mapping using a composition operation.

3.2 Abstract syntax and semantics

The basic concepts of OWL, whose mappings are useful for the rewriting process, are the classes c, the object properties op, the datatype properties dp and the individuals i. Since we are working in the context of SPARQL queries, some mapping types may not be useful for the query rewriting process. For example, a mapping containing aggregates would be meaningless, since aggregates cannot be represented in the current SPARQL. Such mapping types are described in [15] and many of them could be useful for post-processing the query results but not during the query rewriting and query answering process.

In order to define the mapping types which are useful for the rewriting process, we use Description Logics (DL). We treat OWL classes as DL concepts, OWL properties as DL roles and OWL individuals as DL individuals. Following our convension, let C, D be OWL classes (treated as atomic concepts), R, S be OWL object properties (treated as atomic roles) and K, L be OWL datatype properties (treated as atomic roles). Similarly, let a, b, c, v_{op} be individuals and v_{dp} be a data value.

An interpretation \mathcal{I} consists of a non-empty set $\Delta^{\mathcal{I}}$ (the domain of the interpretation) and an interpretation function, which assigns to every atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, to every atomic role B a binary relation $B^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and to every individual k an element $k^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ (based on [2]).

In Table 1, Table 2 and Table 3 we present the set of class and property constructors which we use for the definition of mappings. In these tables we introduce some new constructors (preceded with asterisk) which should not be confused with the basic Description Logics constructors defined in [2]. In addition to the concept/role constructors, a DL knowledge base consists of assertional axioms which are presented in Table 4.

Table 1: Class constructors used in the definition of mappings.

Name	Syntax	Semantics
Intersection Union *Class Restriction	$\begin{array}{l} C \sqcap D \\ C \sqcup D \\ \forall C.(R \ \overline{\operatorname{cp}} \ v_{op}) \\ \forall C.(K \ \operatorname{cp} \ v_{dp}) \\ \forall C.(R \ \overline{\operatorname{cp}} \ S) \\ \forall C.(K \ \operatorname{cp} \ L) \end{array}$	$\begin{array}{l} C^{\mathcal{I}} \cap D^{\mathcal{I}} \\ C^{\mathcal{I}} \cup D^{\mathcal{I}} \\ \{ \alpha \in C^{\mathcal{I}} \mid \exists b. \; (\alpha, b) \in R^{\mathcal{I}} \wedge b \; \overline{\operatorname{cp}} \; v_{op} \} \\ \{ \alpha \in C^{\mathcal{I}} \mid \exists b. \; (\alpha, b) \in K^{\mathcal{I}} \wedge b \; \operatorname{cp} \; v_{dp} \} \\ \{ \alpha \in C^{\mathcal{I}} \mid \exists b, \exists c. \; (\alpha, b) \in R^{\mathcal{I}} \wedge (\alpha, c) \in S^{\mathcal{I}} \wedge b \; \overline{\operatorname{cp}} \; c \} \\ \{ \alpha \in C^{\mathcal{I}} \mid \exists b, \exists c. \; (\alpha, b) \in K^{\mathcal{I}} \wedge (\alpha, c) \in L^{\mathcal{I}} \wedge b \; \operatorname{cp} \; c \} \end{array}$
	$\overline{\mathtt{cp}} \in \{ \neq, = \}, \mathtt{cp}$	$a \in \{\neq,=,\leq,\geq,<,>\}$

Table 2: Object property constructors used in the definition of mappings.

Name	Syntax	Semantics
Intersection	$R\sqcap S$	$R^{\mathcal{I}} \cap S^{\mathcal{I}}$
Union	$R \sqcup S$	$R^{\mathcal{I}} \cup S^{\mathcal{I}}$
Composition	$R \circ S$	$\{(\alpha, c) \mid \exists b. \ (\alpha, b) \in R^{\mathcal{I}} \land (b, c) \in S^{\mathcal{I}}\}$
*Inverse	inv(R)	$\{(b,\alpha) \mid (\alpha,b) \in R^{\mathcal{I}}\}$
*Domain Restriction	$\forall R.domain(C)$	$\{(\alpha, b) \mid (\alpha, b) \in R^{\mathcal{I}} \land \alpha \in C^{\mathcal{I}}\}$
*Range Restriction	$\forall R.range(C)$	$\{(\alpha, b) \mid (\alpha, b) \in R^{\mathcal{I}} \land b \in C^{\mathcal{I}}\}$

Definition 3.1 (Class expression). A class expression is a class or any complex expression between two or more classes, using union or intersection operations. A class expression is denoted as CE and is defined recursively in (1). Any class expression can be restricted to the values of one or more object property expressions OPE (Definition 3.2) or datatype property expressions DPE (Definition 3.3), using the comparators $\overline{cp} \in \{\neq, =\}$ and $cp \in \{\neq, =, \leq, <, >\}$, respectively. Moreover, it is possible for a class expression to be restricted on a set of individuals having property values (either individuals v_{op} or data values v_{dp}) with a specific relationship between them, defined either by \overline{cp} or cp.

Table 3: Datatype property constructors used in the definition of mappings.

Name	Syntax	Semantics
Intersection	$K \sqcap L$	$K^{\mathcal{I}} \cap L^{\mathcal{I}}$
Union	$K \sqcup L$	$K^{\mathcal{I}} \cup L^{\mathcal{I}}$
Composition	$R \circ K$	$\{(\alpha, c) \mid \exists b. \ (\alpha, b) \in R^{\mathcal{I}} \land (b, c) \in K^{\mathcal{I}}\}$
*Domain Restriction	$\forall K.domain(C)$	$\{(\alpha, b) \mid (\alpha, b) \in K^{\mathcal{I}} \land \alpha \in C^{\mathcal{I}}\}$
*Range Restriction	$\forall K.range(cp \ v_{dp})$	$\{(\alpha, b) \mid (\alpha, b) \in K^{\mathcal{I}} \land b \text{ cp } v_{dp}\}$
	· • •	
	$\mathtt{cp} \in \{ \neq, =, \leq, \geq, <$	$\{z, >\}$

Table 4: Terminological and assertional axioms used in the definition of mappings.

Name	Syntax	Semantics
Class inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
	$C \sqsupseteq D$	$C^{\mathcal{I}} \supseteq D^{\mathcal{I}}$
Object property inclusion	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
	$R \sqsupseteq S$	$R^{\mathcal{I}} \supseteq S^{\mathcal{I}}$
Datatype property inclusion	$K \sqsubseteq L$	$K^{\mathcal{I}} \subseteq L^{\mathcal{I}}$
	$K \sqsupseteq L$	$K^{\mathcal{I}} \supseteq L^{\mathcal{I}}$
Class equality	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
Object property equality	$R\equiv S$	$R^{\mathcal{I}} = S^{\mathcal{I}}$
Datatype property equality	$K\equiv L$	$K^{\mathcal{I}} = L^{\mathcal{I}}$
Individual equality	$a \equiv b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$

 $CE := c \mid CE \sqcap CE \mid CE \sqcup CE \mid \forall CE.(OPE \ \overline{cp} \ v_{op}) \mid \forall CE.(DPE \ cp \ v_{dp}) \\ \mid \forall CE.(OPE_1 \ \overline{cp} \ OPE_2) \mid \forall CE.(DPE_1 \ cp \ DPE_2)$ (1)

Definition 3.2 (Object property expression). An object property expression is an object property or any complex expression between two or more object properties, using composition, union or intersection operations. An object property expression is denoted as OPE and is defined recursively in (2). Inverse property operations are possible to appear inside an object property expression. Any object property expression can be restricted on its domain and/or range by using a class expression defining the applied restrictions.

$$OPE := op | OPE \circ OPE | OPE \sqcap OPE \sqcup OPE \sqcup OPE | inv(OPE) | \forall OPE.domain(CE) | \forall OPE.range(CE)$$
(2)

Definition 3.3 (Datatype property expression). A datatype property expression is a datatype property or any complex expression between object and datatype properties using the composition operation, or between two or more datatype properties, using union or intersection operations. A datatype property expression is denoted as DPE and is defined

recursively in (3). Any datatype property expression can be restricted on its domain values by using a class expression defining the applied restrictions. In addition, the range values of a datatype property expression can be restricted on various data values v_{dp} , using a comparator $cp \in \{\neq, =, \leq, \geq, <, >\}$.

$$DPE := dp \mid OPE \circ DPE \mid DPE \sqcap DPE \mid DPE \sqcup DPE \\ \mid \forall DPE.domain(CE) \mid \forall DPE.range(cp \ v_{dp})$$
(3)

3.3 Ontology mapping types

Although, N:M cardinality mappings can be identified between two ontologies, many problems arise in the exploitation of such mapping types in SPARQL query rewriting. The main problem is the identification of the source ontology's mapped expression inside a SPARQL query, which needs special treatment in order to be overcomed.

In this section we present a rich set of 1:N cardinality mapping types, in order for these mapping types to be used for the rewriting of a SPARQL query. Since our query rewriting methodology is generic, we will be discussing for mappings between a source and a target ontology rather than between global and local ontologies.

Class mapping. A class from a source ontology s can be mapped to a class expression from a target ontology t (refer to (4)).

$$c_s \ rel \ CE_t, \ rel := \equiv | \sqsubseteq | \supseteq$$

$$\tag{4}$$

Object property mapping. An object property from a source ontology s can be mapped to an object property expression from a target ontology t (refer to (5)).

$$op_s \ rel \ OPE_t, \ rel := \equiv | \sqsubseteq | \supseteq$$

$$(5)$$

Datatype property mapping. A datatype property from a source ontology s can be mapped to a datatype property expression from a target ontology t (refer to (6)).

$$dp_s \ rel \ DPE_t, \ rel := \equiv | \sqsubseteq | \sqsupseteq$$

$$(6)$$

We note here that the equivalence between two different properties or between a property and a property expression, denotes equivalence between the domains and ranges of those properties or property expressions. Similarly, the subsumption relationships between two different properties or between a property and a property expression denote analogous relationships between the domains and ranges of those properties or property expressions. The proofs for the above statements are available in the Appendix B.

Individual mapping. An individual from a source ontology s can be mapped to an individual from a target ontology t (refer to (7)).

$$i_s \equiv i_t$$
 (7)

Table 5: Class mappings based in Figure 1.

Class Mappings		
a.	$src: Book \equiv trg: Textbook$	
b.	$src: Product \supseteq trg: Textbook$	
c.	$src: Publisher \equiv trg: Publisher$	
d.	$src: Collection \sqsubseteq trg: Series$	
e.	$src: Novel \sqsubseteq trg: Literature$	
f.	$src: Poetry \sqsubseteq trg: Literature$	
g.	$src: Biography \equiv trg: Biography$	
h.	$src: Autobiography \equiv \forall trg: Biography.(trg: author = trg: topic)$	
i.	$src: NewPublication \equiv trg: Computing \sqcap trg: NewRelease$	
j.	$src: Science \equiv trg: ComputerScience \sqcup trg: Mathematics$	
k.	$src: Popular \equiv (trg: ComputerScience \sqcup trg: Mathematics) \sqcap$	
	$\sqcap trg: BestSeller$	
1.	$src: Pocket \equiv \forall trg: Textbook.(trg: size \leq 14)$	

Table 6: Object property mappings based in Figure 1.

Object Property Mappings		
m.	$src: publisher \equiv inv(trg: publishes)$	
n.	$src: partOf \equiv \forall trg: partOf.domain(\forall trg: Textbook.(trg: size \leq 14))$	

Table 7: Datatype property mappings based in Figure 1.

Dat	tatype Property Mappings
0.	$src:name \supseteq trg:title$
р.	$src: id \supseteq trg: isbn$
q.	$src: price \equiv trg: price$
r.	$src: review \equiv trg: editorial Review \sqcup trg: customer Review$
s.	$src: author \equiv trg: author \circ trg: name$

Table 8: Individual mappings based in Figure 1.

Table 8: Individual mappings based in Figure 1.	
Individual Mappings	_
t. $src: CSF oundations \equiv trg: FoundationsOfCS$	_

3.4 Mapping representation

In the previous sections we presented the abstract syntax used for the mapping definition. Using this abstract syntax we list, in Table 5, Table 6, Table 7 and Table 8, a possible set of correspondences for the ontologies presented in Figure 1.

In order to implement our framework, the need of a serializable language is of major importance. As mentioned in Section 2.1 many languages have been proposed for the task of mapping representation (C-OWL [7], SWRL [20], the Alignment Format [14], MAFRA [24], etc.). Although, the language that fulfills the majority of our requirements is EDOAL⁴ (Expressive and Declarative Ontology Alignment Language). Previous versions of this language have been defined in [16] and [35].

EDOAL combines the Alignment Format [14], which is used in order to represent the output of ontology matching algorithms, and the OMWG mapping language [36], which is an expressive ontology alignment language. The expressiveness, the simplicity, the Semantic Web compliance (given its RDF syntax) and the capability of using any kind of ontology language are the key features of this language.

4 The SPARQL query language

SPARQL [33] is the standard language for querying RDF [26] data. The evaluation of a SPARQL query is based on graph pattern matching. The "Where" clause of a SPARQL query consists of a graph pattern (see Definition 4.4), which is defined recursively and contains triple patterns and SPARQL operators. The operators of the SPARQL algebra which can be applied on graph patterns are: AND, UNION, OPT and FILTER. Triple patterns (see Definition 4.3) are just like RDF triples (see Definition 4.1) except that each of the subject, predicate and object parts may be a variable.

SPARQL allows four query forms: *Select*, *Ask*, *Construct* and *Describe*. In addition, SPARQL provides various solution sequence modifiers which can be applied on the initial solution sequence. The supported SPARQL solution sequence modifiers are: *Distinct*, *OrderBy*, *Reduced*, *Limit*, and *Offset*.

In Section 4.1, we provide a set of definitions regarding the syntax of SPARQL and RDF. Afterwards, in Section 4.2 we present the semantics of SPARQL graph pattern expressions based on [31].

4.1 Syntax of SPARQL and RDF

Let I be the set of IRIs, L be the set of the RDF Literals, and B be the set of the blank nodes. Assume additionally the existence of an infinite set V of variables disjoint from the previous sets (I, B, L).

Definition 4.1 (RDF Triple). A triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an RDF triple, where s, p, and o are a subject, predicate, and object, respectively.

Definition 4.2 (RDF Graph). An RDF graph G is a set of RDF triples.

⁴http://alignapi.gforge.inria.fr/edoal.html

Definition 4.3 (Triple pattern). A triple $(s, p, o) \in (I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is called a triple pattern.

Definition 4.4 (Graph pattern). A SPARQL graph pattern expression is defined recursively as follows:

- A triple pattern is a graph pattern.
- If P_1 and P_2 are graph patterns, then expressions $(P_1 AND P_2)$, $(P_1 OPT P_2)$, and $(P_1 UNION P_2)$ are graph patterns (conjunction graph pattern, optional graph pattern, and union graph pattern, respectively).
- If P is a graph pattern and R is a SPARQL built-in condition, then the expression (P FILTER R) is a graph pattern (a filter graph pattern).

We note that a SPARQL built-in condition is constructed using IRIs, RDF literals, variables and constants, as well as logical connectives, operators (&&, $\parallel, !, =, ! =, >, <, \ge, \le, +, -, *, /)$ and built-in functions (e.g. bound, isIRI, isLiteral, datatype, lang, str, regex).

Definition 4.5 (Basic graph pattern). A finite sequence of conjunctive triple patterns and possible filters is called basic graph pattern.

4.2 Semantics of SPARQL graph pattern expressions

In this section we provide an overview of the semantics of SPARQL graph pattern expressions defined in [31], considering a function-based representation of a graph pattern evaluation over an RDF dataset.

In order not to confuse, the notation and the terminology followed in this section is differentiated in some cases, compared to the notation and terminology followed in [31]. In Table 9 we provide the notation which is used for defining the semantics of SPARQL graph pattern expressions.

Definition 4.6 (SPARQL graph pattern solution). A graph pattern solution $\omega : V \rightarrow (I \cup B \cup L)$ is a partial function that assigns RDF terms of an RDF dataset to variables of a SPARQL graph pattern. The domain of ω , dom(ω), is the subset of V where ω is defined. The empty graph pattern solution ω_{\emptyset} is the graph pattern solution with empty domain. The SPARQL graph pattern evaluation result is a set Ω of graph pattern solutions ω .

Two graph pattern solutions ω_1 and ω_2 are compatible when for all $x \in dom(\omega_1) \cap dom(\omega_2)$, it is the case that $\omega_1(x) = \omega_2(x)$. Furthermore, two graph pattern solutions with disjoint domains are always compatible, and the empty graph pattern solution ω_{\emptyset} is compatible with any other graph pattern solution.

Let Ω_1 and Ω_2 be sets of graph pattern solutions and \mathcal{J} be a set of SPARQL variables. The join, union, difference, projection and left outer join operations between Ω_1 and Ω_2 are defined as follows:

Notation	Description
V	The set of variables.
Ι	The set of IRIs.
В	The set of blank nodes.
L	The set of RDF Literals.
ω	A graph pattern solution $\omega: V \to (I \cup B \cup L)$.
$dom(\omega)$	Domain of a graph pattern solution ω (subset
var(t)	of V). The variables of a triple pattern t .
$\omega(t)$	The triple obtained by replacing the variables in triple pattern t according to a graph pattern solution ω (abusing notation).
$\omega \models R$	A graph pattern solution ω satisfies a built-in condition R .
[[·]]	Graph pattern evaluation function.
\bowtie	Graph pattern solution-based join.
M	Graph pattern solution-based left outer join.
	Graph pattern solution-based difference.
$\pi_{\{\dots\}}$	Graph pattern solution-based projection.
U	Graph pattern solution-based union.
\cap	Set intersection.
?x,?y	SPARQL variables.
bound	SPARQL unary predicate.
AND, OPT, UNION, FILTER	SPARQL graph pattern operators.
\neg, \lor, \land	Logical not, or, and.
$=,\leq,\geq,<,>$	Inequality/equality operators.

Table 9: The notation which is used for defining the semantics of SPARQL graph pattern expressions.

$$\begin{split} \Omega_1 &\bowtie \Omega_2 = \{ \omega_1 \cup \omega_2 \mid \omega_1 \in \Omega_1, \ \omega_2 \in \Omega_2 \text{ are compatible } graph \ pattern \ solutions \}, \\ \Omega_1 \cup \Omega_2 &= \{ \omega \mid \omega \in \Omega_1 \text{ or } \omega \in \Omega_2 \}, \\ \Omega_1 \setminus \Omega_2 &= \{ \omega \in \Omega_1 \mid \text{ for all } \omega' \in \Omega_2, \ \omega \text{ and } \omega' \text{ are not compatible} \}, \\ \pi_{\mathcal{J}}(\Omega_1) &= \{ \omega \mid \omega' \in \Omega_1, \ dom(\omega) = dom(\omega') \cap \mathcal{J} \text{ and } \forall x \in dom(\omega), \ \omega(x) = \omega'(x) \}, \\ \Omega_1 \Join \Omega_2 &= (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2) \end{split}$$

The semantics of SPARQL graph pattern expressions is defined as a function $[[\cdot]]_D$ which takes a graph pattern expression and an RDF dataset D and returns a set of graph pattern solutions (see Definition 4.8). Refer to Definition 4.7 for the semantics of FILTER expressions, which can be part of a SPARQL graph pattern.

Definition 4.7 (SPARQL FILTER expression evaluation). Given a graph pattern solution ω and a built-in condition R, we say that ω satisfies R, denoted by $\omega \models R$, if:

- 1. R is bound(?x) and $?x \in dom(\omega)$;
- 2. R is $?x \text{ oprt } c, ?x \in dom(\omega) \text{ and } \omega(?x) \text{ oprt } c$, where $oprt \rightarrow = | \leq | \geq | < | >$;
- 3. R is ?x oprt ?y, ?x $\in dom(\omega)$, ?y $\in dom(\omega)$ and ω (?x) oprt ω (?y), where oprt $\rightarrow = |$ $\leq | \geq | < | >$;
- 4. R is $(\neg R_1)$, R_1 is a built-in condition, and it is not the case that $\omega \models R_1$;
- 5. R is $(R_1 \vee R_2), R_1$ and R_2 are built-in conditions, and $\omega \models R_1$ or $\omega \models R_2$;
- 6. R is $(R_1 \wedge R_2)$, R_1 and R_2 are built-in conditions, $\omega \models R_1$ and $\omega \models R_2$.

Definition 4.8 (SPARQL graph pattern evaluation). Let D be an RDF dataset over $(I \cup B \cup L)$, t a triple pattern, P, P_1 , P_2 graph patterns and R a built-in condition. The evaluation of a graph pattern over D, denoted by $[[\cdot]]_D$, is defined recursively as follows:

- 1. $[[t]]_D = \{ \omega \mid dom(\omega) = var(t) \text{ and } \omega(t) \in D \}$
- 2. $[[(P_1 \text{ AND } P_2)]]_D = [[P_1]]_D \bowtie [[P_2]]_D$
- 3. $[[(P_1 \text{ OPT } P_2)]]_D = [[P_1]]_D \bowtie [[P_2]]_D$
- 4. $[[(P_1 \text{ UNION } P_2)]]_D = [[P_1]]_D \cup [[P_2]]_D$
- 5. $[[(P \text{ FILTER } R)]]_D = \{\omega \in [[P]]_D \mid \omega \models R\}$

For a detailed description of SPARQL semantics and for a complete set of illustrative examples, refer to [31].

5 SPARQL query rewriting overview

In this section we present an overview of the SPARQL query rewriting process. Query rewriting is done by using a predefined set of mappings which is based on the different mapping types described in Section 3.3.

The SPARQL query rewriting process uses only the query's graph pattern rewriting. The rewritten query is produced by replacing the rewritten graph pattern to the initial query's graph pattern. Consequently the rewriting process is independent of the query type (i.e. SELECT, CONSTRUCT, ASK, DESCRIBE) and the SPARQL solution sequence modifiers (i.e. ORDER BY, DISTINCT, REDUCED, LIMIT, OFFSET). Except from the above, the SPARQL graph pattern operators (i.e. AND, UNION, OPTIONAL, FILTER) which may appear inside the initial query's graph pattern do not result in modifications, since they do not affect the rewriting procedure.

Since a graph pattern consists basically of triple patterns, the most important part of a SPARQL query rewriting is the query's triple pattern rewriting. SPARQL triple patterns can refer either to data or schema information. In order to present the rewriting procedure in depth and due to the difficulty in handling all the different triple pattern types in the same manner, we distinguish triple patterns into Data Triple Patterns (see Definition 5.1) and Schema Triple Patterns (see Definition 5.2).

Let L be the set of literals, V the set of variables, I the set of IRIs, I_{RDF} the set containing the IRIs of the RDF vocabulary (e.g. rdf : type), I_{RDFS} the set containing the IRIs of the RDF Schema vocabulary (e.g. rdfs : subClassOf) and I_{OWL} the set containing the IRIs of the OWL vocabulary (e.g. owl : equivalentClass).

Definition 5.1 (Data Triple Pattern). The triple patterns that only apply to data and not schema info are considered to be Data Triple Patterns. A tuple $t \in DTP$ (Data Triple Pattern set - see (8)) is a Data Triple Pattern.

$$DTP = (I' \cup L \cup V) \times (I' \cup \{rdf : type, owl : sameAs\}) \times (I' \cup L \cup V)$$
(8)

$$I' = I - I_{RDF} - I_{RDFS} - I_{OWL} \tag{9}$$

Definition 5.2 (Schema Triple Pattern). The triple patterns that only apply to schema and not data info are considered to be Schema Triple Patterns. A tuple $t \in STP$ (Schema Triple Pattern set - see (10)) is a Schema Triple Pattern.

$$STP = ((I \cup L \cup V) \times I \times (I \cup L \cup V)) - DTP$$
⁽¹⁰⁾

The factor which is mainly used for the categorization of a triple pattern is the triple pattern's predicate part. The only exception occurs when the predicate part of a triple pattern contains the RDF property rdf: type. In this case, the object part of the triple pattern should be checked. If a triple pattern's object part contains an RDF/RDFS/OWL IRI, then the triple pattern concerns schema info. In Table 10 we present the categorization of a triple pattern set, into Data/Schema Triple Patterns. Triple patterns having a variable on their predicate part are not taken into consideration, since they can deal either with data or with schema info.

Table 10: Triple pattern categorization example, based on the ontologies presented in Figure 1.

Category	Triple Pattern (s, p, o)
Data Triple Patterns	(?x, rdf: type, src: Product)
	(?x, src: author, ?y) $(?x, src: price, "12"^xsd: int)$
Schema Triple Patterns	(?x, rdfs : subClassOf, src : Product) (src : author, rdfs : domain, ?x)
Non-categorized (non-supported)	(src: Pocket, owl: equivalentClass, ?x) $(src: CSFoundations, ?x, "52"^xsd: int)$ (src: Popular, ?x, src: Science)

Since a triple pattern consists of three parts (subject, predicate, object), in order to rewrite it using the predefined mappings for each of the subject, predicate and object parts, we have to follow a three-step procedure. Firstly, a triple pattern is rewritten using the mapping which has been defined for its predicate part. Afterwards, the resulted graph pattern is rewritten triple pattern by triple pattern, using the mappings of the triple patterns' object parts. Finally, the same procedure is repeated for the triple patterns' subject parts. It is worth to mention that SPARQL variables, literal constants and RDF/RDFS/OWL IRIs which may appear in the subject, predicate or object of a triple pattern remain the same during the rewriting procedure. Consequently, the SPARQL variables of the initial query appear also in the rewritten query.

In Section 6 and Section 7, we provide a set of functions which perform triple pattern rewriting using predefined mappings for a triple pattern's subject, predicate and object parts, after taking into consideration the triple pattern's type. In Table 11, we present the notation used for the definition of these functions.

 Table 11: The notation used for the definition of the different triple pattern rewriting functions.

Symbol	Notation
x_s	The indicator s denotes that the entity x (class, object property,
	datatype property or individual) belongs to the source ontology.
x_t	The indicator t denotes that the entity x (class, object property,
	datatype property or individual) belongs to the target ontology.
$\mathcal{D}_{y}^{x}(t,\mu)$	The \mathcal{D} function produces the resulted form of a Data Triple Pat-
0	tern t , after being rewritten by x (subject or predicate or object)
	using a mapping μ . The exponent x gets values from the set
	$\{s, p, o\}$, in order to demonstrate the part of t which is used by
	the rewriting process. The indicator y gets values from the set
	$\{c, op, dp, i, *\}$, in order to show the type of x (e.g. class, object
	property, etc.). The asterisk is used to denote any type (class,
	object property, datatype property or individual).
$\mathcal{S}_{u}^{x}(t,\mu)$	The \mathcal{S} function produces the resulted form of a Schema Triple
<u>.</u>	Pattern t after being rewritten. The exponent x and the indicator
	y are defined above.

In addition to the triple patterns, a graph pattern may contain filters. The SPARQL variables, literal constants, operators (&&, \parallel , !, =, ! =, >, <, >=, <=, +, -, *, /) and built-in functions (e.g. bound, isIRI, isLiteral, datatype, lang, str, regex) which may appear inside a FILTER expression remain the same during the rewriting process. For class IRIs and property IRIs which may appear inside a FILTER expression of a SPARQL query, we use 1:1 cardinality mappings for the expression rewriting. This poses a minor limitation, considering that an IRI can appear inside a FILTER expression only for equality and inequality operations. Thus, the rewriting of a FILTER expression is performed by substituting any IRIs that refer to a class, property, or individual, according to the specified mappings.

We note that the rewriting of a triple pattern, is not dependent on mapping relationships (i.e. equivalence, subsumption). These relationships, affect only the evaluation results of the rewritten query after being posed over the target ontology. A complete algorithm and a set of examples, which show the rewriting process of a graph pattern, based on a set of predefined mappings are presented in Section 8.

6 Data Triple Pattern rewriting

In this section, we provide the functions which perform Data Triple Pattern rewriting based on a set of mappings. These functions are actually rewriting steps in the process of Data Triple Pattern rewriting and are also semantics preserving (see Definition 6.1).

Let DS_s and DS_t be the RDF datasets of a source and a target ontology respectively. Similarly, let DS_m be the RDF dataset which is produced by merging [29] the DS_s and DS_t datasets using a set of mappings \mathcal{M} .

Definition 6.1 (Semantics preserving rewriting). Given a complete set (i.e. a set that contains every possible mapping) of sound (i.e. valid) mappings \mathcal{M} between DS_s and DS_t , the rewriting step performed for a triple pattern t, based on a mapping $\mu \in \mathcal{M}$, is semantics preserving if and only if the evaluation result of t and the evaluation result of the rewritten graph pattern gp' over DS_m , preserve the mapping semantics.

In other words, having a set $\mathcal{J} = var(t)$ of SPARQL variables, the relationship $(\equiv, \sqsubseteq, \sqsupseteq)$ that holds for the mappings used in the rewriting process, should also hold between $[[t]]_{DS_m}$ and $[[gp']]_{DS_m}$ projected on \mathcal{J} .

$$[[t]]_{DS_m} \ rel \ \pi_{\mathcal{J}} \left([[gp']]_{DS_m} \right), \ rel := \equiv |\sqsubseteq| \ \supseteq$$

$$(11)$$

$$\mathcal{J} = var(t) \cap var(gp') = var(t) \tag{12}$$

In Section 6.1 we describe the Data Triple Pattern rewriting process using a mapping for the triple pattern's subject part, while in Sections 6.2 and 6.3 we present the Data Triple Pattern rewriting process using mappings for the triple pattern's object and predicate parts respectively.

6.1 Rewriting by triple pattern's subject part

Generally, when a class or a property appears on the subject part of a triple pattern we conclude that the triple pattern involves schema info, as there is no way for a non RDF/ RDFS/OWL IRI to appear at the same time in the triple pattern's predicate part. Thus, the only case mentioned for the rewriting of a Data Triple Pattern by its subject part concerns individuals appearing in the triple pattern's subject part.

Rewriting based on individual mapping. Let i_s be an individual from the source ontology which is mapped to an individual i_t from the target ontology. Having a Data Triple Pattern $t = (i_s, predicate, object)$ with i_s in its subject part and anything in its predicate and object parts, we can rewrite it by its subject part, using a predefined mapping μ and the function (13).

$$\mathcal{D}_i^s(t,\mu) = (i_t, predicate, object) \quad \text{if } \mu: \ i_s \equiv i_t \tag{13}$$

Example 6.1. Consider the query posed over the source ontology of Figure 1: "Return the type of the CSFoundations individual". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
SELECT ?x
WHERE
{
    src:CSFoundations rdf:type ?x.
```

}

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (src: CSFoundations, rdf: type, ?x) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's subject part, the result of the triple pattern's t rewriting by its subject part is provided by invoking the function (13).

t = (src: CSF oundations, rdf: type, ?x) $\mu: src: CSF oundations \equiv trg: FoundationsOfCS$

Using the parameters defined above, as well as the function (13), the triple pattern t is rewritten as follows:

 $\mathcal{D}_{i}^{s}(t,\mu) = (trg:FoundationsOfCS, rdf:type, ?x)$

In Lemma 6.1 we summarize the functions presented in this section, which are used for the rewriting of a Data Triple Pattern based on a mapping for the triple pattern's subject part.

Lemma 6.1. Having a Data Triple Pattern t and a predefined mapping μ for its subject part, we can rewrite it by its subject, by invoking the function (14). Considering the semantics of the initial triple pattern, as well as the semantics of the resulted graph pattern, this rewriting step is semantics preserving.

$$\mathcal{D}_*^s(t,\mu) = \begin{cases} \mathcal{D}_i^s(t,\mu) & \text{if } t = (i_s, predicate, object) \\ \emptyset & \text{elsewhere} \end{cases}$$
(14)

The proof of Lemma 6.1 is available in the Appendix B.

6.2 Rewriting by triple pattern's object part

When a property appears on the object part of a triple pattern, we conclude that the triple pattern deals with schema info as there is no way for a non RDF/RDFS/OWL IRI to appear at the same time in the triple pattern's predicate part. Similarly, in case that a class appears on a triple pattern's object part, the only factor which can be used in order to determine the triple pattern's type (Data or Schema Triple Pattern), is whether the RDF property rdf: type appears on the predicate part or not. Thus, the only cases mentioned for the rewriting of a Data Triple Pattern by its object part concern individuals appearing to the triple pattern's object part, as well as classes with the precondition that the RDF property rdf: type appears on the triple pattern's predicate part at the same time.

Rewriting based on class mapping. Let c_s be a class from the source ontology which is mapped to a class expression from the target ontology. Having a Data Triple Pattern $t = (subject, rdf : type, c_s)$ with the class c_s in its object part, the RDF property rdf : typein its predicate and anything in its subject part, we can rewrite it by its object part, using a predefined mapping μ and the function (15).

ſ	$(subject, rdf: type, c_t)$	if $\mu: c_s \to c_t$	
$\mathcal{D}_{c}^{o}(t,\mu) = $	$\mathcal{D}_c^o(t_1,\mu_1)$ UNION $\mathcal{D}_c^o(t_2,\mu_2)$	if $\mu : c_s \rightarrow c_{t1} \sqcup c_{t2}$, where $t_1 = (subject, rdf : type, c_{t1})$, $\mu_1 : c_{t1} \equiv CE_{t1}$, and $t_2 = (subject, rdf : type, c_{t2})$, $\mu_2 : c_{t2} \equiv CE_{t2}$	
	$\mathcal{D}_c^o(t_1,\mu_1)$ AND $\mathcal{D}_c^o(t_2,\mu_2)$	if $\mu : c_s \rightarrow c_{t1} \sqcap c_{t2}$, where $t_1 = (subject, rdf : type, c_{t1})$, $\mu_1 : c_{t1} \equiv CE_{t1}$, and $t_2 = (subject, rdf : type, c_{t2})$, $\mu_2 : c_{t2} \equiv CE_{t2}$	(15)
	$\mathcal{D}_{c}^{o}(t_{1}, \mu_{1}) \text{ AND } \mathcal{D}_{op}^{p}(t_{2}, \mu_{2})$ FILTER(?var $\overline{cp} v_{op}$)	$ \begin{array}{l} \text{if } \mu: \ c_s \rightarrow \forall c_t.(op_t \ \overline{\texttt{cp}} \ v_{op}), \\ \text{where } \overline{\texttt{cp}} \in \{\neq, =\}, \\ v_{op} = individual, \\ t_1 = (subject, rdf: type, c_t), \\ \mu_1: \ c_t \equiv CE_t, \\ \text{and } t_2 = (subject, op_t, ?var), \\ \mu_2: \ op_t \equiv OPE_t \\ \end{array} $	
	$\mathcal{D}_{c}^{o}(t_{1},\mu_{1})$ AND $\mathcal{D}_{dp}^{p}(t_{2},\mu_{2})$ FILTER(?var cp $v_{dp})$	$ \begin{array}{l} \text{if } \mu: \ c_s \rightarrow \forall c_t.(dp_t \ \operatorname{cp} \ v_{dp}), \\ \text{where } \operatorname{cp} \in \{ \neq, =, \leq, \geq, <, > \}, \\ v_{dp} = data \ value, \\ t_1 = (subject, rdf: type, c_t), \\ \mu_1: \ c_t \equiv CE_t, \\ \text{and } t_2 = (subject, dp_t, ?var), \\ \mu_2: \ dp_t \equiv DPE_t \\ \end{array} $	
	$\mathcal{D}_{c}^{o}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{op}^{p}(t_{2},\mu_{2})$ AND $\mathcal{D}_{op}^{p}(t_{3},\mu_{3})$ FILTER(?var_1 cp ?var_2)	$ \begin{array}{l} \text{if } \mu: \ c_s \rightarrow \forall c_t.(op_{t1} \ \overline{\texttt{cp}} \ op_{t2}), \\ \text{where } \overline{\texttt{cp}} \in \{\neq, =\}, \\ t_1 = (subject, rdf: type, c_t), \\ \mu_1: \ c_t \equiv CE_t, \\ t_2 = (subject, op_{t1}, ?var_1), \\ \mu_2: \ op_{t1} \equiv OPE_{t1}, \\ \text{and } t_3 = (subject, op_{t2}, ?var_2), \\ \mu_3: \ op_{t2} \equiv OPE_{t2} \end{array} $	
	$\mathcal{D}_{c}^{o}(t_{1}, \mu_{1}) \text{ AND } \mathcal{D}_{dp}^{p}(t_{2}, \mu_{2})$ AND $\mathcal{D}_{dp}^{p}(t_{3}, \mu_{3})$ FILTER(?var_1 cp ?var_2)	if $\mu : c_s \rightarrow \forall c_t.(dp_{t1} \text{ cp } dp_{t2}),$ where $\text{cp} \in \{\neq, =, \leq, \geq, <, >\},$ $t_1 = (subject, rdf : type, c_t),$ $\mu_1 : c_t \equiv CE_t,$ $t_2 = (subject, dp_{t1}, ?var_1),$ $\mu_2 : dp_{t1} \equiv DPE_{t1},$ and $t_3 = (subject, dp_{t2}, ?var_2),$ $\mu_3 : dp_{t2} \equiv DPE_{t2}$	

The functions \mathcal{D}_{op}^p and \mathcal{D}_{dp}^p are used by the function (15) in order to provide the graph

pattern that forms a restricted property and are defined in Section 6.3.

Example 6.2. Consider the query posed over the source ontology of Figure 1: "Return the poetry books". The SPARQL syntax of the source query is shown below:

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (?x, rdf : type, src : Poetry) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's object part, the result of the triple pattern's t rewriting by its object part is provided by invoking the function (15).

$$\begin{split} t &= (?x, rdf: type, src: Poetry) \\ \mu: \; src: Poetry \; \sqsubseteq \; trg: Literature \end{split}$$

The mapping μ is of type $c_s \to c_t$. Thus, using the parameters defined above, as well as the function (15), the triple pattern t is rewritten as follows:

 $\mathcal{D}_{c}^{o}(t,\mu) = (?x, rdf: type, trg: Literature)$

Example 6.3. Consider the query posed over the source ontology of Figure 1: "Return the scientific books". The SPARQL syntax of the source query is shown below:

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (?x, rdf : type, src : Science) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's object part, the result of the triple pattern's t rewriting by its object part is provided by invoking the function (15).

t = (?x, rdf : type, src : Science)

 μ : src: Science \equiv trg: ComputerScience \sqcup trg: Mathematics

The mapping μ is of type $c_s \to c_{t1} \sqcup c_{t2}$. Following the definition of the function (15), two triple patterns t_1 and t_2 are created and the complex mapping μ is decomposed into the mappings μ_1 and μ_2 . The triple patterns t_1 and t_2 contain the classes c_{t1} and c_{t2} on their object part, respectively. The mapping of the class c_{t1} is provided by μ_1 , while the mapping of the class c_{t2} is provided by μ_2 .

$$t_1 = (?x, rdf : type, c_{t1})$$
$$t_2 = (?x, rdf : type, c_{t2})$$
$$\mu_1 : c_{t1} \equiv trg : ComputerScience$$
$$\mu_2 : c_{t2} \equiv trg : Mathematics$$

Thus,

 $\mathcal{D}_{c}^{o}(t,\mu) = \mathcal{D}_{c}^{o}(t_{1},\mu_{1}) \text{ UNION } \mathcal{D}_{c}^{o}(t_{2},\mu_{2})$

The mappings μ_1 and μ_2 are of type $c_s \to c_t$. Thus, using the parameters defined above, as well as the function (15) for the rewriting of the triple patterns t_1 and t_2 , the initial triple pattern t is rewritten as follows:

$$\begin{aligned} \mathcal{D}_{c}^{o}(t,\mu) &= \mathcal{D}_{c}^{o}(t_{1},\mu_{1}) \text{ UNION } \mathcal{D}_{c}^{o}(t_{2},\mu_{2}) \\ &= (?x,rdf:type,trg:ComputerScience) \text{ UNION } \\ (?x,rdf:type,trg:Mathematics) \end{aligned}$$

Example 6.4. Consider the query posed over the source ontology of Figure 1: "Return the popular scientific books". The SPARQL syntax of the source query is shown below:

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (?x, rdf : type, src : Popular) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's object part, the result of the triple pattern's t rewriting by its object part is provided by invoking the function (15).

t = (?x, rdf : type, src : Popular)

 μ : src: Popular \equiv (trg: ComputerScience \sqcup trg: Mathematics) \sqcap trg: BestSeller

The mapping μ is of type $c_s \to c_{t1} \sqcap c_{t2}$. Following the definition of the function (15), two triple patterns t_1 and t_2 are created and the complex mapping μ is decomposed into the mappings μ_1 and μ_2 . The triple patterns t_1 and t_2 contain the classes c_{t1} and c_{t2} on their object part, respectively. The mapping of the class c_{t1} is provided by μ_1 , while the mapping of the class c_{t2} is provided by μ_2 .

$$t_1 = (?x, rdf : type, c_{t1})$$

$$t_2 = (?x, rdf : type, c_{t2})$$

$$\mu_1 : c_{t1} \equiv trg : ComputerScience \sqcup trg : Mathematics$$

$$\mu_2 : c_{t2} \equiv trg : BestSeller$$

Thus,

$$\mathcal{D}_c^o(t,\mu) = \mathcal{D}_c^o(t_1,\mu_1) \text{ AND } \mathcal{D}_c^o(t_2,\mu_2)$$

Similarly, the resulted complex mapping μ_1 is of type $c_s \rightarrow c_{t3} \sqcup c_{t4}$. Consequently, two triple patterns t_3 and t_4 are created and the complex mapping μ_1 is decomposed into the mappings μ_3 and μ_4 . The triple patterns t_3 and t_4 contain the classes c_{t3} and c_{t4} on their object part respectively. The mapping of the class c_{t3} is provided by μ_3 , while the mapping of the class c_{t4} is provided by μ_4 .

$$t_{3} = (?x, rdf : type, c_{t3})$$
$$t_{4} = (?x, rdf : type, c_{t4})$$
$$\mu_{3} : c_{t3} \equiv trg : ComputerScience$$
$$\mu_{4} : c_{t4} \equiv trg : Mathematics$$

Thus,

$$\mathcal{D}_{c}^{o}(t,\mu) = \mathcal{D}_{c}^{o}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{c}^{o}(t_{2},\mu_{2})$$
$$= \left(\mathcal{D}_{c}^{o}(t_{3},\mu_{3}) \text{ UNION } \mathcal{D}_{c}^{o}(t_{4},\mu_{4})\right) \text{ AND } \mathcal{D}_{c}^{o}(t_{2},\mu_{2})$$

The mappings μ_2 , μ_3 and μ_4 are of type $c_s \to c_t$. Thus, using the parameters defined above, as well as the function (15) for the rewriting of the triple patterns t_2 , t_3 and t_4 , the initial triple pattern t is rewritten as follows:

$$\mathcal{D}_{c}^{o}(t,\mu) = \mathcal{D}_{c}^{o}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{c}^{o}(t_{2},\mu_{2})$$

$$= \left(\mathcal{D}_{c}^{o}(t_{3},\mu_{3}) \text{ UNION } \mathcal{D}_{c}^{o}(t_{4},\mu_{4})\right) \text{ AND } \mathcal{D}_{c}^{o}(t_{2},\mu_{2})$$

$$= \left(\left(?x,rdf:type,trg:ComputerScience\right) \text{ UNION } (?x,rdf:type,trg:Mathematics}\right) \text{ AND } (?x,rdf:type,trg:BestSeller)$$

Rewriting based on individual mapping. Let i_s be an individual from the source ontology which is mapped to an individual i_t from the target ontology. Having a Data Triple Pattern $t = (subject, predicate, i_s)$ with i_s in its object part and anything in its predicate and subject parts, we can rewrite it by its object part, using a predefined mapping μ and the function (16).

$$\mathcal{D}_{i}^{o}(t,\mu) = (subject, predicate, i_{t}) \quad \text{if } \mu: \ i_{s} \equiv i_{t}$$

$$(16)$$

Example 6.5. Consider the query posed over the source ontology of Figure 1: "Return the individuals which are specified to be the same with the CSFoundations individual". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.
@PREFIX owl: <http://www.w3.org/2002/07/owl#>.
```

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (?x, owl : sameAs, src : CSFoundations) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's object part, the result of the triple pattern's t rewriting by its object part is provided by invoking the function (16).

t = (?x, owl : sameAs, src : CSFoundations) $\mu : src : CSFoundations \equiv trg : FoundationsOfCS$

Using the parameters defined above, as well as the function (16), the triple pattern t is rewritten as follows:

 $\mathcal{D}_{i}^{o}(t,\mu) = (?x, owl : sameAs, trg : FoundationsOfCS)$

In Lemma 6.2 we summarize the functions presented in this section, which are used for the rewriting of a Data Triple Pattern based on a mapping for the triple pattern's object part.

Lemma 6.2. Having a Data Triple Pattern t and a predefined mapping μ for its object part, we can rewrite it by its object, by invoking the function (17). Considering the semantics of the initial triple pattern, as well as the semantics of the resulted graph pattern, this rewriting step is semantics preserving.

$$\mathcal{D}^{o}_{*}(t,\mu) = \begin{cases} \mathcal{D}^{o}_{i}(t,\mu) & \text{if } t = (subject, predicate, i_{s}) \\ \mathcal{D}^{o}_{c}(t,\mu) & \text{if } t = (subject, rdf : type, c_{s}) \\ \emptyset & \text{elsewhere} \end{cases}$$
(17)

The proof of Lemma 6.2 is available in the Appendix B.

6.3 Rewriting by triple pattern's predicate part

In order to rewrite a Data Triple Pattern by its predicate part only property mappings can be used, since a class or an individual cannot appear on a triple pattern's predicate part.

Rewriting based on object property mapping. Let op_s be an object property from the source ontology which is mapped to an object property expression from the target ontology. Having a Data Triple Pattern $t = (subject, op_s, object)$ with op_s in its predicate part and anything in its subject and object parts, we can rewrite it by its predicate part, using a predefined mapping μ and the function (18).

$\mathcal{D}_{op}^{p}(t,\mu) = \langle$	$(subject, op_t, object)$	if $\mu: op_s \to op_t$
	$\mathcal{D}_{op}^{p}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{op}^{p}(t_{2},\mu_{2})$	$ \begin{array}{l} \text{if } \mu: \ op_s \rightarrow op_{t1} \circ op_{t2}, \\ \text{where } t_1 = (subject, op_{t1}, ?var), \\ \mu_1: \ op_{t1} \equiv OPE_{t1}, \\ \text{and } t_2 = (?var, op_{t2}, object), \\ \mu_2: \ op_{t2} \equiv OPE_{t2} \end{array} $
	$\mathcal{D}_{op}^{p}(t_{1},\mu_{1})$ UNION $\mathcal{D}_{op}^{p}(t_{2},\mu_{2})$	$ \begin{array}{l} \text{if } \mu: \ op_s \rightarrow op_{t1} \sqcup op_{t2}, \\ \text{where } t_1 = (subject, op_{t1}, object), \\ \mu_1: \ op_{t1} \equiv OPE_{t1}, \\ \text{and } t_2 = (subject, op_{t2}, object), \\ \mu_2: \ op_{t2} \equiv OPE_{t2} \end{array} $
	$\mathcal{D}_{op}^{p}(t_1,\mu_1)$ AND $\mathcal{D}_{op}^{p}(t_2,\mu_2)$	$ \begin{array}{l} \text{if } \mu: \ op_s \to op_{t1} \sqcap op_{t2}, \\ \text{where } t_1 = (subject, op_{t1}, object), \\ \mu_1: \ op_{t1} \equiv OPE_{t1}, \\ \text{and } t_2 = (subject, op_{t2}, object), \\ \mu_2: \ op_{t2} \equiv OPE_{t2} \end{array} $ (18)
	$\mathcal{D}_{op}^p(t_1,\mu_1)$	$ \begin{array}{l} \text{if } \mu: \ op_s \rightarrow inv(op_t), \\ \text{where } t_1 = (object, op_t, subject) \\ \text{and } \mu_1: \ op_t \equiv OPE_t \end{array} $
	$\mathcal{D}_{op}^{p}(t_{1},\mu_{1})$ AND $\mathcal{D}_{c}^{o}(t_{2},\mu_{2})$	$ \begin{array}{l} \text{if } \mu: \ op_s \rightarrow \forall op_t.domain(c_t), \\ \text{where } t_1 = (subject, op_t, object), \\ \mu_1: \ op_t \equiv OPE_t, \\ \text{and } t_2 = (subject, rdf: type, c_t), \\ \mu_2: \ c_t \equiv CE_t \end{array} $
	$\mathcal{D}_{op}^{p}(t_{1},\mu_{1})$ AND $\mathcal{D}_{c}^{o}(t_{2},\mu_{2})$	$ \begin{array}{l} \text{if } \mu: \ op_s \rightarrow \forall op_t.range(c_t), \\ \text{where } t_1 = (subject, op_t, object), \\ \mu_1: \ op_t \equiv OPE_t, \\ \text{and } t_2 = (object, rdf: type, c_t), \\ \mu_2: \ c_t \equiv CE_t \end{array} $

Example 6.6. Consider the query posed over the source ontology of Figure 1: "Return the publisher of the book CSFoundations". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.
SELECT ?x
WHERE
{
    src:CSFoundations src:publisher ?x.
}
```

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (src : CSFoundations, src : publisher, ?x) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's predicate part, the result of the triple pattern's t rewriting by its predicate part is provided by invoking the function (18).

$$t = (src: CSF oundations, src: publisher, ?x)$$

 $\mu: src: publisher \equiv inv(trg: publishes)$

The mapping μ is of type $op_s \rightarrow inv(op_t)$. Following the definition of the function (18), a triple patterns t_1 is created and the complex mapping μ is transformed to the mapping μ_1 . The triple pattern t_1 contains an object property op_t on its predicate part and its mapping is provided by μ_1 .

 $t_1 = (?x, op_t, src : CSF oundations)$ $\mu_1 : op_t \equiv trg : publishes$

Thus,

$$\mathcal{D}_{op}^p(t,\mu) = \mathcal{D}_{op}^p(t_1,\mu_1)$$

The mapping μ_1 is of type $op_s \to op_t$. Thus, using the parameters defined above, as well as the function (18) for the rewriting of the triple pattern t_1 , the initial triple pattern t is rewritten as follows:

$$\mathcal{D}_{op}^{p}(t,\mu) = \mathcal{D}_{op}^{p}(t_{1},\mu_{1})$$
$$= (?x,trg: publishes, src: CSF oundations)$$

Rewriting based on datatype property mapping. Let dp_s be a datatype property from the source ontology which is mapped to a datatype property expression from the target ontology. Having a Data Triple Pattern $t = (subject, dp_s, object)$ with dp_s in its predicate part and anything in its subject and object parts, we can rewrite it by its predicate part, using a predefined mapping μ and the function (19).

$$\mathcal{D}_{pp}^{p}(t,\mu) = \begin{cases} (subject, dp_{t}, object) & \text{if } \mu : dp_{s} \rightarrow dp_{t} \\ \mathcal{D}_{pp}^{p}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{dp}^{p}(t_{2},\mu_{2}) & \text{if } \mu : dp_{s} \rightarrow op_{t} \circ dp_{t}, \\ \text{where } t_{1} = (subject, op_{t}, ?var), \\ \mu_{1} : op_{t} \equiv OPE_{t}, \\ \mathcal{D}_{dp}^{p}(t_{1},\mu_{1}) \text{ UNION } \mathcal{D}_{dp}^{p}(t_{2},\mu_{2}) & \text{if } \mu : dp_{s} \rightarrow dp_{t} \sqcup dp_{t2}, \\ \text{where } t_{1} = (subject, dp_{t1}, object), \\ \mu_{1} : dp_{t} \equiv DPE_{t}, \\ \text{and } t_{2} = (subject, dp_{t2}, object), \\ \mu_{2} : dp_{t2} \equiv DPE_{t2} \\ \mathcal{D}_{dp}^{p}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{dp}^{p}(t_{2},\mu_{2}) & \text{if } \mu : dp_{s} \rightarrow dp_{t} \sqcap dp_{t2}, \\ \text{where } t_{1} = (subject, dp_{t1}, object), \\ \mu_{1} : dp_{t1} \equiv DPE_{t1}, \\ \text{and } t_{2} = (subject, dp_{t1}, object), \\ \mu_{1} : dp_{t1} \equiv DPE_{t1}, \\ \text{and } t_{2} = (subject, dp_{t2}, object), \\ \mu_{2} : dp_{t2} \equiv DPE_{t2} \\ \mathcal{D}_{dp}^{p}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{c}^{o}(t_{2},\mu_{2}) & \text{if } \mu : dp_{s} \rightarrow \forall dp_{t}. domain(c_{t}), \\ \text{where } t_{1} = (subject, dp_{t}, object), \\ \mu_{1} : dp_{t} \equiv DPE_{t2} \\ \mathcal{D}_{dp}^{p}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{c}^{o}(t_{2},\mu_{2}) & \text{if } \mu : dp_{s} \rightarrow \forall dp_{t}. domain(c_{t}), \\ \text{where } t_{1} = (subject, dp_{t}, object), \\ \mu_{1} : dp_{t} \equiv DPE_{t}, \\ \text{and } t_{2} = (subject, rdf : type, c_{t}), \\ \mu_{2} : c_{t} \equiv CE_{t} \\ \mathcal{D}_{dp}^{p}(t_{1},\mu_{1}) & \text{if } \mu : dp_{s} \rightarrow \forall dp_{t}. range(cp \ v_{dp}), \\ \text{FILTER}(object \ cp \ v_{dp}) & \text{where } cp \in \{\neq, =, \leq, >, <, >\}, \\ v_{dp} = data \ value, \\ \text{and } t_{1} = (subject, dp_{t}, object), \\ \mu_{1} : dp_{t} \equiv DPE_{t} \\ \text{otherwise} DPE_{$$

Example 6.7. Consider the query posed over the source ontology of Figure 1: "Return the name of the CSFoundations individual which is of type Book". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.
```

SELECT ?x
WHERE
{
 src:CSFoundations src:name ?x.
}

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (src : CSFoundations, src : name, ?x) by its

subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's predicate part, the result of the triple pattern's t rewriting by its predicate part is provided by invoking the function (19).

t = (src : CSF oundations, src : name, ?x)

 μ : src:name \supseteq trg:title

The mappings μ is of type $dp_s \to dp_t$. Thus, using the parameters defined above, as well as the function (19), the triple pattern t is rewritten as follows:

$$\mathcal{D}_{dp}^{p}(t,\mu) = (src: CSFoundations, trg: title, ?x)$$

Example 6.8. Consider the query posed over the source ontology of Figure 1: "Return the available reviews for the book CSFoundations". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.
```

```
SELECT ?x
WHERE
{
    src:CSFoundations src:review ?x.
}
```

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (src : CSFoundations, src : review, ?x) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's predicate part, the result of the triple pattern's t rewriting by its predicate part is provided by invoking the function (19).

t = (src : CSF oundations, src : review, ?x) $\mu : src : review \equiv trg : editorial Review \sqcup trg : customer Review$

The mapping μ is of type $dp_s \rightarrow dp_{t1} \sqcup dp_{t2}$. Following the definition of the function (19), two triple patterns t_1 and t_2 are created and the complex mapping μ is decomposed into the mappings μ_1 and μ_2 . The triple patterns t_1 and t_2 contain the datatype properties dp_{t1} and dp_{t2} on their predicate part, respectively. The mapping of the datatype property dp_{t1} is provided by μ_1 , while the mapping of the datatype property dp_{t2} is provided by μ_2 .

 $t_{1} = (src : CSF oundations, dp_{t1}, ?x)$ $t_{2} = (src : CSF oundations, dp_{t2}, ?x)$ $\mu_{1} : dp_{t1} \equiv trg : editorial Review$ $\mu_{2} : dp_{t2} \equiv trg : customer Review$

Thus,

$$\mathcal{D}_{dp}^p(t,\mu) = \mathcal{D}_{dp}^p(t_1,\mu_1) \text{ UNION } \mathcal{D}_{dp}^p(t_2,\mu_2)$$

The mappings μ_1 and μ_2 are of type $dp_s \to dp_t$. Thus, using the parameters defined above, as well as the function (19) for the rewriting of the triple patterns t_1 and t_2 , the initial triple pattern t is rewritten as follows:

$$\mathcal{D}_{dp}^{p}(t,\mu) = \mathcal{D}_{dp}^{p}(t_{1},\mu_{1}) \text{ UNION } \mathcal{D}_{dp}^{p}(t_{2},\mu_{2})$$

= (src: CSF oundations, trg: editorial Review, ?x) UNION(src: CSF oundations, trg: customer Review, ?x)

In Lemma 6.3 we summarize the functions presented in this section, which are used for the rewriting of a Data Triple Pattern based on a mapping for the triple pattern's predicate part.

Lemma 6.3. Having a Data Triple Pattern t and a predefined mapping μ for its predicate part, we can rewrite it by its predicate, by invoking the function (20). Considering the semantics of the initial triple pattern, as well as the semantics of the resulted graph pattern, this rewriting step is semantics preserving.

$$\mathcal{D}^{p}_{*}(t,\mu) = \begin{cases} \mathcal{D}^{p}_{op}(t,\mu) & \text{if } t = (subject, op_{s}, object) \\ \mathcal{D}^{p}_{dp}(t,\mu) & \text{if } t = (subject, dp_{s}, object) \\ \emptyset & \text{elsewhere} \end{cases}$$
(20)

The proof of Lemma 6.3 is available in the Appendix B.

6.4 Combination examples

In this section we provide a set of examples that combine some of the functions presented in the previous sections in order to rewrite a triple pattern based on a specific triple pattern's part (i.e. subject, predicate, object).

Example 6.9. Consider the query posed over the source ontology of Figure 1: "Return the pocket-sized books". The SPARQL syntax of the source query is shown below:

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (?x, rdf : type, src : Pocket) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's object part, the result of the triple pattern's t rewriting by its object part is provided by invoking the function (15).

t = (?x, rdf : type, src : Pocket)

$$\mu: src: Pocket \equiv \forall trg: Textbook.(trg: size \leq 14)$$

Taking a closer look at the mapping μ , we conclude that the source ontology's class **Pocket** is mapped to the target ontology's class **Textbook**, restricted on its **size** property values. Consequently, the mapping μ is of type $c_s \to \forall c_t.(dp_t \text{ cp } v_{dp})$. Following the definition of the function (15), two triple patterns t_1 and t_2 are created and the complex mapping μ is decomposed into the mappings μ_1 and μ_2 . The triple pattern t_1 contains a class c_t on its object part, while the triple pattern t_2 contains a datatype property dp_t on its predicate part. The mapping of the class c_t is provided by μ_1 , while the mapping of the property dp_t is provided by μ_2 .

 $t_1 = (?x, rdf : type, c_t)$ $t_2 = (?x, dp_t, ?var)$ $\mu_1 : c_t \equiv trg : Textbook$ $\mu_2 : dp_t \equiv trg : size$

Thus,

$$\mathcal{D}_{c}^{o}(t,\mu) = \mathcal{D}_{c}^{o}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{dp}^{p}(t_{2},\mu_{2}) \text{ FILTER}(?var \leq 14)$$

The mapping μ_2 , as well as the triple pattern t_2 are used by the function (19), in order to form the graph pattern representing the **size** property. Thus, using the parameters defined above, as well as the function (15) for the rewriting of the trille pattern t_1 , the initial triple pattern t is rewritten as follows:

 $\begin{aligned} \mathcal{D}_{c}^{o}(t,\mu) &= \mathcal{D}_{c}^{o}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{dp}^{p}(t_{2},\mu_{2}) \text{ FILTER}(?var \leq 14) \\ &= (?x,rdf:type,trg:Textbook) \text{ AND } (?x,trg:size,?var) \\ &\text{FILTER}(?var \leq 14) \end{aligned}$

Example 6.10. Consider the query posed over the source ontology of Figure 1: "Return the autobiography books". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
```

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (?x, rdf : type, src : Autobiography) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's object part, the result of the triple pattern's t rewriting by its object part is provided by invoking the function (15).

t = (?x, rdf : type, src : Autobiography)

μ : src: Autobiography $\equiv \forall trg: Biography.(trg: author = trg: topic)$

Taking a closer look at the mapping μ , we conclude that the source ontology's class **Autobiography** is mapped to the target ontology's class **Biography**, restricted on its **author** property values. The mapping μ is of type $c_s \to \forall c_t.(dp_{t1} \text{ cp } dp_{t2})$. Following the definition of the function (15), three triple patterns t_1, t_2 and t_3 are created and the complex mapping μ is decomposed into the mappings μ_1, μ_2 and μ_3 . The triple pattern t_1 contains a class c_t on its object part, while the triple patterns t_2 and t_3 contain the datatype properties dp_{t1} and dp_{t2} on their predicate part, respectively. The mapping of the class c_t is provided by μ_1 , the mapping of the property dp_{t1} is provided by μ_2 and the mapping of the property dp_{t2} is provided by μ_3 .

 $t_{1} = (?x, rdf : type, c_{t})$ $t_{2} = (?x, dp_{t1}, ?var_{1})$ $t_{3} = (?x, dp_{t2}, ?var_{2})$ $\mu_{1} : c_{t} \equiv trg : Biography$ $\mu_{2} : dp_{t1} \equiv trg : author$ $\mu_{2} : dp_{t2} \equiv trg : topic$

Thus,

$$\mathcal{D}_{c}^{o}(t,\mu) = \mathcal{D}_{c}^{o}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{dp}^{p}(t_{2},\mu_{2})$$

AND $\mathcal{D}_{dp}^{p}(t_{3},\mu_{3}) \text{ FILTER}(?var_{1}=?var_{2})$

The mapping μ_2 , as well as the triple pattern t_2 are used by the function (19), in order to form the graph pattern representing the property **author**. Similarly, the mapping μ_3 , as well as the triple pattern t_3 are used by the same function, in order to form the graph pattern representing the property **topic**. Finally, using the parameters defined above, as well as the function (15) for the rewriting of the triple pattern t_1 , the initial triple pattern t is rewritten as follows:

$$\begin{aligned} \mathcal{D}_{c}^{o}(t,\mu) &= \mathcal{D}_{c}^{o}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{dp}^{p}(t_{2},\mu_{2}) \\ & \text{AND } \mathcal{D}_{dp}^{p}(t_{3},\mu_{3}) \text{ FILTER}(?var_{1}=?var_{2}) \end{aligned} \\ &= (?x,rdf:type,trg:Biography) \text{ AND } (?x,trg:author,?var_{1}) \\ & \text{ AND } (?x,trg:topic,?var_{2}) \text{ FILTER}(?var_{1}=?var_{2}) \end{aligned}$$

Example 6.11. Consider the query posed over the source ontology of Figure 1: "Return the pocket-sized books which are part of a collection". The SPARQL syntax of the source query is shown below:

@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (?x, src : partOf, ?y) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's predicate part, the result of the triple pattern's t rewriting by its predicate part is provided by invoking the function (18).

t = (?x, src: partOf, ?y) $\mu: src: partOf \equiv \forall trg: partOf.domain(\forall trg: Textbook.(trg: size \le 14))$

Taking a closer look at the mapping μ , we conclude that the source ontology's property **partOf** is mapped to the target ontology's property **partOf**, restricted on its domain values. Consequently, the mapping μ is of type $op_s \rightarrow \forall op_t.domain(c_{t1})$. Following the definition of the function (18), two triple patterns t_1 and t_2 are created and the complex mapping μ is decomposed into the mappings μ_1 and μ_2 . The triple pattern t_1 contains an object property op_t on its predicate part and its mapping is provided by μ_1 , while the triple pattern t_2 contains a class c_{t1} on its object part and its mapping is provided by μ_2 .

$$t_1 = (?x, op_t, ?y)$$

$$t_2 = (?x, rdf : type, c_{t1})$$

$$\mu_1 : op_t \equiv trg : partOf$$

$$\mu_2 : c_{t1} \equiv \forall trg : Textbook.(trg : size \le 14)$$

Thus,

$$\mathcal{D}_{op}^p(t,\mu) = \mathcal{D}_{op}^p(t,\mu_1) \text{ AND } \mathcal{D}_c^o(t_2,\mu_2)$$

The mapping μ_2 actually specifies a mapping between the domain of the source ontology's property **partOf** and the target ontology's class **Textbook**, restricted on its **size** property values. Following the definition of the function (15), two triple patterns t_3 and t_4 are created and the complex mapping μ_2 is decomposed into the mappings μ_3 and μ_4 . The triple pattern t_3 contains a class c_{t2} on its object part, while the triple pattern t_4 contains a datatype property dp_t on its predicate part. The mapping of the class c_{t2} is provided by μ_3 , while the mapping of the property dp_t is provided by μ_4 .

$$t_3 = (?x, rdf : type, c_{t2})$$

$$t_4 = (?x, dp_t, ?var)$$
$$\mu_3 : c_{t2} \equiv trg : Textbook$$
$$\mu_4 : dp_t \equiv trg : size$$

Thus,

$$\mathcal{D}_{op}^{p}(t,\mu) = \mathcal{D}_{op}^{p}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{c}^{o}(t_{2},\mu_{2})$$
$$= \mathcal{D}_{op}^{p}(t_{1},\mu_{1}) \text{ AND } \left(\mathcal{D}_{c}^{o}(t_{3},\mu_{3}) \text{ AND } \mathcal{D}_{dp}^{p}(t_{4},\mu_{4}) \right.$$
$$\operatorname{FILTER}(?var \leq 14))$$

The mapping μ_4 , as well as the triple pattern t_4 are used by the function (19) in order to form the graph pattern representing the **size** property. Thus, using the parameters defined above, as well as the functions (18) and (15) for the rewriting of the triple patterns t_1 and t_3 , the initial triple pattern t is rewritten as follows:

Example 6.12. Consider the query posed over the source ontology of Figure 1: "Return the authors of the book CSFoundations". The SPARQL syntax of the source query is shown below:

@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.

```
SELECT ?x
WHERE
{
    src:CSFoundations src:author ?x.
}
```

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (src : CSFoundations, src : author, ?x) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's predicate part, the result of the triple pattern's t rewriting by its predicate part is provided by invoking the function (19).

t = (src: CSF oundations, src: author, ?x)

 $\mu: src: author \equiv trg: author \circ trg: name$

The mapping μ is of type $dp_s \to op_t \circ dp_t$. Following the definition of the function (19), two triple patterns t_1 and t_2 are created and the complex mapping μ is decomposed into the mappings μ_1 and μ_2 . The triple patterns t_1 and t_2 contain an object property op_t and a datatype property dp_t on their predicate part, respectively. The mapping of the object property op_t is provided by μ_1 , while the mapping of the datatype property dp_t is provided by μ_2 .

 $t_{1} = (src : CSF oundations, op_{t}, ?var)$ $t_{2} = (?var, dp_{t}, ?x)$ $\mu_{1} : op_{t} \equiv trg : author$ $\mu_{2} : dp_{t} \equiv trg : name$

Thus,

$$\mathcal{D}_{dp}^p(t,\mu) = \mathcal{D}_{op}^p(t_1,\mu_1) \text{ AND } \mathcal{D}_{dp}^p(t_2,\mu_2)$$

The mapping μ_1 is of type $op_s \rightarrow op_t$, while the mapping μ_2 is of type $dp_s \rightarrow dp_t$. Thus, using the parameters defined above, as well as the functions (18) and (19) for the rewriting of the triple patterns t_1 and t_2 , the initial triple pattern t is rewritten as follows:

$$\mathcal{D}_{dp}^{p}(t,\mu) = \mathcal{D}_{op}^{p}(t_{1},\mu_{1}) \text{ AND } \mathcal{D}_{dp}^{p}(t_{2},\mu_{2})$$
$$= (src: CSFoundations, trg: author, ?var) \text{ AND}$$
$$(?var, trg: name, ?x)$$

7 Schema Triple Pattern rewriting

In order to rewrite a triple pattern any mapping type presented in Section 3.3 can be used. However, in some cases the mapped expressions should be relaxed in order for a mapping to be used by the Schema Triple pattern rewriting process.

In the Example 7.1, we show that a mapping is used as it is in order to be exploited by the Data Triple Pattern rewriting process, although, the same mapping should be relaxed in order to be used for the rewriting of a Schema Triple Pattern.

Example 7.1. In Figure 1, let the source ontology's class **Pocket** be mapped to the class **Textbook** from the target ontology, restricted on its **size** property values. This correspondence can be represented as follows:

$$\mu: src: Pocket \equiv \forall trg: Textbook.(trg: size \leq 14)$$

Having a Data Triple Pattern t = (?x, rdf : type, src : Pocket) and the mapping μ , it is clear that the entire mapping should be used in order to rewrite the triple pattern t. This results from the fact that the mapping μ relates a class from the source ontology with an unnamed class (i.e. set of instances) from the target ontology, and the triple pattern tconcerns data info and specifically a set of instances. On the contrary, having the mapping μ that we presented before, as well as a Schema Triple Pattern t' = (src : Pocket, rdfs : subClassOf, ?x), it is clear that the class restriction cannot be used in order to rewrite it. As mentioned before, the class Pocket is mapped to an unnamed class. Thus, using the class restriction for the rewriting of t', makes the evaluation results prone to whether the target ontology defines the unnamed class, which is very unlikely, and contains schema info about it as well. Consequently, in order to rewrite the triple pattern t' and also to avoid tricky hypothesis, the mapping μ should be transformed to a similar one, having the property restriction of the target ontology's mapped expression removed (i.e. $\mu' : src : Pocket \sqsubseteq trg : Textbook$). Such a relaxation step, seems to be reliable for Schema Triple Patterns, in the sense that it is based on some inferred facts which are more likely to return the desirable query results.

The operations that determine whether a mapping should be relaxed in order to be used for the rewriting of a Schema Triple Pattern are the following:

- Class expression restrictions.
- Object/datatype property expression restrictions on domain/range values.
- Composition operations between object/datatype property expressions.
- Inverse object property expression operations.

Mapped expressions containing the above operations are relaxed in order to be used for the rewriting of a Schema Triple Pattern. In this case, a mapped class expression CE (see Definition 3.1) is transformed to a similar class expression CE' (defined recursively in (21)), having any class restrictions removed.

$$CE' := c \mid CE' \sqcap CE' \mid CE' \sqcup CE'$$

$$(21)$$

A mapped object property expression OPE (see Definition 3.2) is transformed to a similar object property expression OPE' (defined recursively in (22)), having any domain/range restrictions, any composed object property expressions and any inverse object property expressions removed.

$$OPE' := op \mid OPE' \sqcap OPE' \mid OPE' \sqcup OPE'$$

$$(22)$$

Similarly, a mapped datatype property expression DPE (see Definition 3.3) is transformed to a similar datatype property expression DPE' (defined recursively in (23)), having any domain/range restrictions and any composed property expressions removed.

$$DPE' := dp \mid DPE' \sqcap DPE' \mid DPE' \sqcup DPE'$$
(23)

Mappings containing mapped expressions that need relaxation, are transformed by substituting the mapped expression with the relaxed one and by modifying the mapping's relationship respectively. The relaxation operations presented above, can also exclude a mapping from being used for the rewriting of a Schema Triple Pattern. For example, a mapping between an object property and a composition of object properties is exluded, since the relaxation method will remove the composition operation and consequently the entire mapped expression. It is worth to say that mappings between individuals do no need any relaxation in order to be used for the rewriting of a Schema Triple Pattern.

Even after preprocessing the defined mappings, a Schema Triple Pattern should be rewritten differently compared to a Data Triple Pattern. The need for handling differently these two different triple pattern types lies on the fact that a Data Triple Pattern deals with data info (e.g. relationships between instances or between instances and data values), while a Schema Triple Pattern deals with schema info (e.g. hierarchies and relationships between *named* classes and/or *named* properties).

In the Example 7.2, we show that handling the rewriting of a Schema Triple Pattern in the same manner with a Data Triple Pattern does not preserve the mapping semantics.

Example 7.2. In Figure 1, let the source ontology's class Science be mapped to the union of classes ComputerScience and Mathematics from the target ontology. This correspondence can be represented as follows:

μ : src: Science \equiv trg: ComputerScience \sqcup trg: Mathematics

A Data Triple Pattern t = (?x, rdf : type, src : Science), involves the instances of class Science. Taking into consideration the mapping μ , the rewritten graph pattern of t should return the instances of the class ComputerScience, as well as the instances of the class Mathematics, using the UNION graph pattern operator.

On the contrary, a Schema Triple Pattern t' = (src: Science, rdfs: subClassOf, ?x)involves the superclasses of the class Science. Using the mapping μ in order to rewrite t', someone would expect the rewritten graph pattern to return the superclasses of the union of classes ComputerScience and Mathematics. Although, such a rewritten graph pattern is very unlikely to match any RDF graph (i.e. no results obtained), due to the fact that the union of classes ComputerScience and Mathematics is not a named class in the target ontology, in order to contain schema info about it. In addition, this differs from returning the superclasses of the class ComputerScience, as well as the superclasses of the class Mathematics, following the treatment which was used for Data Triple Pattern rewriting.

One method to make a rewritten graph pattern semantically correspondent to the initial triple pattern t' is by using inference. In this case, a graph pattern that matches the common superclasses of the classes ComputerScience and Mathematics forms the solution.

In order to rewrite a Schema Triple Pattern using 1:N cardinality mappings, simple types of inference based on DL axioms are used. The Schema Triple Patterns which can be handled using inference are those having on their predicate part one of the OWL/RDF/ RDFS properties appearing on the set *SSP* (Supported Schema Predicates - see (24)).

$$SSP = \begin{cases} rdf : type, \\ rdfs : subClassOf, \\ rdfs : subPropertyOf, \\ owl : equivalentClass, \\ owl : equivalentProperty, \\ owl : complementOf, \\ owl : disjointWith \end{cases}$$

$$(24)$$

Let SSP_c (see (25)) be the supported OWL/RDF/RDFS property set which can be applied on classes, and SSP_p (see (26)) be the supported OWL/RDF/RDFS property set which can be applied on properties.

$$SSP_{c} = \begin{cases} rdf : type, \\ rdfs : subClassOf, \\ owl : equivalentClass, \\ owl : complementOf, \\ owl : disjointWith \end{cases}$$
(25)
$$SSP_{p} = \begin{cases} rdf : type, \\ rdfs : subPropertyOf, \\ owl : equivalentProperty \end{cases}$$
(26)

The sets presented above are divided further, to the sets
$$SSP'_c$$
 (see (27)) and SSP'_p (see (28)) respectively, for the purpose of common inference treatment.

$$SSP'_{c} = SSP_{c} - \{rdfs : subClassOf\}$$

$$(27)$$

$$SSP'_{p} = SSP_{p} - \{rdfs : subPropertyOf\}$$
⁽²⁸⁾

Let B, C, D, G be atomic concepts (i.e. classes) and K, L, R, S be atomic roles (i.e. properties). The Table 12 and Table 13 summarize the class and property axioms which are used for the rewriting of Schema Triple Patterns, respectively. We note that the complement operation is denoted by using the indicator c.

Table 12: Class axioms used for the rewriting of Schema Triple Patterns.

Type	Axioms
Subsumption	if $B \sqsubseteq C$ and $G \equiv C$ then $B \sqsubseteq G$
	if $B \sqsubseteq C$ and $B \sqsubseteq D$ and $G \equiv C \sqcap D$ then $B \sqsubseteq G$
	if $B \sqsubseteq C$ or $B \sqsubseteq D$ and $G \equiv C \sqcup D$ then $B \sqsubseteq G$
	if $B \sqsupseteq C$ and $G \equiv C$ then $B \sqsupseteq G$
	if $B \sqsupseteq C$ or $B \sqsupseteq D$ and $G \equiv C \sqcap D$ then $B \sqsupseteq G$
	if $B \sqsupseteq C$ and $B \sqsupseteq D$ and $G \equiv C \sqcup D$ then $B \sqsupseteq G$
Equivalence	if $B \equiv C$ and $G \equiv C$ then $B \equiv G$
	if $B \equiv C$ and $B \equiv D$ and $G \equiv C \sqcap D$ then $B \equiv G$
	if $B \equiv C$ and $B \equiv D$ and $G \equiv C \sqcup D$ then $B \equiv G$
Complementarity	if $B \equiv C^c$ and $G \equiv C$ then $B \equiv G^c$
	if $B \equiv C^c$ and $B \equiv D^c$ and $G \equiv C \sqcap D$ then $B \equiv G^c$
	if $B \equiv C^c$ and $B \equiv D^c$ and $G \equiv C \sqcup D$ then $B \equiv G^c$
Disjointness	if $B \sqcap C = \emptyset$ and $G \equiv C$ then $B \sqcap G = \emptyset$
	if $B \sqcap C = \emptyset$ and $B \sqcap D = \emptyset$ and $G \equiv C \sqcap D$ then $B \sqcap G = \emptyset$
	if $B \sqcap C = \emptyset$ and $B \sqcap D = \emptyset$ and $G \equiv C \sqcup D$ then $B \sqcap G = \emptyset$

It is worth to say that in case of 1:1 cardinality mappings, every Schema Triple Pattern having any OWL/RDF/RDFS property on its predicate part can be rewritten. In Section 7.1 we describe the Schema Triple Pattern rewriting process using a mapping for the triple pattern's subject part, while in Section 7.2 we present the Schema Triple Pattern rewriting process using a mapping for the triple pattern's object part. The rewriting of a Schema Triple Pattern by its predicate part does not result in modifications since the triple pattern's predicate part is an RDF/RDFS/OWL property which does not affect the rewriting procedure.

Table 13: Property axioms used for the rewriting of Schema Triple Patterns.

Type	Axioms	
Subsumption	if $K \sqsubseteq L$ and $S \equiv L$ then $K \sqsubseteq S$	
	if $K \sqsubseteq L$ and $K \sqsubseteq R$ and $S \equiv L \sqcap R$ then $K \sqsubseteq S$	
	if $K \sqsubseteq L$ or $K \sqsubseteq R$ and $S \equiv L \sqcup R$ then $K \sqsubseteq S$	
	if $K \supseteq L$ and $S \equiv L$ then $K \supseteq S$	
	if $K \supseteq L$ or $K \supseteq R$ and $S \equiv L \sqcap R$ then $K \supseteq S$	
	if $K \sqsupseteq L$ and $K \sqsupseteq R$ and $S \equiv L \sqcup R$ then $K \sqsupseteq S$	
Equivalence	if $K \equiv L$ and $S \equiv L$ then $K \equiv S$	
	if $K \equiv L$ and $K \equiv R$ and $S \equiv L \sqcap R$ then $K \equiv S$	
	if $K \equiv L$ and $K \equiv R$ and $S \equiv L \sqcup R$ then $K \equiv S$	

7.1 Rewriting by triple pattern's subject part

A class or a property can appear in the subject part of a Schema Triple Pattern, as opposed to the Data Triple Patterns.

Rewriting based on class mapping. Let c_s be a class from the source ontology which is mapped to a class expression from the target ontology. Having a Schema Triple Pattern $t = (c_s, predicate, object)$ with c_s in its subject part, an RDF/RDFS/OWL property in its predicate and anything in its object part, we can rewrite it by its subject part, using a predefined mapping μ and the function (29).

$$S_{c}^{s}(t,\mu) = \begin{cases} (c_{t}, predicate, object) & \text{if } \mu : c_{s} \to c_{t} \\ S_{c}^{s}(t_{1},\mu_{1}) \text{ UNION } S_{c}^{s}(t_{2},\mu_{2}) & \text{if } \mu : c_{s} \to c_{t1} \sqcap c_{t2} \text{ and} \\ predicate = rdfs : subClassOf, \\ \text{where } t_{1} = (c_{t1}, predicate, object), \\ \mu_{1} : c_{t1} \equiv CE_{t1}, \\ \text{and } t_{2} = (c_{t2}, predicate, object), \\ \mu_{2} : c_{t2} \equiv CE_{t2} \end{cases}$$

$$S_{c}^{s}(t_{1},\mu_{1}) \text{ AND } S_{c}^{s}(t_{2},\mu_{2}) & \text{if } \mu : c_{s} \to c_{t1} \{\sqcap \mid \sqcup \}c_{t2} \text{ and} \\ predicate \in SSP_{c}', \\ \text{or if } \mu : c_{s} \to c_{t1} \sqcup c_{t2} \text{ and} \\ predicate = rdfs : subClassOf, \\ \text{where } t_{1} = (c_{t1}, predicate, object), \\ \mu_{1} : c_{t1} \equiv CE_{t1}, \\ \text{and } t_{2} = (c_{t2}, predicate, object), \\ \mu_{2} : c_{t2} \equiv CE_{t2} \end{cases}$$

$$(29)$$

Example 7.3. Consider the query posed over the source ontology of Figure 1: "Return the superclasses of the class Science". The SPARQL syntax of the source query is shown below:

@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.
@PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

```
SELECT ?x
WHERE
{
    src:Science rdfs:subClassOf ?x.
}
```

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (src : Science, rdfs : subClassOf, ?x) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's subject part, the result of the triple pattern's t rewriting by its subject part is provided by invoking the function (29).

```
t = (src : Science, rdfs : subClassOf, ?x)
\mu : src : Science \equiv trg : ComputerScience \sqcup trg : Mathematics
```

The mapping μ is of type $c_s \to c_{t1} \sqcup c_{t2}$. Following the definition of the function (29), two triple patterns t_1 and t_2 are created and the complex mapping μ is decomposed into the mappings μ_1 and μ_2 . The triple patterns t_1 and t_2 contain the classes c_{t1} and c_{t2} on their subject part, respectively. The mapping of the class c_{t1} is provided by μ_1 , while the mapping of the class c_{t2} is provided by μ_2 .

 $t_{1} = (c_{t1}, rdfs : subClassOf, ?x)$ $t_{2} = (c_{t2}, rdfs : subClassOf, ?x)$ $\mu_{1} : c_{t1} \equiv trg : ComputerScience$ $\mu_{2} : c_{t2} \equiv trg : Mathematics$

Thus,

$$\mathcal{S}_c^s(t,\mu) = \mathcal{S}_c^s(t_1,\mu_1) \text{ AND } \mathcal{S}_c^s(t_2,\mu_2)$$

The mappings μ_1 and μ_2 are of type $c_s \to c_t$. Thus, using the parameters defined above, as well as the function (29) for the rewriting of the triple patterns t_1 and t_2 , the initial triple pattern t is rewritten as follows:

$$\begin{split} \mathcal{S}_{c}^{s}(t,\mu) &= \mathcal{S}_{c}^{s}(t_{1},\mu_{1}) \text{ AND } \mathcal{S}_{c}^{s}(t_{2},\mu_{2}) \\ &= (trg:ComputerScience,rdfs:subClassOf,?x) \text{ AND} \\ (trg:Mathematics,rdfs:subClassOf,?x) \end{split}$$

Example 7.4. Consider the query posed over the source ontology of Figure 1: "Return the superclasses of the class NewPublication". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#>.
@PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
SELECT ?x
WHERE
{
    src:NewPublication rdfs:subClassOf ?x.
}
```

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (src : NewPublication, rdfs : subClassOf, ?x) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's subject part, the result of the triple pattern's t rewriting by its subject part is provided by invoking the function (29).

t = (src : NewPublication, rdfs : subClassOf, ?x) $\mu : src : NewPublication \equiv trg : Computing \sqcap trg : NewRelease$

The mapping μ is of type $c_s \to c_{t1} \sqcap c_{t2}$. Following the definition of the function (29), two triple patterns t_1 and t_2 are created and the complex mapping μ is decomposed into the mappings μ_1 and μ_2 . The triple patterns t_1 and t_2 contain the classes c_{t1} and c_{t2} on their subject part, respectively. The mapping of the class c_{t1} is provided by μ_1 , while the mapping of the class c_{t2} is provided by μ_2 .

 $t_{1} = (c_{t1}, rdfs : subClassOf, ?x)$ $t_{2} = (c_{t2}, rdfs : subClassOf, ?x)$ $\mu_{1} : c_{t1} \equiv trg : Computing$ $\mu_{2} : c_{t2} \equiv trg : NewRelease$

Thus,

$$\mathcal{S}_c^o(t,\mu) = \mathcal{S}_c^o(t_1,\mu_1) \text{ UNION } \mathcal{S}_c^o(t_2,\mu_2)$$

The mappings μ_1 and μ_2 are of type $c_s \to c_t$. Thus, using the parameters defined above, as well as the function (29) for the rewriting of the triple patterns t_1 and t_2 , the initial triple pattern t is rewritten as follows:

$$\begin{split} \mathcal{S}_{c}^{o}(t,\mu) &= \mathcal{S}_{c}^{o}(t_{1},\mu_{1}) \text{ UNION } \mathcal{S}_{c}^{o}(t_{2},\mu_{2}) \\ &= (trg:Computing,rdfs:subClassOf,?x) \text{ UNION } \\ (trg:NewRelease,rdfs:subClassOf,?x) \end{split}$$

Example 7.5. Consider the query posed over the source ontology of Figure 1: "Return the classes which are specified to be equivalent to the class Science". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.
@PREFIX owl: <http://www.w3.org/2002/07/owl#>.
SELECT ?x
WHERE
{
    src:Science owl:equivalentClass ?x.
}
```

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (src : Science, owl : equivalentClass, ?x) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's subject part, the result of the triple pattern's t rewriting by its subject part is provided by invoking the function (29).

t = (src: Science, owl: equivalentClass, ?x)

 μ : src: Science \equiv trg: ComputerScience \sqcup trg: Mathematics

The mapping μ is of type $c_s \to c_{t1} \sqcup c_{t2}$. Following the definition of the function (29), two triple patterns t_1 and t_2 are created and the complex mapping μ is decomposed into the mappings μ_1 and μ_2 . The triple patterns t_1 and t_2 contain the classes c_{t1} and c_{t2} on their subject part, respectively. The mapping of the class c_{t1} is provided by μ_1 , while the mapping of the class c_{t2} is provided by μ_2 .

> $t_{1} = (c_{t1}, owl : equivalentClass, ?x)$ $t_{2} = (c_{t2}, owl : equivalentClass, ?x)$ $\mu_{1} : c_{t1} \equiv trg : ComputerScience$ $\mu_{2} : c_{t2} \equiv trg : Mathematics$

Thus,

$$\mathcal{S}_c^s(t,\mu) = \mathcal{S}_c^s(t_1,\mu_1) \text{ AND } \mathcal{S}_c^s(t_2,\mu_2)$$

The mappings μ_1 and μ_2 are of type $c_s \to c_t$. Thus, using the parameters defined above, as well as the function (29) for the rewriting of the triple patterns t_1 and t_2 , the initial triple pattern t is rewritten as follows:

 $\mathcal{S}_c^s(t,\mu) = \mathcal{S}_c^s(t_1,\mu_1) \text{ AND } \mathcal{S}_c^s(t_2,\mu_2)$

 $= (trg: ComputerScience, owl: equivalentClass, ?x) \text{ AND} \\ (trg: Mathematics, owl: equivalentClass, ?x)$

Rewriting based on object property mapping. Let op_s be an object property from the source ontology which is mapped to an object property expression from the target ontology. Having a Schema Triple Pattern $t = (op_s, predicate, object)$ with op_s in its subject part, an RDF/RDFS/OWL property in its predicate and anything in its object part, we can rewrite it by its subject part, using a predefined mapping μ and the function (30).

$$S_{op}^{s}(t,\mu) = \begin{cases} (op_{t}, predicate, object) & \text{if } \mu : op_{s} \to op_{t} \\ S_{op}^{s}(t_{1},\mu_{1}) \text{ UNION } S_{op}^{s}(t_{2},\mu_{2}) & \text{if } \mu : op_{s} \to op_{t1} \sqcap op_{t2} \text{ and} \\ predicate = rdfs : subPropertyOf, \\ where t_{1} = (op_{t1}, predicate, object), \\ \mu_{1} : op_{t1} \equiv OPE_{t1}, \\ \text{and } t_{2} = (op_{t2}, predicate, object), \\ \mu_{2} : op_{t2} \equiv OPE_{t2} \end{cases} \\ S_{op}^{s}(t_{1},\mu_{1}) \text{ AND } S_{op}^{s}(t_{2},\mu_{2}) & \text{if } \mu : op_{s} \to op_{t1}\{\sqcap \mid \sqcup\}op_{t2} \text{ and} \\ predicate = rdfs : subPropertyOf, \\ \text{where } t_{1} = (op_{t1}, predicate, object), \\ \mu_{1} : op_{s} \to op_{t1} \sqcup op_{t2} \text{ and} \\ predicate = rdfs : subPropertyOf, \\ \text{where } t_{1} = (op_{t1}, predicate, object), \\ \mu_{1} : op_{t1} \equiv OPE_{t1}, \\ \text{and } t_{2} = (op_{t2}, predicate, object), \\ \mu_{2} : op_{t2} \equiv OPE_{t2} \end{cases}$$

$$(30)$$

Rewriting based on datatype property mapping. Let dp_s be a datatype property from the source ontology which is mapped to a datatype property expression from the target ontology. Having a Schema Triple Pattern $t = (dp_s, predicate, object)$ with dp_s in its subject part, an RDF/RDFS/OWL property in its predicate and anything in its object part, we can rewrite it by its subject part, using a predefined mapping μ and the function (31).

$$S_{dp}^{s}(t,\mu) = \begin{cases} (dp_{t}, predicate, object) & \text{if } \mu : dp_{s} \to dp_{t} \\ S_{dp}^{s}(t_{1},\mu_{1}) \text{ UNION } S_{dp}^{s}(t_{2},\mu_{2}) & \text{if } \mu : dp_{s} \to dp_{t1} \sqcap dp_{t2} \text{ and} \\ predicate = rdfs : subPropertyOf, \\ \text{where } t_{1} = (dp_{t1}, predicate, object), \\ \mu_{1} : dp_{t1} \equiv DPE_{t1}, \\ \text{and } t_{2} = (dp_{t2}, predicate, object), \\ \mu_{2} : dp_{t2} \equiv DPE_{t2} \end{cases} \\ S_{dp}^{s}(t_{1},\mu_{1}) \text{ AND } S_{dp}^{s}(t_{2},\mu_{2}) & \text{if } \mu : dp_{s} \to dp_{t1} \{\sqcap \mid \sqcup \} dp_{t2} \text{ and} \\ predicate \in SSP'_{p}, \\ \text{or if } \mu : dp_{s} \to dp_{t1} \sqcup dp_{t2} \text{ and} \\ predicate = rdfs : subPropertyOf, \\ \text{where } t_{1} = (dp_{t1}, predicate, object), \\ \mu_{1} : dp_{t} \equiv DPE_{t1}, \\ \text{and } t_{2} = (dp_{t2}, predicate, object), \\ \mu_{2} : dp_{t2} \equiv DPE_{t2} \end{cases}$$

$$(31)$$

The functions (30) and (31) are used similarly with the function (29), which performs triple pattern rewriting by subject part, based on a class mapping.

Rewriting based on individual mapping. Let i_s be an individual from the source ontology which is mapped to an individual i_t from the target ontology. Having a Schema Triple Pattern $t = (i_s, predicate, object)$ with i_s in its subject part, an RDF/RDFS/OWL property in its predicate and anything in its object part, we can rewrite it by its subject part, using a predefined mapping μ and the function (32).

$$S_i^s(t,\mu) = (i_t, predicate, object) \quad \text{if } \mu: \ i_s = i_t \tag{32}$$

In Lemma 7.1 we summarize the functions presented in this section, which are used for the rewriting of a Schema Triple Pattern based on a mapping for the triple pattern's subject part.

Lemma 7.1. Having a Schema Triple Pattern t and a predefined mapping μ for its subject part, we can rewrite it by its subject, by invoking the function (33).

$$S_*^s(t,\mu) = \begin{cases} S_i^s(t,\mu) & \text{if } t = (i_s, predicate, object) \\ S_c^s(t,\mu) & \text{if } t = (c_s, predicate, object) \\ S_{op}^s(t,\mu) & \text{if } t = (op_s, predicate, object) \\ S_{dp}^s(t,\mu) & \text{if } t = (dp_s, predicate, object) \end{cases}$$
(33)

7.2 Rewriting by triple pattern's object part

Unlike the Data Triple Patterns, a property can appear in the object part of a Schema Triple Pattern.

Rewriting based on class mapping. Let c_s be a class from the source ontology which is mapped to a class expression from the target ontology. Having a Schema Triple Pattern $t = (subject, predicate, c_s)$ with c_s in its object part, an RDF/RDFS/OWL property in its predicate and anything in its subject part, we can rewrite it by its object part, using a predefined mapping μ and the function (34).

$$S_{c}^{o}(t,\mu) = \begin{cases} (subject, predicate, c_{t}) & \text{if } \mu : c_{s} \to c_{t} \\ S_{c}^{o}(t_{1},\mu_{1}) \text{ UNION } S_{c}^{o}(t_{2},\mu_{2}) & \text{if } \mu : c_{s} \to c_{t1} \sqcup c_{t2} \text{ and} \\ predicate = rdfs : subClassOf, \\ \text{where } t_{1} = (subject, predicate, c_{t1}), \\ \mu_{1} : c_{t1} \equiv CE_{t1}, \\ \text{and } t_{2} = (subject, predicate, c_{t2}), \\ \mu_{2} : c_{t2} \equiv CE_{t2} \end{cases}$$

$$S_{c}^{o}(t_{1},\mu_{1}) \text{ AND } S_{c}^{o}(t_{2},\mu_{2}) & \text{if } \mu : c_{s} \to c_{t1} \{ \Box \mid \sqcup \} c_{t2} \text{ and} \\ predicate \in SSP_{c}', \\ \text{or if } \mu : c_{s} \to c_{t1} \sqcap c_{t2} \text{ and} \\ predicate = rdfs : subClassOf, \\ \text{where } t_{1} = (subject, predicate, c_{t1}), \\ \mu_{1} : c_{t1} \equiv CE_{t1}, \\ \text{and } t_{2} = (subject, predicate, c_{t2}), \\ \mu_{2} : c_{t2} \equiv CE_{t2} \end{cases}$$

$$(34)$$

Example 7.6. Consider the query posed over the source ontology of Figure 1: "Return the subclasses of the class Science". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#>.
@PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
```

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (?x, rdfs : subClassOf, src : Science) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's object part, the result of the triple pattern's t rewriting by its object part is provided by invoking the function (34).

$$t = (?x, rdfs : subClassOf, src : Science)$$

 $\mu : src : Science \equiv trg : ComputerScience \sqcup trg : Mathematics$

The mapping μ is of type $c_s \to c_{t1} \sqcup c_{t2}$. Following the definition of the function (34), two triple patterns t_1 and t_2 are created and the complex mapping μ is decomposed into the mappings μ_1 and μ_2 . The triple patterns t_1 and t_2 contain the classes c_{t1} and c_{t2} on their object part, respectively. The mapping of the class c_{t1} is provided by μ_1 , while the mapping of the class c_{t2} is provided by μ_2 .

$$t_1 = (?x, rdfs : subClassOf, c_{t1})$$
$$t_2 = (?x, rdfs : subClassOf, c_{t2})$$

 $\mu_1: c_{t1} \equiv trg: ComputerScience$

 $\mu_2: c_{t2} \equiv trg: Mathematics$

Thus,

$$\mathcal{S}_c^o(t,\mu) = \mathcal{S}_c^o(t_1,\mu_1) \text{ UNION } \mathcal{S}_c^o(t_2,\mu_2)$$

The mappings μ_1 and μ_2 are of type $c_s \to c_t$. Thus, using the parameters defined above, as well as the function (34) for the rewriting of the triple patterns t_1 and t_2 , the initial triple pattern t is rewritten as follows:

$$\begin{aligned} \mathcal{S}_{c}^{o}(t,\mu) &= \mathcal{S}_{c}^{o}(t_{1},\mu_{1}) \text{ UNION } \mathcal{S}_{c}^{o}(t_{2},\mu_{2}) \\ &= (?x,rdfs:subClassOf,trg:ComputerScience) \text{ UNION} \\ (?x,rdfs:subClassOf,trg:Mathematics) \end{aligned}$$

Example 7.7. Consider the query posed over the source ontology of Figure 1: "Return the subclasses of the class NewPublication". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#>.
@PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
```

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (?x, rdfs : subClassOf, src : NewPublication) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's object part, the result of the triple pattern's t rewriting by its object part is provided by invoking the function (34).

t = (?x, rdfs : subClassOf, src : NewPublication) $\mu : src : NewPublication \equiv trg : Computing \ \sqcap \ trg : NewRelease$

The mapping μ is of type $c_s \to c_{t1} \sqcap c_{t2}$. Following the definition of the function (34), two triple patterns t_1 and t_2 are created and the complex mapping μ is decomposed into the mappings μ_1 and μ_2 . The triple patterns t_1 and t_2 contain the classes c_{t1} and c_{t2} on their object part, respectively. The mapping of the class c_{t1} is provided by μ_1 , while the mapping of the class c_{t2} is provided by μ_2 .

$$t_1 = (?x, rdfs : subClassOf, c_{t1})$$

$$t_2 = (?x, rdfs : subClassOf, c_{t2})$$

 $\mu_1: c_{t1} \equiv trg: Computing$ $\mu_2: c_{t2} \equiv trg: NewRelease$

Thus,

$$\mathcal{S}_c^o(t,\mu) = \mathcal{S}_c^o(t_1,\mu_1) \text{ AND } \mathcal{S}_c^o(t_2,\mu_2)$$

The mappings μ_1 and μ_2 are of type $c_s \to c_t$. Thus, using the parameters defined above, as well as the function (34) for the rewriting of the triple patterns t_1 and t_2 , the initial triple pattern t is rewritten as follows:

$$\begin{aligned} \mathcal{S}_{c}^{o}(t,\mu) &= \mathcal{S}_{c}^{o}(t_{1},\mu_{1}) \text{ AND } \mathcal{S}_{c}^{o}(t_{2},\mu_{2}) \\ &= (?x,rdfs:subClassOf,trg:Computing) \text{ AND} \\ (?x,rdfs:subClassOf,trg:NewRelease) \end{aligned}$$

Example 7.8. Consider the query posed over the source ontology of Figure 1: "Return the classes which have been specified to be disjoint with the class Science". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.owl-ontologies.com/SourceOntology.owl#>.
@PREFIX owl: <http://www.w3.org/2002/07/owl#>.
```

```
SELECT ?x
WHERE
{
     ?x owl:disjointWith src:Science.
}
```

In order to rewrite the SPARQL query for posing it over the target ontology of Figure 1, we have to rewrite the triple pattern t = (?x, owl : disjointWith, src : Science) by its subject, predicate and object parts. Taking into consideration a mapping μ of the triple pattern's object part, the result of the triple pattern's t rewriting by its object part is provided by invoking the function (34).

t = (?x, owl : disjointWith, src : Science)

 μ : src: Science \equiv trg: ComputerScience \sqcup trg: Mathematics

The mapping μ is of type $c_s \to c_{t1} \sqcup c_{t2}$. Following the definition of the function (34), two triple patterns t_1 and t_2 are created and the complex mapping μ is decomposed into the mappings μ_1 and μ_2 . The triple patterns t_1 and t_2 contain the classes c_{t1} and c_{t2} on their object part, respectively. The mapping of the class c_{t1} is provided by μ_1 , while the mapping of the class c_{t2} is provided by μ_2 .

 $t_1 = (?x, owl : disjointWith, c_{t1})$ $t_2 = (?x, owl : disjointWith, c_{t2})$

 $\mu_1: c_{t1} \equiv trg: ComputerScience$

 $\mu_2: c_{t2} \equiv trg: Mathematics$

Thus,

$$\mathcal{S}_c^o(t,\mu) = \mathcal{S}_c^o(t_1,\mu_1) \text{ AND } \mathcal{S}_c^o(t_2,\mu_2)$$

The mappings μ_1 and μ_2 are of type $c_s \to c_t$. Thus, using the parameters defined above, as well as the function (34) for the rewriting of the triple patterns t_1 and t_2 , the initial triple pattern t is rewritten as follows:

$$\begin{split} \mathcal{S}_{c}^{o}(t,\mu) &= \mathcal{S}_{c}^{o}(t_{1},\mu_{1}) \text{ AND } \mathcal{S}_{c}^{o}(t_{2},\mu_{2}) \\ &= (?x,owl:disjointWith,trg:ComputerScience) \text{ AND} \\ (?x,owl:disjointWith,trg:Mathematics) \end{split}$$

Rewriting based on object property mapping. Let op_s be an object property from the source ontology which is mapped to an object property expression from the target ontology. Having a Schema Triple Pattern $t = (subject, predicate, op_s)$ with op_s in its object part, an RDF/RDFS/OWL property in its predicate and anything in its subject part, we can rewrite it by its object part, using a predefined mapping μ and the function (35).

$$S_{op}^{o}(t,\mu) = \begin{cases} (subject, predicate, op_{t}) & \text{if } \mu : op_{s} \to op_{t} \\ S_{op}^{o}(t_{1},\mu_{1}) \text{ UNION } S_{op}^{o}(t_{2},\mu_{2}) & \text{if } \mu : op_{s} \to op_{t1} \sqcup op_{t2} \text{ and} \\ predicate = rdfs : subPropertyOf, \\ \text{where } t_{1} = (subject, predicate, op_{t1}), \\ \mu_{1} : op_{t1} \equiv OPE_{t1}, \\ \text{and } t_{2} = (subject, predicate, op_{t2}), \\ \mu_{2} : op_{t2} \equiv OPE_{t2} \end{cases} \\ S_{op}^{o}(t_{1},\mu_{1}) \text{ AND } S_{op}^{o}(t_{2},\mu_{2}) & \text{if } \mu : op_{s} \to op_{t1} \{\Box \mid \sqcup\} op_{t2} \text{ and} \\ predicate \in SSP'_{p}, \\ \text{or if } \mu : op_{s} \to op_{t1} \sqcap op_{t2} \text{ and} \\ predicate = rdfs : subPropertyOf, \\ \text{where } t_{1} = (subject, predicate, op_{t1}), \\ \mu_{1} : op_{t1} \equiv OPE_{t1}, \\ \text{and } t_{2} = (subject, predicate, op_{t2}), \\ \mu_{2} : op_{t2} \equiv OPE_{t2} \end{cases}$$

$$(35)$$

Rewriting based on datatype property mapping. Let dp_s be a datatype property from the source ontology which is mapped to a datatype property expression from the target ontology. Having a Schema Triple Pattern $t = (subject, predicate, dp_s)$ with dp_s in its object part, an RDF/RDFS/OWL property in its predicate and anything in its subject part, we can rewrite it by its object part, using a predefined mapping μ and the function (36).

$$S_{dp}^{o}(t,\mu) = \begin{cases} (subject, predicate, dp_t) & \text{if } \mu : dp_s \to dp_t \\ S_{dp}^{o}(t_1,\mu_1) \text{ UNION } S_{dp}^{o}(t_2,\mu_2) & \text{if } \mu : dp_s \to dp_{t1} \sqcup dp_{t2} \text{ and} \\ predicate = rdfs : subPropertyOf, \\ \text{where } t_1 = (subject, predicate, dp_{t1}), \\ \mu_1 : dp_{t1} \equiv DPE_{t1}, \\ \text{and } t_2 = (subject, predicate, dp_{t2}), \\ \mu_2 : dp_{t2} \equiv DPE_{t2} \end{cases} \\ S_{dp}^{o}(t_1,\mu_1) \text{ AND } S_{dp}^{o}(t_2,\mu_2) & \text{if } \mu : dp_s \to dp_{t1} \{ \Box \mid \sqcup \} dp_{t2} \text{ and} \\ predicate \in SSP'_p, \\ \text{or if } \mu : dp_s \to dp_{t1} \sqcap dp_{t2} \text{ and} \\ predicate = rdfs : subPropertyOf, \\ \text{where } t_1 = (subject, predicate, dp_{t1}), \\ \mu_1 : dp_{t1} \equiv DPE_{t1}, \\ \text{and } t_2 = (subject, predicate, dp_{t1}), \\ \mu_1 : dp_{t1} \equiv DPE_{t1}, \\ \text{and } t_2 = (subject, predicate, dp_{t2}), \\ \mu_2 : dp_{t2} \equiv DPE_{t2} \end{cases}$$

$$(36)$$

The functions (35) and (36) are used similarly with the function (34), which performs triple pattern rewriting by object part, based on a class mapping.

Rewriting based on individual mapping. Let i_s be an individual from the source ontology which is mapped to an individual i_t from the target ontology. Having a Schema Triple Pattern $t = (subject, predicate, i_s)$ with i_s in its object part, an RDF/RDFS/OWL property in its predicate and anything in its subject part, we can rewrite it by its object part, using a predefined mapping μ and the function (37).

$$\mathcal{S}_{i}^{o}(t,\mu) = (subject, predicate, i_{t}) \quad \text{if } \mu: \ i_{s} = i_{t}$$

$$(37)$$

In Lemma 7.2 we summarize the functions presented in this section, which are used for the rewriting of a Schema Triple Pattern based on a mapping for the triple pattern's object part.

Lemma 7.2. Having a Schema Triple Pattern t and a predefined mapping μ for its object part, we can rewrite it by its object, by invoking the function (38).

$$S^{o}_{*}(t,\mu) = \begin{cases} S^{o}_{i}(t,\mu) & \text{if } t = (subject, predicate, i_{s}) \\ S^{o}_{c}(t,\mu) & \text{if } t = (subject, predicate, c_{s}) \\ S^{o}_{op}(t,\mu) & \text{if } t = (subject, predicate, op_{s}) \\ S^{o}_{dp}(t,\mu) & \text{if } t = (subject, predicate, dp_{s}) \end{cases}$$
(38)

8 Graph pattern rewriting

In this section, we present the algorithms performing graph pattern rewriting, based on a set of predefined mappings. Algorithm 1 takes as input a SPARQL query's graph pattern GP_{in} ,

as well as a set of mappings \mathcal{M} . Firstly, it rewrites every FILTER expression inside the graph pattern. Afterwards, it rewrites the graph pattern triple pattern by triple pattern, using the mappings of the triple patterns' predicate parts. Similarly, it rewrites the resulted graph pattern, using the mappings of the triple patterns' object parts and then using the mappings of their subject parts. Finally, after removing any unnecessary brackets the resulted graph pattern is ready to replace the initial query's graph pattern (posed over the source ontology) in order for the resulted query to be posed over the target ontology.

Algorithm 1 Graph Pattern Rewriting (GP_{in} : input graph pattern, \mathcal{M} : mapping set)

1: let $\overline{GP_{out}}$ be the rewriting result of $\overline{GP_{in}}$

- 2: $GP_{out} \leftarrow GP_{in}$ after replacing any user defined IRIs (class, property, individual) inside FILTER expressions using the 1:1 cardinality mappings of \mathcal{M}
- 3: $GP_{out} \leftarrow$ Triple Pattern Rewriting $(GP_{out}, \mathcal{M}, predicate)$
- 4: $GP_{out} \leftarrow$ Triple Pattern Rewriting $(GP_{out}, \mathcal{M}, object)$
- 5: $GP_{out} \leftarrow$ Triple Pattern Rewriting $(GP_{out}, \mathcal{M}, subject)$
- 6: $GP_{out} \leftarrow GP_{out}$ after removing any unnecessary brackets
- 7: return GP_{out}

Algorithm 2 rewrites each triple pattern of a graph pattern, using the mappings of the triple pattern's subject, predicate or object parts. It takes as input a SPARQL graph pattern GP_{in} , a set of mappings \mathcal{M} , as well as the part of the triple pattern x (subject, predicate or object) which will be used for the rewriting. The initial graph pattern's operators (AND, UNION, OPTIONAL, FILTER) remain the same during the rewriting process, while SPARQL variables, literal constants and RDF/RDFS/OWL IRIs which may appear in the subject, predicate or object of a triple pattern do not affect the rewriting procedure. This means that the SPARQL variables of the initial query also appear in the rewritten query.

Example 8.1. Consider the query posed over the source ontology of Figure 1: "Return the ids of the products named Linux". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
```

```
SELECT ?id
WHERE
{
     ?x rdf:type src:Product.
     ?x src:id ?id.
     ?x src:name ?name.
     FILTER(?name="Linux")
}
```

In order to rewrite the SPARQL query's graph pattern, we have to use Algorithm 1, as well as a set of predefined mappings. Let the available predefined mappings be the mappings μ_1 , μ_2 and μ_3 presented below. The inputs of Algorithm 1 are GP_{in} and \mathcal{M} .

$$GP_{in} = (?x, rdf : type, src : Product) \text{ AND } (?x, src : id, ?id) \text{ AND } (?x, src : name, ?name) \text{ FILTER}(?name = "Linux")$$

Algorithm 2 Triple Pattern Rewriting (GP_{in}) : input graph pattern, \mathcal{M} : mapping set, x: triple pattern part)

Require: $x \in \{subject, predicate, object\}$ 1: let subject(t) be the subject of a triple pattern t 2: let predicate(t) be the predicate of a triple pattern t 3: let object(t) be the object of a triple pattern t 4: let $relax(\mu)$ be the relaxed mapping used for Schema Triple Pattern rewriting 5: let GP_{out} be the rewriting result of GP_{in} for each basic graph pattern BGP in GP_{in} do 6: let GP_{temp1} be the rewriting result of BGP7: 8: for each triple pattern t in BGP do 9: let GP_{temp2} be the rewriting result of t if x(t) is a variable, or a literal constant, or an RDF/RDFS/OWL property then 10: 11: $GP_{temp2} \leftarrow t$ 12:else if $t \in DTP$ then {in case that t is a Data Triple Pattern} 13: if x = subject then 14:let $\mu_s \in \mathcal{M}$ be the mapping of t's subject 15: $GP_{temp2} \leftarrow \mathcal{D}^s_*(t,\mu_s)$ 16:else if x = predicate then 17: let $\mu_p \in \mathcal{M}$ be the mapping of t's predicate 18: $GP_{temp2} \leftarrow \mathcal{D}^p_*(t,\mu_p)$ 19: 20: else let $\mu_o \in \mathcal{M}$ be the mapping of t's object 21: 22: $GP_{temp2} \leftarrow \mathcal{D}^o_*(t,\mu_o)$ 23: end if else {in case that t is a Schema Triple Pattern} 24:if x = subject then 25:let $\mu_s \in \mathcal{M}$ be the mapping of t's subject 26: $\mu'_s \leftarrow relax(\mu_s)$ 27: 28: $GP_{temp2} \leftarrow \mathcal{S}^s_*(t, \mu'_s)$ else 29:if x = object then 30: let $\mu_o \in \mathcal{M}$ be the mapping of t's object 31: $\mu'_o \leftarrow relax(\mu_o)$ 32: $GP_{temp2} \leftarrow \mathcal{S}^o_*(t, \mu_o)$ 33: end if 34: end if 35: end if 36: end if 37: 38: $GP_{temp1} \leftarrow GP_{temp1}$ after appending GP_{temp2} end for 39: 40: $GP_{temp1} \leftarrow GP_{temp1}$ after applying any filters according to the BGP form $GP_{temp1} \leftarrow \{GP_{temp1}\}$ after applying any brackets in order to form the graph pattern 41: precendence according to the GP_{in} form $GP_{out} \leftarrow GP_{out}$ after appending GP_{temp1} 42: $GP_{out} \leftarrow GP_{out}$ after appending any operators/filters to the rewritten graph pattern 43: according to the GP_{in} form 44: end for 45: return GPout

$$\mathcal{M} = \left\{ \begin{array}{ll} \mu_1 : \ src : Product \ \supseteq \ trg : Textbook, \\ \mu_2 : \ src : id \ \supseteq \ trg : isbn, \\ \mu_3 : \ src : name \ \supseteq \ trg : title \end{array} \right.$$

 GP_{in} contains a FILTER operation which does not affect the rewriting procedure since the filter's expression consists of SPARQL variables and literal constants. Consequently, the algorithm proceeds to the invocation of Algorithm 2 in order to rewrite each triple pattern of GP_{in} , using the mappings of the triple patterns' predicate parts. The input of Algorithm 2 is the initial graph pattern GP_{in} , as well as the set of mappings \mathcal{M} .

The graph pattern GP_{in} , is actually a basic graph pattern since it consists of a triple pattern sequence followed by a FILTER operation. Thus, Algorithm 2 firstly rewrites (in step 3) the basic graph pattern GP_{in} triple pattern by triple pattern, using the mappings of the triple patterns' predicate parts.

The triple pattern $t_1 = (?x, rdf : type, src : Product)$ remains the same after the rewriting process, since its predicate part consists of the RDF property rdf : type. On the contrary, the rewriting result of the triple pattern $t_2 = (?x, src : id, ?id)$, as well as the rewriting result of the triple pattern $t_3 = (?x, src : name, ?name)$ are provided by invoking the function (20).

$$\mathcal{D}^p_*(t_2, \mu_2) = \mathcal{D}^p_{dp}((?x, src: id, ?id), \mu_2)$$
$$= (?x, trg: isbn, ?id)$$

$$\mathcal{D}^p_*(t_3, \mu_3) = \mathcal{D}^p_{dp}((?x, src : name, ?name), \mu_3)$$
$$= (?x, trg : title, ?name)$$

Consequently, the output of Algorithm 2 is a graph pattern, having its triple patterns rewritten by their predicate part and is presented below:

$$GP_p = (?x, rdf : type, src : Product) \text{ AND } (?x, trg : isbn, ?id) \text{ AND } (?x, trg : title, ?name) \text{ FILTER}(?name = "Linux")$$

Similarly, Algorithm 1 uses Algorithm 2 in order to rewrite the triple patterns of GP_p by their object parts. All the triple patterns except of t_1 remain the same, since they contain a SPARQL variable on their object part. The rewriting result of the triple pattern t_1 , is provided by invoking the function (17).

$$\mathcal{D}^{o}_{*}(t_{1},\mu_{1}) = \mathcal{D}^{o}_{c}((?x,rdf:type,src:Product),\mu_{1})$$
$$= (?x,rdf:type,trg:Textbook)$$

Consequently, the output of second invocation of Algorithm 2 is a graph pattern, having its triple patterns rewritten by their object part and is presented below:

$$\begin{array}{ll} GP_o &= (?x, rdf: type, trg: Textbook) \text{ AND } (?x, trg: isbn, ?id) \text{ AND } \\ (?x, trg: title, ?name) \text{ FILTER}(?name = "Linux") \end{array}$$

Finally, Algorithm 1 proceeds to step 5 (Algorithm 2 invocation) in order to rewrite the triple patterns of GP_o by their subject parts. Although, the graph patern remains the same since every triple pattern of GP_o consists of a SPARQL variable in its subject part.

```
GP_s = (?x, rdf : type, trg : Textbook) AND (?x, trg : isbn, ?id) AND (?x, trg : title, ?name) FILTER(?name = "Linux")
```

Taking into consideration the graph pattern GP_s , which is the output of the last invocation of Algorithm 2, the rewritten SPARQL query which will be posed over the target ontology of Figure 1 is presented below:

```
?x trg:isbn ?id.
?x trg:title ?name.
FILTER(?name="Linux")
```

}

Example 8.2. Consider the query posed over the source ontology of Figure 1: "Return the individuals of every class which is specified to be subclass of the class NewPublication". The SPARQL syntax of the source query is shown below:

```
@PREFIX src: <http://www.ontologies.com/SourceOntology.owl#>.
@PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
SELECT ?x
WHERE
{
    ?x rdf:type ?y.
    ?y rdfs:subClassOf src:NewPublication.
}
```

In order to rewrite the SPARQL query's graph pattern, we have to use Algorithm 1, as well as a set of predefined mappings. Let the available predefined mappings be the mapping μ presented below. The inputs of Algorithm 1 are GP_{in} and \mathcal{M} .

 $GP_{in} = (?x, rdf : type, ?y)$ AND (?x, rdfs : subClassOf, src : Science)

 $\mathcal{M} = \{ \mu : src : NewPublication \equiv trg : Computing \sqcap trg : NewRelease \}$

 GP_{in} does not contain any FILTER operations, thus the algorithm proceeds to the invocation of Algorithm 2 in order to rewrite each triple pattern of GP_{in} , using the mappings of the triple patterns' predicate parts. The input of Algorithm 2 is the initial graph pattern GP_{in} , as well as the set of mappings \mathcal{M} .

The graph pattern GP_{in} , is actually a basic graph pattern since it consists of a triple pattern sequence. Thus, Algorithm 2 firstly rewrites (in step 3) the basic graph pattern GP_{in} triple pattern by triple pattern, using the mappings of the triple patterns' predicate parts.

Although, all the triple patterns remain the same, since they contain an RDF/S property on their predicate part. Consequently, the output of Algorithm 2 is presented below:

 $GP_p = (?x, rdf : type, ?y)$ AND (?y, rdfs : subClassOf, src : NewPublication)

Similarly, Algorithm 1 uses Algorithm 2 in order to rewrite the triple patterns of GP_p by their object parts. The rewriting result of the triple pattern $t_2 = (?y, rdfs : subClassOf, src : NewPublication)$, is provided by invoking the function (38), while the triple pattern $t_1 = (?x, rdf : type, ?y)$ remains the same, since it contains a SPARQL variable on its object part.

 $\begin{aligned} \mathcal{S}^{o}_{*}(t_{2},\mu) &= \mathcal{S}^{o}_{c}((?y,rdfs:subClassOf,src:NewPublication),\mu) \\ &= (?y,rdfs:subClassOf,trg:Computing) \text{ AND} \\ (?y,rdfs:subClassOf,trg:NewRelease) \end{aligned}$

Consequently, the output of the second invocation of Algorithm 2 is a graph pattern, having its triple patterns rewritten by their object part and is presented below:

 $\begin{array}{ll} GP_o &= (?x, rdf: type, ?y) \mbox{ AND} \\ & (?y, rdfs: subClassOf, trg: Computing) \mbox{ AND} \\ & (?y, rdfs: subClassOf, trg: NewRelease) \end{array}$

Finally, Algorithm 1 proceeds to step 5 (Algorithm 2 invocation) in order to rewrite the triple patterns of GP_o by their subject parts. Although, the graph patern remains the same since every triple pattern of GP_o contains a SPARQL variable in its subject part.

 $\begin{array}{ll} GP_s &= (?x, rdf: type, ?y) \mbox{ AND} \\ & (?y, rdfs: subClassOf, trg: Computing) \mbox{ AND} \\ & (?y, rdfs: subClassOf, trg: NewRelease) \end{array}$

Taking into consideration the graph pattern GP_s , which is the output of the last invocation of Algorithm 2, the rewritten SPARQL query which will be posed over the target ontology of Figure 1 is presented below:

```
@PREFIX trg: <http://www.ontologies.com/TargetOntology.owl#>.
@PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
```

SELECT ?x WHERE

```
{
    ?x rdf:type ?y.
    ?y rdfs:subClassOf trg:Computing.
    ?y rdfs:subClassOf trg:NewRelease.
}
```

Example 8.3. Consider the query posed over the source ontology of Figure 1: "Return the titles of the pocket-sized scientific books". The SPARQL syntax of the source query is shown below:

In order to rewrite the SPARQL query's graph pattern, we have to use Algorithm 1, as well as a set of predefined mappings. Let the available predefined mappings be the mappings μ_1 , μ_2 and μ_3 presented below. The inputs of Algorithm 1 are GP_{in} and \mathcal{M} .

 $GP_{in} = (?x, src: name, ?name) \text{ AND } (?x, rdf: type, src: Science)$ AND (?x, rdf: type, src: Pocket)

$$\mathcal{M} = \begin{cases} \mu_1 : \ src : name \ \supseteq \ trg : title, \\ \mu_2 : \ src : Science \ \equiv \ trg : ComputerScience \ \sqcup \ trg : Mathematics, \\ \mu_3 : \ src : Pocket \ \equiv \ \forall trg : Textbook.(trg : size \le 14) \end{cases}$$

 GP_{in} does not contain any FILTER operations, thus the algorithm proceeds to the invocation of Algorithm 2 in order to rewrite each triple pattern of GP_{in} , using the mappings of the triple patterns' predicate parts. The input of Algorithm 2 is the initial graph pattern GP_{in} , as well as the set of mappings \mathcal{M} .

The graph pattern GP_{in} , is actually a basic graph pattern since it consists of a triple pattern sequence. Thus, Algorithm 2 firstly rewrites (in step 3) the basic graph pattern GP_{in} triple pattern by triple pattern, using the mappings of the triple patterns' predicate parts.

The triple patterns $t_2 = (?x, rdf : type, src : Science)$ and $t_3 = (?x, rdf : type, src : Pocket)$ remain the same since their predicate parts consist of the RDF property rdf : type. On the contrary, the rewriting result of the triple pattern $t_1 = (?x, src : name, ?name)$, is provided by invoking the function (20).

$$\mathcal{D}^{p}_{*}(t_{1},\mu_{1}) = \mathcal{D}^{p}_{dp}((?x,src:name,?name),\mu_{1})$$
$$= (?x,trg:title,?name)$$

Consequently, the output of Algorithm 2 is a graph pattern, having its triple patterns rewritten by their predicate part and is presented below:

$$GP_p = (?x, trg: title, ?name) \text{ AND } (?x, rdf: type, src: Science)$$

AND $(?x, rdf: type, src: Pocket)$

Similarly, Algorithm 1 uses Algorithm 2 in order to rewrite the triple patterns of GP_p by their object parts. All the triple patterns except of t_2 and t_3 remain the same, since they contain a SPARQL variable on their object part. The rewriting result of the triple patterns t_2 and t_3 , is provided by invoking the function (17).

$$\mathcal{D}^{o}_{*}(t_{2},\mu_{2}) = \mathcal{D}^{o}_{c}((?x,rdf:type,src:Science),\mu_{2})$$

$$= (?x,rdf:type,trg:ComputerScience)$$
UNION (?x,rdf:type,trg:Mathematics)
$$\mathcal{D}^{o}_{*}(t_{3},\mu_{3}) = \mathcal{D}^{o}_{c}((?x,rdf:type,src:Pocket),\mu_{3})$$

= (?x, rdf: type, trg: Textbook)AND (?x, trg: size, ?var) FILTER(?var ≤ 14)

Consequently, the output of the second invocation of Algorithm 2 is a graph pattern, having its triple patterns rewritten by their object part and is presented below:

 $\begin{array}{ll} GP_o &= (?x,trg:title,?name) \text{ AND} \\ & ((?x,rdf:type,trg:ComputerScience) \text{ UNION} \\ & (?x,rdf:type,trg:Mathematics)) \text{ AND } (?x,rdf:type,trg:Textbook) \\ & \text{ AND}(?x,trg:size,?var) \text{ FILTER}(?var \leq 14) \end{array}$

Finally, Algorithm 1 proceeds to step 5 (Algorithm 2 invocation) in order to rewrite the triple patterns of GP_o by their subject parts. Although, the graph patern remains the same since every triple pattern of GP_o contains a SPARQL variable in its subject part.

 $\begin{array}{ll} GP_s &= (?x,trg:title,?name) \text{ AND} \\ & \left((?x,rdf:type,trg:ComputerScience) \text{ UNION} \\ & (?x,rdf:type,trg:Mathematics) \right) \text{ AND } (?x,rdf:type,trg:Textbook) \\ & \text{ AND}(?x,trg:size,?var) \text{ FILTER}(?var \leq 14) \end{array}$

Taking into consideration the graph pattern GP_s , which is the output of the last invocation of Algorithm 2, the rewritten SPARQL query which will be posed over the target ontology of Figure 1 is presented below:

```
@PREFIX trg: <http://www.ontologies.com/TargetOntology.owl#>.
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
SELECT ?name
WHERE
```

Example 8.4. Consider the query posed over the source ontology of Figure 1: "Return the titles (at most 10) of the poetry or autobiography books written by Dante". The SPARQL syntax of the source query is shown below:

```
LIMIT 10
```

}

In order to rewrite the SPARQL query's graph pattern, we have to use Algorithm 1, as well as a set of predefined mappings. Let the available predefined mappings be the mappings μ_1 , μ_2 , μ_3 and μ_4 presented below. The inputs of Algorithm 1 are GP_{in} and \mathcal{M} .

```
 \begin{array}{ll} GP_{in} &= \left((?x, rdf: type, src: Poetry) \text{ UNION } (?x, rdf: type, src: Autobiography)\right) \\ &\quad \text{AND } (?x, src: author, ?author) \text{ AND } (?x, src: name, ?name) \\ &\quad \text{FILTER } (regex(?author, "Dante")) \end{array}
```

```
\mathcal{M} = \left\{ \begin{array}{l} \mu_1: \ src: Poetry \ \sqsubseteq \ trg: Literature, \\ \mu_2: \ src: Autobiography \ \equiv \ \forall trg: Biography.(trg: author = trg: topic), \\ \mu_3: \ src: author \ \equiv \ trg: author \ \circ \ trg: name, \\ \mu_4: \ src: name \ \sqsupseteq \ trg: title \end{array} \right\}
```

 GP_{in} contains a FILTER operation which does not affect the rewriting procedure since the filter's expression consists of SPARQL variables, literal constants and built-in functions. Consequently, the algorithm proceeds to the invocation of Algorithm 2 in order to rewrite each triple pattern of GP_{in} , using the mappings of the triple patterns' predicate parts. The input of Algorithm 2 is the initial graph pattern GP_{in} , as well as the set of mappings \mathcal{M} .

Firstly, the Algorithm 2 rewrites (in step 3) every basic graph pattern of GP_{in} triple pattern by triple pattern, using the mappings of the triple patterns' predicate parts.

The triple patterns $t_1 = (?x, rdf : type, src : Poetry)$ and $t_2 = (?x, rdf : type, src : Autobiography)$ remain the same after the rewriting process, since their predicate part consists of the RDF property rdf : type. On the contrary, the rewriting result of the triple pattern $t_3 = (?x, src : author, ?author)$, as well as the rewriting result of the triple pattern $t_4 = (?x, src : name, ?name)$ are provided by invoking the function (20).

$$\mathcal{D}^{p}_{*}(t_{3},\mu_{3}) = \mathcal{D}^{p}_{dp}((?x,src:author,?author),\mu_{3})$$
$$= (?x,trg:author,?var) \text{ AND}$$
$$(?var,trg:name,?author)$$

$$\mathcal{D}^p_*(t_4, \mu_4) = \mathcal{D}^p_{dp}((?x, src: name, ?name), \mu_4)$$
$$= (?x, trg: title, ?name)$$

Consequently, the output of Algorithm 2 is a graph pattern, having its triple patterns rewritten by their predicate part and is presented below:

 $\begin{array}{ll} GP_p &= \left((?x, rdf: type, src: Poetry) \text{ UNION } (?x, rdf: type, src: Autobiography)\right) \\ &\quad \text{AND } (?x, trg: author, ?var_1) \text{ AND } (?var_1, trg: name, ?author) \\ &\quad \text{AND } (?x, trg: title, ?name) \text{ FILTER}(regex(?author, "Dante")) \end{array}$

Similarly, Algorithm 1 uses Algorithm 2 in order to rewrite the triple patterns of GP_p by their object parts. All the triple patterns except of t_1 and t_2 remain the same, since they contain a SPARQL variable on their object part. The rewriting result of the triple patterns t_1 and t_2 , is provided by invoking the function (17).

$$\mathcal{D}^{o}_{*}(t_{1},\mu_{1}) = \mathcal{D}^{o}_{c}((?x,rdf:type,src:Poetry),\mu_{1})$$
$$= (?x,rdf:type,trg:Literature)$$
$$\mathcal{D}^{o}_{*}(t_{2},\mu_{2}) = \mathcal{D}^{o}_{c}((?x,rdf:type,src:Autobiography),\mu_{2})$$

$$= (?x, rdf : type, trg : Biography) \text{ AND} (?x, trg : author, ?var_2) \text{ AND} (?x, trg : topic, ?var_3) \text{ FILTER}(?var_2 = ?var_3)$$

Consequently, the output of second invocation of Algorithm 2 is a graph pattern, having its triple patterns rewritten by their object part and is presented below:

 $\begin{aligned} GP_o &= \left((?x, rdf: type, trg: Literature) \text{ UNION } ((?x, rdf: type, trg: Biography) \\ \text{AND } (?x, trg: author, ?var_2) \text{ AND } (?x, trg: topic, ?var_3) \\ \text{FILTER}(?var_2 = ?var_3)) \right) \text{ AND } (?x, trg: author, ?var_1) \\ \text{AND } (?var_1, trg: name, ?author) \text{ AND } (?x, trg: title, ?name) \\ \text{FILTER } (regex(?author, "Dante")) \end{aligned}$

Finally, Algorithm 1 proceeds to step 5 (Algorithm 2 invocation) in order to rewrite the triple patterns of GP_o by their subject parts. Although, the graph patern remains the same since every triple pattern of GP_o consists of a SPARQL variable in its subject part.

```
 \begin{split} GP_s &= \left((?x, rdf: type, trg: Literature) \text{ UNION } \left((?x, rdf: type, trg: Biography) \right. \\ &\quad \text{AND } (?x, trg: author, ?var_2) \text{ AND } (?x, trg: topic, ?var_3) \\ &\quad \text{FILTER}(?var_2 = ?var_3)) \right) \text{ AND } (?x, trg: author, ?var_1) \\ &\quad \text{AND } (?var_1, trg: name, ?author) \text{ AND } (?x, trg: title, ?name) \\ &\quad \text{FILTER } (regex(?author, "Dante")) \end{split}
```

Taking into consideration the graph pattern GP_s , which is the output of the last invocation of Algorithm 2, the rewritten SPARQL query which will be posed over the target ontology of Figure 1 is presented below:

```
@PREFIX trg: <http://www.ontologies.com/TargetOntology.owl#>.
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
```

```
SELECT ?name
WHERE
{
    ł
       {?x rdf:type trg:Literature}
       UNION
       {?x rdf:type trg:Biography.
        ?x trg:author ?var_2.
        ?x trg:name ?var_3.
        FILTER(?var_2 = ?var_3)}
    }
    ?x trg:author ?var_1.
    ?var_1 trg:name ?author.
    ?x trg:title ?name.
    FILTER regex(?author, "Dante")
}
LIMIT 10
```

9 Implementation

The SPARQL query rewriting methodology presented in this paper has been implemented as part of a Semantic Query Mediation Prototype Infrastructure developed in the TUC-MUSIC

Lab. The system has been implemented using Java 2SE as a software platform, and the Jena Software framework for SPARQL query parsing. The architecture of this infrastructure is shown in Figure 14 where many of the Mediator's implementation details are not presented for simplicity reasons.

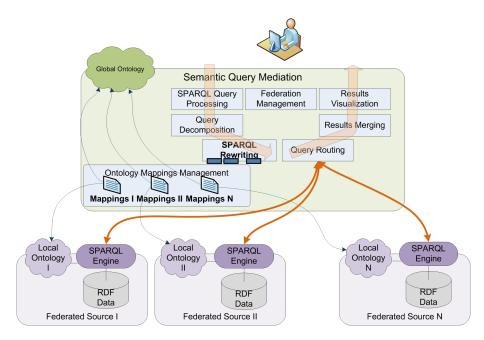


Figure 14: System reference architecture.

For each federated source, a dedicated Query Rewriting component is dynamically created by a Query Rewriting Factory. Such a component is able to rewrite an input SPARQL query based on some predefined mappings. As a representation language for the mappings between two overlapping ontologies we use the language discussed in Section 3.4.

During the system's start-up each component is initialized with the mappings between the mediator's global ontology and the (local) ontology used in the federated source for which this component is responsible.

At run time, when a SPARQL query is posed to the Mediator, it is processed, decomposed, and rewritten for each federated source by the corresponding query rewriting component. Afterwards, the rewritten queries are submitted (routed) to the federated sites for local evaluation. Finally, the returned results from the local sources (to which queries have been routed to) are merged, and optionally visualized (taking into account which part of the initial SPARQL query was answered by each resource) for presentation to the end users.

10 Conclusion

The web of data is heterogeneous, distributed and highly structured. Querying mechanisms in this environment have to overcome the problems arising from the heterogeneous and distributed nature of data, and they have to allow the expression of complex and structured queries. The ontology mappings and SPARQL query mediation presented in this paper aim to satisfy those requirements in the Semantic Web environment. The mediator uses mappings between the global OWL ontology of the mediator and the local ontologies of the federated knowledge bases. SPARQL queries of end users and applications which are posed over the mediator, are decomposed and rewritten in order to be submitted to the federated sources. The rewritten SPARQL queries are locally evaluated and the results are returned to the mediator. Two aspects of this system were discussed in this paper:

- A formal model for describing executable ontology mappings (i.e. mappings which can be used in SPARQL query rewriting) that satisfy real-world requirements and can be exploited in query rewriting. We have presented a mapping model that allows the definition of a rich set of ontology mappings and we have shown real-world examples of its functionality.
- A complete set of SPARQL query rewriting functions and algorithms that allow SPARQL queries which are expressed based on the mediator's global ontology to be rewritten in terms of the local ontologies. These functions are semantics preserving (i.e. preserve the mapping semantics).

Our current research focuses on evaluating the system performance, exploiting advanced reasoning techniques during the query rewriting, and developing methodologies for the optimization of the query mediation process. Moreover, this work is going to be integrated with our XS2OWL [41] and SPARQL2XQuery [6] frameworks, in order to allow access to heterogeneous web repositories.

References

- Akahani, J., k. Hiramatsu, Satoh, T.: Approximate query reformulation for multiple ontologies in the semantic web. Tech. rep. (2003)
- [2] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
- [3] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference. Tech. rep., W3C (2004), http://www.w3.org/TR/owl-ref/
- Benslimane, S.M., Merazi, A., Malki, M., Bensaber., D.A.: Ontology mapping for querying heterogeneous information sources. INFOCOMP Journal of Computer Science 7(3), 52–59 (2008)
- [5] Bernstein, A., Kiefer, C., Stocker, M.: Optarq: A sparql optimization approach based on triple pattern selectivity estimation. Tech. rep., University of Zurich (2007)
- Bikakis, N., Gioldasis, N., Tsinaraki, C., Christodoulakis, S.: Querying xml data with sparql. In: DEXA. Lecture Notes in Computer Science, vol. 5690, pp. 372–381. Springer (2009)
- [7] Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: C-owl: Contextualizing ontologies. In: International Semantic Web Conference. Lecture Notes in Computer Science, vol. 2870, pp. 164–179. Springer (2003)

- [8] Chebotko, A., Lu, S., Fotouhi, F.: Semantics preserving sparql-to-sql translation. Data Knowl. Eng. 68(10), 973–1000 (2009)
- [9] Choi, N., Song, I.Y., Han, H.: A survey on ontology mapping. SIGMOD Record 35(3), 34–41 (2006)
- [10] Correndo, G., Salvadores, M., Millard, I., Glaser, H., Shadbolt, N.: Sparql query rewriting for implementing data integration over linked data. In: 1st International Workshop on Data Semantics (DataSem 2010) (2010)
- [11] Doerr, M., Ore, C.E., Stead, S.: The CIDOC conceptual reference model A new standard for knowledge sharing. In: 26th International Conference on Conceptual Modeling
 - ER 2007. CRPIT, vol. 83, pp. 51–56 (2007)
- [12] Ehrig, M., Staab, S.: Qom quick ontology mapping. In: International Semantic Web Conference. Lecture Notes in Computer Science, vol. 3298, pp. 683–697. Springer (2004)
- [13] Elliott, B., Cheng, E., Thomas-Ogbuji, C., Ozsoyoglu, Z.M.: A complete translation from sparql into efficient sql. In: IDEAS '09: Proceedings of the 2009 International Database Engineering & Applications Symposium. pp. 31–42. ACM (2009)
- [14] Euzenat, J.: An api for ontology alignment. In: International Semantic Web Conference. Lecture Notes in Computer Science, vol. 3298, pp. 698–712. Springer (2004)
- [15] Euzenat, J., Polleres, A., Scharffe, F.: Processing ontology alignments with sparql. In: CISIS. pp. 913–917 (2008)
- [16] Euzenat, J., Scharffe, F., Zimmermann, A.: Expressive alignment language and implementation. deliverable 2.2.10, Knowledge Web NoE (2007), ftp://ftp.inrialpes.fr/ pub/exmo/reports/kweb-2210.pdf
- [17] Euzenat, J., Shvaiko, P.: Ontology Matching. Springer, Heidelberg (2007)
- [18] Euzenat, J., et al.: D2.2.3: State of the art on ontology alignment (2004)
- [19] Groppe, J., Groppe, S., Kolbaum, J.: Optimization of sparql by using coresparql. In: ICEIS (1). pp. 107–112 (2009)
- [20] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: Swrl: A semantic web rule language combining owl and ruleml. W3c member submission, World Wide Web Consortium (2004), http://www.w3.org/Submission/SWRL
- [21] Jain, P., Yeh, P.Z., Verma, K., Henson, C.A., Sheth, A.P.: Sparql query re-writing using partonomy based transformation rules. In: GeoS '09: Proceedings of the 3rd International Conference on GeoSpatial Semantics. pp. 140–158. Springer-Verlag (2009)
- [22] Jing, Y., Jeong, D., Baik, D.K.: Sparql graph pattern rewriting for owl-dl inference queries. Knowl. Inf. Syst. 20(2), 243–262 (2009)
- [23] Kalfoglou, Y., Schorlemmer, W.M.: Ontology mapping: The state of the art. In: Semantic Interoperability and Integration. Dagstuhl Seminar Proceedings, vol. 04391. IBFI, Schloss Dagstuhl, Germany (2005)
- [24] Maedche, A., Motik, B., Silva, N., Volz, R.: Mafra a mapping framework for distributed ontologies. In: EKAW. Lecture Notes in Computer Science, vol. 2473, pp. 235–250. Springer (2002)

- [25] Makris, K., Bikakis, N., Gioldasis, N., Tsinaraki, C., Christodoulakis, S.: Towards a mediator based on owl and sparql. In: WSKS (1). Lecture Notes in Computer Science, vol. 5736, pp. 326–335 (2009)
- [26] Manola, F., Miller, E.: RDF primer. World Wide Web Consortium, Recommendation (2004), http://www.w3.org/TR/2004/REC-rdf-primer-20040210
- [27] Needleman, M.H.: Dublin core metadata element set 24(3-4), 131–135 (1998)
- [28] Noy, N.F.: Semantic integration: a survey of ontology-based approaches. SIGMOD Record (ACM Special Interest Group on Management of Data) 33(4), 65–70 (2004)
- [29] Noy, N.F., Musen, M.A.: PROMPT: Algorithm and tool for automated ontology merging and alignment. In: Proc. of AAAI/IAAI-2000: 450-455 (2000)
- [30] Pan, R., Ding, Z., Yu, Y., Peng, Y.: A bayesian network approach to ontology mapping. In: International Semantic Web Conference. Lecture Notes in Computer Science, vol. 3729, pp. 563–577. Springer (2005)
- [31] Pérez, J., Arenas, M., Gutierrez, C.: Semantics and complexity of sparql. ACM Trans. Database Syst. 34(3) (2009)
- [32] Polleres, A., Scharffe, F., Schindlauer, R.: Sparql++ for mapping between rdf vocabularies. In: OTM Conferences (1). Lecture Notes in Computer Science, vol. 4803, pp. 878–896. Springer (2007)
- [33] Prud'hommeaux, E., Seaborne, A.: Sparql query language for rdf. Tech. rep., W3C (2008), http://www.w3.org/TR/rdf-sparql-query/
- [34] Quilitz, B., Leser, U.: Querying distributed rdf data sources with sparql. In: ESWC. Lecture Notes in Computer Science, vol. 5021, pp. 524–538. Springer (2008)
- [35] Scharffe, F.: Correspondence Patterns Representation. Ph.D. thesis, University of Innsbruck (2009), http://www.scharffe.fr/pub/phd-thesis/manuscript.pdf
- [36] Scharffe, F., de Bruijn, J.: A language to specify mappings between ontologies. In: SITIS. pp. 267–271 (2005)
- [37] Schenk, S., Staab, S.: Networked graphs: a declarative mechanism for sparql rules, sparql views and rdf data integration on the web. In: WWW. pp. 585–594. ACM (2008)
- [38] Schmidt, M., Meier, M., Lausen, G.: Foundations of sparql query optimization. CoRR abs/0812.3788 (2008)
- [39] Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches pp. 146–171 (2005)
- [40] Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: Sparql basic graph pattern optimization using selectivity estimation. In: WWW '08: Proceeding of the 17th international conference on World Wide Web. pp. 595–604. ACM (2008)
- [41] Tsinaraki, C., Christodoulakis, S.: Interoperability of xml schema applications with owl domain knowledge and semantic web tools. In: ODBASE. LNCS, vol. 4803. Springer (2007)

- [42] Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S.: Ontology-based integration of information - a survey of existing approaches. In: Workshop on Ontologies and Information Sharing. pp. 108–117 (2001)
- [43] Wiederhold, G.: Mediators in the architecture of future information systems. IEEE Computer 25(3), 38–49 (1992)

A Semantics of property relationships

A.1 Equivalence/subsumption between properties in OWL

Lemma A.1. The subsumption between two properties using the RDFS property rdfs: subPropertyOf implies subsumption between their domains, as well as subsumption between their ranges.

Proof. Let p_1 , p_2 be object/datatype properties with domains $domain_{p_1}$, $domain_{p_2}$ and ranges $range_{p_1}$, $range_{p_2}$, respectively. The subsumption between these two properties $(p_1 \sqsubseteq p_2)$ is interpreted in OWL by using the RDF triple $(p_1, rdfs : subPropertyOf, p_2)$. Considering that $\alpha \in domain_{p_1}$ and $b \in range_{p_1}$, each RDF triple of the form (α, p_1, b) implies the RDF triple (α, p_2, b) .

Consequently, for the correspondences between the domains and ranges of the properties p_1 and p_2 , we reach the following conclusions:

- $\forall x, [domain_{p_1}(x) \Rightarrow domain_{p_2}(x)], \text{ thus: } domain_{p_1} \sqsubseteq domain_{p_2}(x)$
- $\forall y, [range_{p_1}(y) \Rightarrow range_{p_2}(y)], \text{ thus: } range_{p_1} \sqsubseteq range_{p_2}$

Lemma A.2. The equivalence between two properties using the OWL property owl : equivalent Property implies equivalence between their domains, as well as equivalence between their ranges. \Box

Proof. Let p_1 , p_2 be object/datatype properties with domains $domain_{p_1}$, $domain_{p_2}$ and ranges $range_{p_1}$, $range_{p_2}$, respectively. The equivalence between these two properties ($p_1 \equiv p_2$) is interpreted in OWL by using the RDF triple ($p_1, owl : equivalentProperty, p_2$), which indirectly implies the following RDF triples:

- 1. $(p_1, rdfs : subPropertyOf, p_2)$
- 2. $(p_2, rdfs : subPropertyOf, p_1)$

Let $\alpha \in domain_{p_1}$, $b \in range_{p_1}$, $c \in domain_{p_2}$ and $d \in range_{p_2}$, the RDF triples above, provide the following implications:

- 1. Each RDF triple of the form (α, p_1, b) implies the RDF triple (α, p_2, b) .
- 2. Each RDF triple of the form (c, p_2, d) implies the RDF triple (c, p_1, d) .

Consequently, for the correspondences between the domains and ranges of the properties p_1 and p_2 , we reach the following conclusions:

- $\forall x, [domain_{p_1}(x) \Leftrightarrow domain_{p_2}(x)], \text{ thus: } domain_{p_1} \equiv domain_{p_2}$
- $\forall y, [range_{p_1}(y) \Leftrightarrow range_{p_2}(y)], \text{ thus: } range_{p_1} \equiv range_{p_2}$

A.2 Equivalence/subsumption between properties in our framework

Among the mapping types defined in Section 3.3, there are correspondences between properties and property expressions. The statements below are directly implied by the semantics presented in Section 3.2.

Lemma A.3. An object property expression is an object property, having its domain and range dependent on the expression's type. \Box

Lemma A.4. A datatype property expression is a datatype property, having its domain and range dependent on the expression's type. \Box

Taking into consideration the OWL equivalence and subsumption semantics between properties (presented in Section A.1), as well as the fact that a property expression is actually a property having a domain and range, we reach the following conclusions which are also adapted in our framework:

- The subsumption (⊑) between a property and a property expression implies subsumption between their domains, as well as subsumption between their ranges.
- The equivalence (≡) between a property and a property expression implies equivalence between their domains, as well as equivalence between their ranges.

B Data triple pattern rewriting proofs

In this appendix we provide the proofs of the lemmas presented in Section 6. In Table 14 we present the notation which is used for these proofs. The majority of the functions and operations presented in Table 14 have been defined in Section 4.2.

In what follows, let DS_s and DS_t be the RDF datasets of a source and a target ontology respectively. Similarly, let DS_m be the RDF dataset which is produced by merging [29] the DS_s and DS_t datasets using a set of mappings \mathcal{M} . Let \mathcal{I} be the interpretation that consists of the non-empty set $\Delta^{\mathcal{I}}$, and contains the classes, the object/datatype properties and the individuals of the RDF dataset DS_m . The interpretation \mathcal{I} consists also of an interpretation function which assigns to every atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, to every atomic role B a binary relation $B^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and to every individual k an element $k^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ (based on [2]).

Notation	Description
ω	A graph pattern solution $\omega: V \to (I \cup B \cup L)$.
$dom(\omega)$	Domain of a graph pattern solution ω (subset
	of V).
$\omega(t)$	The triple obtained by replacing the variables
	in triple pattern t according to a graph pattern
	solution ω (abusing notation).
$\omega^s(t)$	The subject part of the triple obtained by re-
	placing the variables in triple pattern t accord-
	ing to a graph pattern solution ω .
$\omega^p(t)$	The predicate part of the triple obtained by
	replacing the variables in triple pattern t ac-
	cording to a graph pattern solution ω .
$\omega^o(t)$	The object part of the triple obtained by re-
	placing the variables in triple pattern t accord-
	ing to a graph pattern solution ω .
$\omega \models R$	A graph pattern solution ω satisfies a built-in
	condition R .
[[•]]	Graph pattern evaluation function.
var(GP)	The variables of a graph pattern GP .
\bowtie	Graph pattern solution-based join.
\bowtie	Graph pattern solution-based left outer join.
/	Graph pattern solution-based difference.
$\pi_{\{\dots\}}$	Graph pattern solution-based projection.
U	Graph pattern solution-based union.
∩ 	Set intersection.
AND, OPT, UNION, FILTER	
\neg, \lor, \land	Logical not, or, and.
$=,\leq,\geq,<,>$	Inequality/equality operators.

Table 14: The notation which is used for the Data triple pattern rewriting proofs.

Proof Overview. The proofs presented in this section and deal with Data Triple Patterns follow a common approach. We consider mappings containing equivalence relationship (\equiv) and we do not provide the proofs for the other mapping types since the approach is very similar for all types.

Let t be the initial triple pattern, gp' be the rewritten graph pattern and \mathcal{J} the set of SPARQL variables appearing in t. First, we use the mapping semantics in order to show that every graph pattern solution of t over the RDF dataset DS_m is a graph pattern solution of gp' over DS_m , for the common graph pattern solution domain $\mathcal{J} = var(t)$, inferring that:

$$[[t]]_{DS_m} \sqsubseteq \pi_{\mathcal{J}} \left([[gp']]_{DS_m} \right) \tag{39}$$

Then, we show that every graph pattern solution of gp' over the RDF dataset DS_m is a graph pattern solution of t over DS_m , for the common graph pattern solution domain \mathcal{J} , inferring that:

$$[[t]]_{DS_m} \supseteq \pi_{\mathcal{J}} ([[gp']]_{DS_m}) \tag{40}$$

From (39) and (40) we derive that $[[t]]_{DS_m} \equiv \pi_{\mathcal{J}}([[gp']]_{DS_m})$. Considering that the mapping used for the rewriting process has the same relationship (equivalence), we conclude the proof. Similarly, for mapping types containing subsumption relationships (\subseteq , \supseteq), we reach either to (39) or to (40) using the mapping semantics, proving that the rewriting step is semantics preserving (i.e. preserves the mappings semantics).

B.1 Proof of Lemma 6.1

In this section, we prove that the rewriting step performed for a Data Triple Pattern, based on a mapping of its subject part, is semantics preserving. According to Lemma 6.1, the only case that we examine concerns individuals appearing in the triple pattern's subject part.

Let i_s be an individual from the source ontology, $t = (i_s, predicate, object)$ be a Data Triple Pattern and \mathcal{J} be the set of SPARQL variables appearing in t. The evaluation of the triple pattern t over the RDF dataset DS_m is presented below:

$$[[t]]_{DS_m} = [[(i_s, predicate, object)]]_{DS_m}$$

We consider the following case:

1. Let i_t be an individual from the target ontology. Having a mapping $\mu : i_s \equiv i_t$ (i.e. $i_s^{\mathcal{I}} \equiv i_t^{\mathcal{I}}$) the rewritten (based on t's subject part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}^i_s(t,\mu) = (i_t, predicate, object)$$

$$[[gp']]_{DS_m} = [[(i_t, predicate, object)]]_{DS_m}$$

We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m}$: $\exists x, \exists y$, such that $(i_s, x, y) \in DS_m$, $\omega^p(t) = x$ and $\omega^o(t) = y$. Moreover, since $i_s \equiv i_t$ then $(i_t, x, y) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$.
- (b) $\forall \omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$: $\exists x, \exists y, \text{ such that } (i_t, x, y) \in DS_m, \ \omega^p(gp') = x \text{ and } \omega^o(gp') = y.$ Moreover, since $i_s \equiv i_t$ then $(i_s, x, y) \in DS_m$, inferring that $\omega \in [[t]]_{DS_m}$.

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} ([[gp']]_{DS_m})$$

This concludes the proof that the rewriting step of a Data Triple Pattern, based on a mapping of its subject part, is semantics preserving.

B.2 Proof of Lemma 6.2

In this section, we prove that the rewriting step performed for a Data Triple Pattern, based on a mapping of its object part, is semantics preserving. According to Lemma 6.2, the only cases that we examine concern individuals and classes appearing in the triple pattern's object part.

To begin with, we prove that the rewriting step performed for a Data Triple Pattern, based on an individual mapping of its object part, is semantics preserving. Let i_s be an individual from the source ontology, $t = (subject, predicate, i_s)$ be a Data Triple Pattern and \mathcal{J} be the set of SPARQL variables appearing in t. The evaluation of the triple pattern t over the RDF dataset DS_m is presented below:

$$[[t]]_{DS_m} = [[(subject, predicate, i_s)]]_{DS_m}$$

We consider the following case:

1. Let i_t be an individual from the target ontology. Having a mapping $\mu : i_s \equiv i_t$ (i.e. $i_s^{\mathcal{I}} = i_t^{\mathcal{I}}$), the rewritten (based on t's object part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_o^i(t,\mu) = (subject, predicate, i_t)$$

$$[[gp']]_{DS_m} = [[(subject, predicate, i_t)]]_{DS_m}$$

We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m}$: $\exists x, \exists y$, such that $(x, y, i_s) \in DS_m$, $\omega^s(t) = x$ and $\omega^p(t) = y$. Moreover, since $i_s \equiv i_t$ then $(x, y, i_t) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$.
- (b) $\forall \omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$: $\exists x, \exists y, \text{ such that } (x, y, i_t) \in DS_m, \ \omega^s(gp') = x \text{ and } \omega^p(gp') = y.$ Moreover, since $i_s \equiv i_t$ then $(x, y, i_s) \in DS_m$, inferring that $\omega \in [[t]]_{DS_m}$.

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{T}} ([[gp']]_{DS_m})$$

This concludes the proof that the rewriting step of a Data Triple Pattern, based on an individual mapping of its object part, is semantics preserving.

Similarly, we prove that the rewriting step performed for a Data Triple Pattern, based on a class mapping of its object part, is semantics preserving. Let c_s be a class from the source ontology, $t = (subject, rdf : type, c_s)$ be a Data Triple Pattern and \mathcal{J} be the set of SPARQL variables appearing in t. The evaluation of the triple pattern t over the RDF dataset DS_m is presented below:

$$[[t]]_{DS_m} = [[(subject, rdf : type, c_s)]]_{DS_m}$$

For the different types of class mappings, we consider the following cases:

1. Let c_t be a class from the target ontology. Having a mapping $\mu : c_s \equiv c_t$ (i.e. $c_s^{\mathcal{I}} = c_t^{\mathcal{I}}$), the rewritten (based on t's object part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_c^o(t,\mu) = (subject, rdf : type, c_t)$$

$$[[gp']]_{DS_m} = [[(subject, rdf : type, c_t)]]_{DS_m}$$

We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \text{ such that } (x, rdf : type, c_s) \in DS_m \text{ and } \omega^s(t) = x.$ Thus, $x \in c_s^{\mathcal{I}}$ and since $c_s^{\mathcal{I}} = c_t^{\mathcal{I}}$ then $(x, rdf : type, c_t) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}\left([[gp']]_{DS_m}\right) : \exists x, \text{ such that } (x, rdf : type, c_t) \in DS_m \text{ and } \omega^s(gp') = x.$ Thus, $x \in c_t^{\mathcal{I}}$ and since $c_s^{\mathcal{I}} = c_t^{\mathcal{I}}$ then $(x, rdf : type, c_s) \in DS_m$, inferring that $\omega \in [[t]]_{DS_m}$.

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} ([[gp']]_{DS_m})$$

2. Let c_{t1} and c_{t2} be classes from the target ontology. Having a mapping $\mu : c_s \equiv c_{t1} \sqcup c_{t2}$ (i.e. $c_s^{\mathcal{I}} = c_{t1}^{\mathcal{I}} \cup c_{t2}^{\mathcal{I}}$), the rewritten (based on t's object part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_c^o(t,\mu) = (subject, rdf : type, c_{t1}) \text{ UNION } (subject, rdf : type, c_{t2})$$

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, rdf : type, c_{t1}) \text{ UNION } (subject, rdf : type, c_{t2})]]_{DS_m} \\ &= [[(subject, rdf : type, c_{t1})]]_{DS_m} \cup [[(subject, rdf : type, c_{t2})]]_{DS_m} \\ &= [[t'_1]]_{DS_m} \cup [[t'_2]]_{DS_m} \end{split}$$

We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \text{ such that } (x, rdf : type, c_s) \in DS_m \text{ and } \omega^s(t) = x.$ Thus, $x \in c_s^{\mathcal{I}}$ and since $c_s^{\mathcal{I}} = c_{t1}^{\mathcal{I}} \cup c_{t2}^{\mathcal{I}}$ then $(x, rdf : type, c_{t1}) \in DS_m$ or $(x, rdf : type, c_{t2}) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$: $\exists x$, such that $(x, rdf : type, c_{t1}) \in DS_m$ or $(x, rdf : type, c_{t2}) \in DS_m$, and $\omega^s(t'_1) = x$ or $\omega^s(t'_2) = x$. Thus, $x \in c_{t1}^{\mathcal{I}} \cup c_{t2}^{\mathcal{I}}$ and since $c_s^{\mathcal{I}} = c_{t1}^{\mathcal{I}} \cup c_{t2}^{\mathcal{I}}$ then $(x, rdf : type, c_s) \in DS_m$, inferring that $\omega \in [[t]]_{DS_m}$.

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} \left([[gp']]_{DS_m} \right)$$

3. Let c_{t1} and c_{t2} be classes from the target ontology. Having a mapping $\mu : c_s \equiv c_{t1} \sqcap c_{t2}$ (i.e. $c_s^{\mathcal{I}} = c_{t1}^{\mathcal{I}} \cap c_{t2}^{\mathcal{I}}$), the rewritten (based on t's object part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_c^o(t,\mu) = (subject, rdf : type, c_{t1}) \text{ AND } (subject, rdf : type, c_{t2})$$

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, rdf : type, c_{t1}) \text{ AND } (subject, rdf : type, c_{t2})]]_{DS_m} \\ &= [[(subject, rdf : type, c_{t1})]]_{DS_m} \bowtie [[(subject, rdf : type, c_{t2})]]_{DS_m} \\ &= [[t'_1]]_{DS_m} \bowtie [[t'_2]]_{DS_m} \end{split}$$

We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \text{ such that } (x, rdf : type, c_s) \in DS_m \text{ and } \omega^s(t) = x.$ Thus, $x \in c_s^{\mathcal{I}}$ and since $c_s^{\mathcal{I}} = c_{t_1}^{\mathcal{I}} \cap c_{t_2}^{\mathcal{I}}$ then $(x, rdf : type, c_{t_1}) \in DS_m$ and $(x, rdf : type, c_{t_2}) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}\left([[gp']]_{DS_m}\right) : \exists x, \text{ such that } (x, rdf : type, c_{t1}) \in DS_m, (x, rdf : type, c_{t2}) \in DS_m \text{ and } \omega^s(t_1') = \omega^s(t_2') = x. \text{ Thus, } x \in c_{t1}^{\mathcal{I}} \cap c_{t2}^{\mathcal{I}} \text{ and since } c_s^{\mathcal{I}} = c_{t1}^{\mathcal{I}} \cap c_{t2}^{\mathcal{I}} \text{ then } (x, rdf : type, c_s) \in DS_m, \text{ inferring that } \omega \in [[t_1]]_{DS_m}.$

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} ([[gp']]_{DS_m})$$

4. Let c_t be a class, op_t be an object property from the target ontology, v_{op} be an individual and $\overline{cp} \in \{\neq, =\}$. Having a mapping $\mu : c_s \to \forall c_t.(op_t \ \overline{cp} \ v_{op})$ (i.e. $c_s^{\mathcal{I}} = \{\alpha \in c_t^{\mathcal{I}} \mid \exists b. (\alpha, b) \in op_t^{\mathcal{I}} \land b \ \overline{cp} \ v_{op}\}$), the rewritten (based on t's object part) graph pattern gp'and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_c^o(t, \mu) = (subject, rdf : type, c_t) \text{ AND } (subject, op_t, ?var)$$

FILTER(?var $\overline{cp} v_{op}$)

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, rdf : type, c_t) \text{ AND } (subject, op_t, ?var) \\ &\quad \text{FILTER}(?var \ \overline{\textbf{cp}} \ v_{op})]]_{DS_m} \\ &= \{\omega \in \left([[(subject, rdf : type, c_t)]]_{DS_m} \bowtie [[(subject, op_t, ?var)]]_{DS_m}\right) \\ &\quad |\omega \models ?var \ \overline{\textbf{cp}} \ v_{op}\} \\ &= \{\omega \in \left([[t'_1]]_{DS_m} \bowtie [[t'_2]]_{DS_m}\right) \mid \omega \models ?var \ \overline{\textbf{cp}} \ v_{op}\} \end{split}$$

Let $\mathcal{L} = \{ \alpha \in c_t^{\mathcal{I}} \mid \exists b. (\alpha, b) \in op_t^{\mathcal{I}} \land b \ \overline{cp} \ v_{op} \}.$ We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \text{ such that } (x, rdf : type, c_s) \in DS_m \text{ and } \omega^s(t) = x.$ Thus, $x \in c_s^{\mathcal{I}} \text{ and since } c_s^{\mathcal{I}} = \mathcal{L} \text{ then } (x, rdf : type, c_t) \in DS_m, \exists y \text{ such that } (x, op_t, y) \in DS_m \text{ and } y \ \overline{\operatorname{cp}} \ v_{op}, \text{ inferring that } \omega \in \pi_{\mathcal{J}} ([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$: $\exists x$, such that $(x, rdf : type, c_t) \in DS_m$, $\exists y$ such that $(x, op_t, y) \in DS_m, y \ \overline{cp} \ v_{op}$ and $\omega^s(t'_1) = \omega^s(t'_2) = x$. Thus, $x \in \mathcal{L}$ and since $c_s^{\mathcal{I}} = \mathcal{L}$ then $(x, rdf : type, c_s) \in DS_m$, inferring that $\omega \in [[t]]_{DS_m}$.

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} \left([[gp']]_{DS_m} \right)$$

5. Let c_t be a class, dp_t be a datatype property from the target ontology, v_{dp} be a data value and $cp \in \{\neq, =, >, <, \geq, \leq\}$. Having a mapping $\mu : c_s \to \forall c_t.(dp_t \ cp \ v_{dp})$ (i.e. $c_s^{\mathcal{I}} = \{\alpha \in c_t^{\mathcal{I}} \mid \exists b. (\alpha, b) \in dp_t^{\mathcal{I}} \land b \ cp \ v_{dp}\}$), the rewritten (based on t's object part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

 $\begin{array}{ll} gp' & = \mathcal{D}_c^o(t,\mu) = (subject,rdf:type,c_t) \text{ AND } (subject,dp_t,?var) \\ & \text{FILTER}(?var \ \texttt{cp} \ v_{dp}) \end{array}$

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, rdf : type, c_t) \text{ AND } (subject, dp_t, ?var) \\ & \text{FILTER}(?var \ \texttt{cp} \ v_{dp})]]_{DS_m} \\ &= \{\omega \in ([[(subject, rdf : type, c_t)]]_{DS_m} \bowtie [[(subject, dp_t, ?var)]]_{DS_m}) \\ & | \omega \models ?var \ \texttt{cp} \ v_{dp} \} \\ &= \{\omega \in ([[t'_1]]_{DS_m} \bowtie [[t'_2]]_{DS_m}) \mid \omega \models ?var \ \texttt{cp} \ v_{dp} \} \end{split}$$

Let $\mathcal{L} = \{ \alpha \in c_t^{\mathcal{I}} \mid \exists b. \ (\alpha, b) \in dp_t^{\mathcal{I}} \land b \text{ cp } v_{dp} \}.$ We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \text{ such that } (x, rdf : type, c_s) \in DS_m \text{ and } \omega^s(t) = x.$ Thus, $x \in c_s^{\mathcal{I}} \text{ and since } c_s^{\mathcal{I}} = \mathcal{L} \text{ then } (x, rdf : type, c_t) \in DS_m, \exists y \text{ such that } (x, dp_t, y) \in DS_m \text{ and } y \text{ cp } v_{dp}, \text{ inferring that } \omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$: $\exists x$, such that $(x, rdf : type, c_t) \in DS_m$, $\exists y$ such that $(x, dp_t, y) \in DS_m$, $y \text{ cp } v_{dp}$ and $\omega^s(t'_1) = \omega^s(t'_2) = x$. Thus, $x \in \mathcal{L}$ and since $c_s^{\mathcal{I}} = \mathcal{L}$ then $(x, rdf : type, c_s) \in DS_m$, inferring that $\omega \in [[t]]_{DS_m}$.

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} \left([[gp']]_{DS_m} \right)$$

6. Let c_t be a class, op_{t1} , op_{t2} be object properties from the target ontology and $\overline{cp} \in \{\neq, =\}$. Having a mapping $\mu : c_s \to \forall c_t.(op_{t1} \ \overline{cp} \ op_{t2})$ (i.e. $c_s^{\mathcal{I}} = \{\alpha \in c_t^{\mathcal{I}} \mid \exists b, \exists c. (\alpha, b) \in op_{t1}^{\mathcal{I}} \land (\alpha, c) \in op_{t2}^{\mathcal{I}} \land b \ \overline{cp} \ c\}$), the rewritten (based on t's object part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_{c}^{o}(t,\mu) = (subject, rdf : type, c_{t}) \text{ AND } (subject, op_{t1}, ?var_{1})$$

AND $(subject, op_{t2}, ?var_{2}) \text{ FILTER}(?var_{1} \ \overline{cp} \ ?var_{2})$

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, rdf : type, c_t) \text{ AND } (subject, op_{t1}, ?var_1) \\ & \text{AND } (subject, op_{t2}, ?var_2) \text{ FILTER}(?var_1 \ \overline{cp} \ ?var_2)]]_{DS_m} \\ &= \{\omega \in \left([[(subject, rdf : type, c_t)]]_{DS_m} \bowtie [[(subject, op_{t1}, ?var_1)]]_{DS_m} \\ & \bowtie [[(subject, op_{t2}, ?var_2)]]_{DS_m}\right) \mid \omega \models ?var_1 \ \overline{cp} \ ?var_2\} \end{split}$$

$$= \{\omega \in \left([[t_1']]_{DS_m} \bowtie [[t_2']]_{DS_m} \bowtie [[t_3']]_{DS_m} \right) \mid \omega \models ?var_1 \ \overline{\mathsf{cp}} \ ?var_2 \}$$

Let $\mathcal{L} = \{ \alpha \in c_t^{\mathcal{I}} \mid \exists b, \exists c. (\alpha, b) \in op_{t1}^{\mathcal{I}} \land (\alpha, c) \in op_{t2}^{\mathcal{I}} \land b \ \overline{cp} \ c \}.$ We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \text{ such that } (x, rdf : type, c_s) \in DS_m \text{ and } \omega^s(t) = x.$ Thus, $x \in c_s^{\mathcal{I}} \text{ and since } c_s^{\mathcal{I}} = \mathcal{L} \text{ then } (x, rdf : type, c_t) \in DS_m, \exists y \text{ such that } (x, op_{t1}, y) \in DS_m, \exists z \text{ such that } (x, op_{t2}, z) \in DS_m \text{ and } y \text{ } \overline{\mathbf{cp}} \text{ } z, \text{ inferring that } \omega \in \pi_{\mathcal{J}} ([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$: $\exists x$, such that $(x, rdf : type, c_t) \in DS_m$, $\exists y$ such that $(x, op_{t1}, y) \in DS_m$, $\exists z$ such that $(x, op_{t2}, z) \in DS_m$, $y \ \overline{cp} \ z$ and $\omega^s(t'_1) = \omega^s(t'_2) = \omega^s(t'_3) = x$. Thus, $x \in \mathcal{L}$ and since $c_s^{\mathcal{I}} = \mathcal{L}$ then $(x, rdf : type, c_s) \in DS_m$, inferring that $\omega \in [[t]]_{DS_m}$.

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} ([[gp']]_{DS_m})$$

7. Let c_t be a class, dp_{t1} , dp_{t2} be datatype properties from the target ontology and $cp \in \{\neq, =, >, <, \geq, \leq\}$. Having a mapping $\mu : c_s \to \forall c_t.(dp_{t1} \ cp \ dp_{t2})$ (i.e. $c_s^{\mathcal{I}} = \{\alpha \in c_t^{\mathcal{I}} \mid \exists b, \exists c. (\alpha, b) \in dp_{t1}^{\mathcal{I}} \land (\alpha, c) \in dp_{t2}^{\mathcal{I}} \land b \ cp \ c\}$), the rewritten (based on t's object part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_c^o(t,\mu) = (subject, rdf : type, c_t) \text{ AND } (subject, dp_{t1}, ?var_1)$$

AND $(subject, dp_{t2}, ?var_2) \text{ FILTER}(?var_1 \text{ cp } ?var_2)$

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, rdf : type, c_t) \text{ AND } (subject, dp_{t1}, ?var_1) \\ \text{ AND } (subject, dp_{t2}, ?var_2) \text{ FILTER}(?var_1 \text{ cp } ?var_2)]]_{DS_m} \\ &= \{\omega \in ([[(subject, rdf : type, c_t)]]_{DS_m} \bowtie [[(subject, dp_{t1}, ?var_1)]]_{DS_m} \\ &\bowtie [[(subject, dp_{t2}, ?var_2)]]_{DS_m}) \mid \omega \models ?var_1 \text{ cp } ?var_2 \} \\ &= \{\omega \in ([[t_1']]_{DS_m} \bowtie [[t_2']]_{DS_m} \bowtie [[t_3']]_{DS_m}) \mid \omega \models ?var_1 \text{ cp } ?var_2 \} \end{split}$$

Let $\mathcal{L} = \{ \alpha \in c_t^{\mathcal{I}} \mid \exists b, \exists c. (\alpha, b) \in dp_{t1}^{\mathcal{I}} \land (\alpha, c) \in dp_{t2}^{\mathcal{I}} \land b \text{ cp } c \}$. We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \text{ such that } (x, rdf : type, c_s) \in DS_m \text{ and } \omega^s(t) = x. \text{ Thus,}$ $x \in c_s^{\mathcal{I}} \text{ and since } c_s^{\mathcal{I}} = \mathcal{L} \text{ then } (x, rdf : type, c_t) \in DS_m, \exists y \text{ such that } (x, dp_{t1}, y) \in DS_m, \exists z \text{ such that } (x, dp_{t2}, z) \in DS_m \text{ and } y \text{ cp } z, \text{ inferring that } \omega \in \pi_{\mathcal{J}} ([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}\left([[gp']]_{DS_m}\right) : \exists x, \text{ such that } (x, rdf : type, c_t) \in DS_m, \exists y \text{ such that } (x, dp_{t1}, y) \in DS_m, \exists z \text{ such that } (x, dp_{t2}, z) \in DS_m, y \text{ cp } z \text{ and } \omega^s(t'_1) = \omega^s(t'_2) = \omega^s(t'_3) = x. \text{ Thus, } x \in \mathcal{L} \text{ and since } c_s^{\mathcal{I}} = \mathcal{L} \text{ then } (x, rdf : type, c_s) \in DS_m, \text{ inferring that } \omega \in [[t]]_{DS_m}.$

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} ([[gp']]_{DS_m})$$

This concludes the proof that the rewriting step of a Data Triple Pattern, based on a class mapping of its object part, is semantics preserving.

B.3 Proof of Lemma 6.3

In this section, we prove that the rewriting step performed for a Data Triple Pattern, based on a mapping of its predicate part, is semantics preserving. According to Lemma 6.3, the only cases that we examine concern object and datatype properties appearing in the triple pattern's predicate part.

To begin with, we prove that the rewriting step performed for a Data Triple Pattern, based on an object property mapping of its predicate part, is semantics preserving. Let op_s be an object property from the source ontology, $t = (subject, op_s, object)$ be a Data Triple Pattern and \mathcal{J} be the set of SPARQL variables appearing in t. The evaluation of the triple pattern t over the RDF dataset DS_m is presented below:

$$[[t]]_{DS_m} = [[(subject, op_s, object)]]_{DS_m}$$

For the different types of object property mappings, we consider the following cases:

1. Let op_t be an object property from the target ontology. Having a mapping $\mu : op_s \equiv op_t$ (i.e. $op_s^{\mathcal{I}} = op_t^{\mathcal{I}}$), the rewritten (based on t's predicate part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_{on}^p(t,\mu) = (subject, op_t, object)$$

 $[[gp']]_{DS_m} = [[(subject, op_t, object)]]_{DS_m}$

We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \exists y, \text{ such that } (x, op_s, y) \in DS_m, \ \omega^s(t) = x \text{ and } \omega^o(t) = y.$ Thus, $(x, y) \in op_s^{\mathcal{I}}$ and since $op_s^{\mathcal{I}} = op_t^{\mathcal{I}}$ then $(x, op_t, y) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{I}}([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$: $\exists x, \exists y, \text{ such that } (x, op_t, y) \in DS_m, \ \omega^s(gp') = x \text{ and } \omega^o(gp') = y$. Thus, $(x, y) \in op_t^{\mathcal{I}}$ and since $op_s^{\mathcal{I}} = op_t^{\mathcal{I}}$ then $(x, op_s, y) \in DS_m$, inferring that $\omega \in [[t]]_{DS_m}$.

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} ([[gp']]_{DS_m})$$

2. Let op_{t1} and op_{t2} be object properties from the target ontology. Having a mapping $\mu : op_s \equiv op_{t1} \circ op_{t2}$ (i.e. $op_s^{\mathcal{I}} = \{(\alpha, c) \mid \exists b. \ (\alpha, b) \in op_{t1}^{\mathcal{I}} \land (b, c) \in op_{t2}^{\mathcal{I}}\})$, the rewritten (based on t's predicate part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_{op}^p(t,\mu) = (subject, op_{t1}, ?var) \text{ AND } (?var, op_{t2}, object)$$

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, op_{t1}, ?var) \text{ AND } (?var, op_{t2}, object)]]_{DS_m} \\ &= [[(subject, op_{t1}, ?var)]]_{DS_m} \bowtie [[(?var, op_{t2}, object)]]_{DS_m} \\ &= [[t'_1]]_{DS_m} \bowtie [[t'_2]]_{DS_m} \end{split}$$

Let $\mathcal{L} = \{(\alpha, c) \mid \exists b. \ (\alpha, b) \in op_{t1}^{\mathcal{I}} \land (b, c) \in op_{t2}^{\mathcal{I}}\}$. We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \exists y, \text{ such that } (x, op_s, y) \in DS_m, \ \omega^s(t) = x \text{ and } \omega^o(t) = y.$ Thus, $(x, y) \in op_s^{\mathcal{I}}$ and since $op_s^{\mathcal{I}} = \mathcal{L}$ then $\exists z \text{ such that } (x, op_{t1}, z) \in DS_m$ and $(z, op_{t2}, y) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{I}}([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}\left([[gp']]_{DS_m}\right) : \exists x, \exists y, \exists z, \text{ such that } (x, op_{t1}, z) \in DS_m, \ (z, op_{t2}, y) \in DS_m, \ \omega^s(t_1') = x, \ \omega^o(t_1') = \omega^s(t_2') = z \text{ and } \omega^o(t_2') = y. \text{ Thus, } (x, y) \in \mathcal{L} \text{ and since } op_s^{\mathcal{I}} = \mathcal{L} \text{ then } (x, op_s, y) \in DS_m, \text{ inferring that } \omega \in [[t]]_{DS_m}.$

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} \left([[gp']]_{DS_m} \right)$$

3. Let op_{t1} and op_{t2} be object properties from the target ontology. Having a mapping $\mu : op_s \equiv op_{t1} \sqcup op_{t2}$ (i.e. $op_s^{\mathcal{I}} = op_{t1}^{\mathcal{I}} \sqcup op_{t2}^{\mathcal{I}}$), the rewritten (based on t's predicate part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_{op}^{p}(t,\mu) = (subject, op_{t1}, object)$$
 UNION $(subject, op_{t2}, object)$

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, op_{t1}, object) \text{ UNION } (subject, op_{t2}, object)]]_{DS_m} \\ &= [[(subject, op_{t1}, object)]]_{DS_m} \cup [[(subject, op_{t2}, object)]]_{DS_m} \\ &= [[t'_1]]_{DS_m} \cup [[t'_2]]_{DS_m} \end{split}$$

We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m}$: $\exists x, \exists y$, such that $(x, op_s, y) \in DS_m$, $\omega^s(t) = x$ and $\omega^o(t) = y$. Thus, $(x, y) \in op_s^{\mathcal{I}}$ and since $op_s^{\mathcal{I}} = op_{t1}^{\mathcal{I}} \cup op_{t2}^{\mathcal{I}}$ then $(x, op_{t1}, y) \in DS_m$ or $(x, op_{t2}, y) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$.
- (b) $\forall \omega \in \pi_{\mathcal{J}}\left([[gp']]_{DS_m}\right)$: $\exists x, \exists y$, such that $(x, op_{t1}, y) \in DS_m$ or $(x, op_{t2}, y) \in DS_m$, $\omega^s(t_1') = x$ or $\omega^s(t_2') = x$, and $\omega^o(t_1') = y$ or $\omega^o(t_2') = y$. Thus, $(x, y) \in op_{t1}^{\mathcal{I}} \cup op_{t2}^{\mathcal{I}}$ and since $op_s^{\mathcal{I}} = op_{t1}^{\mathcal{I}} \cup op_{t2}^{\mathcal{I}}$ then $(x, op_s, y) \in DS_m$, inferring that $\omega \in [[t]]_{DS_m}$.

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} ([[gp']]_{DS_m})$$

4. Let op_{t1} and op_{t2} be object properties from the target ontology. Having a mapping $\mu : op_s \equiv op_{t1} \sqcap op_{t2}$ (i.e. $op_s^{\mathcal{I}} = op_{t1}^{\mathcal{I}} \cap op_{t2}^{\mathcal{I}}$), the rewritten (based on t's predicate part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_{op}^{p}(t,\mu) = (subject, op_{t1}, object) \text{ AND } (subject, op_{t2}, object)$$

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, op_{t1}, object) \text{ AND } (subject, op_{t2}, object)]]_{DS_m} \\ &= [[(subject, op_{t1}, object)]]_{DS_m} \bowtie [[(subject, op_{t2}, object)]]_{DS_m} \\ &= [[t'_1]]_{DS_m} \bowtie [[t'_2]]_{DS_m} \end{split}$$

We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \exists y, \text{ such that } (x, op_s, y) \in DS_m, \ \omega^s(t) = x \text{ and } \omega^o(t) = y.$ Thus, $(x, y) \in op_s^{\mathcal{I}}$ and since $op_s^{\mathcal{I}} = op_{t_1}^{\mathcal{I}} \cap op_{t_2}^{\mathcal{I}}$ then $(x, op_{t_1}, y) \in DS_m$ and $(x, op_{t_2}, y) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}\left([[gp']]_{DS_m}\right) : \exists x, \exists y, \text{ such that } (x, op_{t1}, y) \in DS_m, (x, op_{t2}, y) \in DS_m, \\ \omega^s(t_1') = \omega^s(t_2') = x \text{ and } \omega^o(t_1') = \omega^o(t_2') = y. \text{ Thus, } (x, y) \in op_{t1}^{\mathcal{I}} \cap op_{t2}^{\mathcal{I}} \text{ and since } op_s^{\mathcal{I}} = op_{t1}^{\mathcal{I}} \cap op_{t2}^{\mathcal{I}} \text{ then } (x, op_s, y) \in DS_m, \text{ inferring that } \omega \in [[t]]_{DS_m}.$

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} ([[gp']]_{DS_m})$$

5. Let op_t be an object property from the target ontology. Having a mapping $\mu : op_s \equiv inv(op_t)$ (i.e. $op_s^{\mathcal{I}} = \{(b, \alpha) \mid (\alpha, b) \in op_t^{\mathcal{I}}\}$), the rewritten (based on t's predicate part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{ID}_{op}^p(t,\mu) = (object, op_t, subject)$$

$$[[gp']]_{DS_m} = [[(object, op_t, subject)]]_{DS_m}$$

Let $\mathcal{L} = \{(b, \alpha) \mid (\alpha, b) \in op_t^{\mathcal{I}}\}$. We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \exists y, \text{ such that } (x, op_s, y) \in DS_m, \ \omega^s(t) = x \text{ and } \omega^o(t) = y.$ Thus, $(x, y) \in op_s^{\mathcal{I}}$ and since $op_s^{\mathcal{I}} = \mathcal{L}$ then $(y, op_t, x) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m}) : \exists x, \exists y, \text{ such that } (x, op_t, y) \in DS_m, \ \omega^s(gp') = x \text{ and } \omega^o(gp') = y.$ Thus, $(x, y) \in \mathcal{L}$ and since $op_s^{\mathcal{I}} = \mathcal{L}$ then $(y, op_s, x) \in DS_m$, inferring that $\omega \in [[t]]_{DS_m}$.

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} \left([[gp']]_{DS_m} \right)$$

6. Let op_t be an object property and c_t be a class from the target ontology. Having a mapping $\mu : op_s \equiv \forall op_t.domain(c_t)$ (i.e. $op_s^{\mathcal{I}} = \{(\alpha, b) \mid (\alpha, b) \in op_t^{\mathcal{I}} \land \alpha \in c_t^{\mathcal{I}}\}$), the rewritten (based on t's object part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_{op}^{p}(t,\mu) = (subject, op_t, object) \text{ AND } (subject, rdf : type, c_t)$$

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, op_t, object) \text{ AND } (subject, rdf : type, c_t)]]_{DS_m} \\ &= [[(subject, op_t, object)]]_{DS_m} \bowtie [[(subject, rdf : type, c_t)]]_{DS_m} \\ &= [[t'_1]]_{DS_m} \bowtie [[t'_2]]_{DS_m} \end{split}$$

Let $\mathcal{L} = \{(\alpha, b) \mid (\alpha, b) \in op_t^{\mathcal{I}} \land \alpha \in c_t^{\mathcal{I}}\}$. We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \exists y, \text{ such that } (x, op_s, y) \in DS_m, \ \omega^s(t) = x \text{ and } \omega^o(t) = y.$ Thus, $(x, y) \in op_s^{\mathcal{I}}$ and since $op_s^{\mathcal{I}} = \mathcal{L}$ then $(x, op_t, y) \in DS_m$ and $(x, rdf : type, c_t) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m}) : \exists x, \exists y, \text{ such that } (x, op_t, y) \in DS_m, (x, rdf : type, c_t) \in DS_m, \ \omega^s(t_1') = \omega^s(t_2') = x \text{ and } \omega^o(t_1') = y. \text{ Thus, } (x, y) \in \mathcal{L} \text{ and since } op_s^{\mathcal{I}} = \mathcal{L} \text{ then } (x, op_s, y) \in DS_m, \text{ inferring that } \omega \in [[t_1]]_{DS_m}.$

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} \left([[gp']]_{DS_m} \right)$$

7. Let op_t be an object property and c_t be a class from the target ontology. Having a mapping $\mu : op_s \equiv \forall op_t.range(c_t)$ (i.e. $op_s^{\mathcal{I}} = \{(\alpha, b) \mid (\alpha, b) \in op_t^{\mathcal{I}} \land b \in c_t^{\mathcal{I}}\}$), the rewritten (based on t's object part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_{op}^{p}(t,\mu) = (subject, op_t, object) \text{ AND } (object, rdf : type, c_t)$$

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, op_t, object) \text{ AND } (object, rdf : type, c_t)]]_{DS_m} \\ &= [[(subject, op_t, object)]]_{DS_m} \bowtie [[(object, rdf : type, c_t)]]_{DS_m} \\ &= [[t'_1]]_{DS_m} \bowtie [[t'_2]]_{DS_m} \end{split}$$

Let $\mathcal{L} = \{(\alpha, b) \mid (\alpha, b) \in op_t^{\mathcal{I}} \land b \in c_t^{\mathcal{I}}\}$. We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m}$: $\exists x, \exists y, \text{ such that } (x, op_s, y) \in DS_m, \ \omega^s(t) = x \text{ and } \omega^o(t) = y$. Thus, $(x, y) \in op_s^{\mathcal{I}}$ and since $op_s^{\mathcal{I}} = \mathcal{L}$ then $(x, op_t, y) \in DS_m$ and $(y, rdf : type, c_t) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$.
- (b) $\forall \omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m}) : \exists x, \exists y, \text{ such that } (x, op_t, y) \in DS_m, (y, rdf : type, c_t) \in DS_m, \ \omega^s(t_1') = x \text{ and } \omega^o(t_1') = \omega^s(t_2') = y. \text{ Thus, } (x, y) \in \mathcal{L} \text{ and since } op_s^{\mathcal{I}} = \mathcal{L} \text{ then } (x, op_s, y) \in DS_m, \text{ inferring that } \omega \in [[t]]_{DS_m}.$

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} \left([[gp']]_{DS_m} \right)$$

This concludes the proof that the rewriting step of a Data Triple Pattern, based on an object property mapping of its predicate part, is semantics preserving.

Similarly, we prove that the rewriting step performed for a Data Triple Pattern, based on a datatype property mapping of its predicate part, is semantics preserving. Let dp_s be an object property from the source ontology, $t = (subject, dp_s, object)$ be a Data Triple Pattern and \mathcal{J} be the set of SPARQL variables appearing in t. The evaluation of the triple pattern t over the RDF dataset DS_m is presented below:

$$[[t]]_{DS_m} = [[(subject, dp_s, object)]]_{DS_m}$$

For the different types of datatype property mappings, we consider the following cases:

1. Let dp_t be a datatype property from the target ontology. Having a mapping $\mu : dp_s \equiv dp_t$ (i.e. $dp_s^{\mathcal{I}} = dp_t^{\mathcal{I}}$), the rewritten (based on t's predicate part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

 $gp' = \mathcal{D}_{dp}^p(t,\mu) = (subject, dp_t, object)$

$$[[gp']]_{DS_m} = [[(subject, dp_t, object)]]_{DS_m}$$

We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \exists y, \text{ such that } (x, dp_s, y) \in DS_m, \ \omega^s(t) = x \text{ and } \omega^o(t) = y.$ Thus, $(x, y) \in dp_s^{\mathcal{I}}$ and since $dp_s^{\mathcal{I}} = dp_t^{\mathcal{I}}$ then $(x, dp_t, y) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$: $\exists x, \exists y, \text{ such that } (x, dp_t, y) \in DS_m, \ \omega^s(gp') = x \text{ and } \omega^o(gp') = y$. Thus, $(x, y) \in dp_t^{\mathcal{I}}$ and since $dp_s^{\mathcal{I}} = dp_t^{\mathcal{I}}$ then $(x, dp_s, y) \in DS_m$, inferring that $\omega \in [[t]]_{DS_m}$.

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} ([[gp']]_{DS_m})$$

2. Let op_t be an object property and dp_t be a datatype property from the target ontology. Having a mapping $\mu : dp_s \equiv op_t \circ dp_t$ (i.e. $dp_s^{\mathcal{I}} = \{(\alpha, c) \mid \exists b. (\alpha, b) \in op_t^{\mathcal{I}} \land (b, c) \in dp_t^{\mathcal{I}}\}$), the rewritten (based on t's predicate part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

 $gp' = \mathcal{D}_{dp}^{p}(t,\mu) = (subject, op_t, ?var) \text{ AND } (?var, dp_t, object)$

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, op_t, ?var) \text{ AND } (?var, dp_t, object)]]_{DS_m} \\ &= [[(subject, op_t, ?var)]]_{DS_m} \bowtie [[(?var, dp_t, object)]]_{DS_m} \\ &= [[t'_1]]_{DS_m} \bowtie [[t'_2]]_{DS_m} \end{split}$$

Let $\mathcal{L} = \{(\alpha, c) \mid \exists b. \ (\alpha, b) \in op_t^{\mathcal{I}} \land (b, c) \in dp_t^{\mathcal{I}}\}$. We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m} : \exists x, \exists y, \text{ such that } (x, dp_s, y) \in DS_m, \ \omega^s(t) = x \text{ and } \omega^o(t) = y.$ Thus, $(x, y) \in dp_s^{\mathcal{I}}$ and since $dp_s^{\mathcal{I}} = \mathcal{L}$ then $\exists z$ such that $(x, op_t, z) \in DS_m$ and $(z, dp_t, y) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m}).$
- (b) $\forall \omega \in \pi_{\mathcal{J}}\left([[gp']]_{DS_m}\right) : \exists x, \exists y, \exists z, \text{ such that } (x, op_t, z) \in DS_m, (z, dp_t, y) \in DS_m, \omega^s(t'_1) = x, \ \omega^o(t'_1) = \omega^s(t'_2) = z \text{ and } \omega^o(t'_2) = y. \text{ Thus, } (x, y) \in \mathcal{L} \text{ and since } dp_s^{\mathcal{I}} = \mathcal{L} \text{ then } (x, dp_s, y) \in DS_m, \text{ inferring that } \omega \in [[t]]_{DS_m}.$

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} ([[gp']]_{DS_m})$$

3. Let dp_{t1} and dp_{t2} be datatype properties from the target ontology. Having a mapping $\mu : dp_s \equiv dp_{t1} \sqcup dp_{t2}$ (i.e. $dp_s^{\mathcal{I}} = dp_{t1}^{\mathcal{I}} \sqcup dp_{t2}^{\mathcal{I}}$), the rewritten (based on t's predicate part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_{dp}^{p}(t,\mu) = (subject, dp_{t1}, object) \text{ UNION } (subject, dp_{t2}, object)$$
$$[[gp']]_{DS_m} = [[(subject, dp_{t1}, object) \text{ UNION } (subject, dp_{t2}, object)]]_{DS_m}$$
$$= [[(subject, dp_{t1}, object)]]_{DS_m} \cup [[(subject, dp_{t2}, object)]]_{DS_m}$$
$$= [[t'_1]]_{DS_m} \cup [[t'_2]]_{DS_m}$$

We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m}$: $\exists x, \exists y$, such that $(x, dp_s, y) \in DS_m$, $\omega^s(t) = x$ and $\omega^o(t) = y$. Thus, $(x, y) \in dp_s^{\mathcal{I}}$ and since $dp_s^{\mathcal{I}} = dp_{t1}^{\mathcal{I}} \cup dp_{t2}^{\mathcal{I}}$ then $(x, dp_{t1}, y) \in DS_m$ or $(x, dp_{t2}, y) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$.
- (b) $\forall \omega \in \pi_{\mathcal{J}}\left([[gp']]_{DS_m}\right) : \exists x, \exists y, \text{ such that } (x, dp_{t1}, y) \in DS_m \text{ or } (x, dp_{t2}, y) \in DS_m, \omega^s(t_1') = x \text{ or } \omega^s(t_2') = x, \text{ and } \omega^o(t_1') = y \text{ or } \omega^o(t_2') = y. \text{ Thus, } (x, y) \in dp_{t1}^{\mathcal{I}} \cup dp_{t2}^{\mathcal{I}} \text{ and since } dp_s^{\mathcal{I}} = dp_{t1}^{\mathcal{I}} \cup dp_{t2}^{\mathcal{I}} \text{ then } (x, dp_s, y) \in DS_m, \text{ inferring that } \omega \in [[t]]_{DS_m}.$

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} ([[gp']]_{DS_m})$$

4. Let dp_{t1} and dp_{t2} be datatype properties from the target ontology. Having a mapping $\mu : dp_s \equiv dp_{t1} \sqcap dp_{t2}$ (i.e. $dp_s^{\mathcal{I}} = dp_{t1}^{\mathcal{I}} \cap dp_{t2}^{\mathcal{I}}$), the rewritten (based on t's predicate part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_{dn}^p(t,\mu) = (subject, dp_{t1}, object) \text{ AND } (subject, dp_{t2}, object)$$

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, dp_{t1}, object) \text{ AND } (subject, dp_{t2}, object)]]_{DS_m} \\ &= [[(subject, dp_{t1}, object)]]_{DS_m} \bowtie [[(subject, dp_{t2}, object)]]_{DS_m} \\ &= [[t'_1]]_{DS_m} \bowtie [[t'_2]]_{DS_m} \end{split}$$

We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m}$: $\exists x, \exists y$, such that $(x, dp_s, y) \in DS_m$, $\omega^s(t) = x$ and $\omega^o(t) = y$. Thus, $(x, y) \in dp_s^{\mathcal{I}}$ and since $dp_s^{\mathcal{I}} = dp_{t1}^{\mathcal{I}} \cap dp_{t2}^{\mathcal{I}}$ then $(x, dp_{t1}, y) \in DS_m$ and $(x, dp_{t2}, y) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$.
- (b) $\forall \omega \in \pi_{\mathcal{J}}\left([[gp']]_{DS_m}\right) : \exists x, \exists y, \text{ such that } (x, dp_{t1}, y) \in DS_m, (x, dp_{t2}, y) \in DS_m, \\ \omega^s(t_1') = \omega^s(t_2') = x \text{ and } \omega^o(t_1') = \omega^o(t_2') = y. \text{ Thus, } (x, y) \in dp_{t1}^{\mathcal{I}} \cap dp_{t2}^{\mathcal{I}} \text{ and since } \\ dp_s^{\mathcal{I}} = dp_{t1}^{\mathcal{I}} \cap dp_{t2}^{\mathcal{I}} \text{ then } (x, dp_s, y) \in DS_m, \text{ inferring that } \omega \in [[t]]_{DS_m}.$

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} \left([[gp']]_{DS_m} \right)$$

5. Let dp_t be a datatype property and c_t be a class from the target ontology. Having a mapping $\mu : dp_s \equiv \forall dp_t.domain(c_t)$ (i.e. $dp_s^{\mathcal{I}} = \{(\alpha, b) \mid (\alpha, b) \in dp_t^{\mathcal{I}} \land \alpha \in c_t^{\mathcal{I}}\}$), the rewritten (based on t's object part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_{dp}^{p}(t,\mu) = (subject, dp_t, object) \text{ AND } (subject, rdf : type, c_t)$$
$$[[gp']]_{DS_m} = [[(subject, dp_t, object) \text{ AND } (subject, rdf : type, c_t)]]_{DS_m}$$
$$= [[(subject, dp_t, object)]]_{DS_m} \bowtie [[(subject, rdf : type, c_t)]]_{DS_m}$$
$$= [[t'_1]]_{DS_m} \bowtie [[t'_2]]_{DS_m}$$

Let $\mathcal{L} = \{(\alpha, b) \mid (\alpha, b) \in dp_t^{\mathcal{I}} \land \alpha \in c_t^{\mathcal{I}}\}$. We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m}$: $\exists x, \exists y$, such that $(x, dp_s, y) \in DS_m$, $\omega^s(t) = x$ and $\omega^o(t) = y$. Thus, $(x, y) \in dp_s^{\mathcal{I}}$ and since $dp_s^{\mathcal{I}} = \mathcal{L}$ then $(x, dp_t, y) \in DS_m$ and $(x, rdf : type, c_t) \in DS_m$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$.
- (b) $\forall \omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m}) : \exists x, \exists y, \text{ such that } (x, dp_t, y) \in DS_m, (x, rdf : type, c_t) \in DS_m, \ \omega^s(t_1') = \omega^s(t_2') = x \text{ and } \omega^o(t_1') = y. \text{ Thus, } (x, y) \in \mathcal{L} \text{ and since } dp_s^{\mathcal{I}} = \mathcal{L} \text{ then } (x, dp_s, y) \in DS_m, \text{ inferring that } \omega \in [[t]]_{DS_m}.$

From the two premises above, we conclude that:

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} \left([[gp']]_{DS_m} \right)$$

6. Let dp_t be a datatype property from the target ontology, v_{dp} be a data value and $cp \in \{\neq, =, >, <, \geq, \leq\}$. Having a mapping $\mu : dp_s \equiv \forall dp_t.range(cp v_{dp})$ (i.e. $dp_s^{\mathcal{I}} = \{(\alpha, b) \mid (\alpha, b) \in dp_t^{\mathcal{I}} \land b \ cp \ v_{dp}\}$), the rewritten (based on t's object part) graph pattern gp' and its evaluation over the RDF dataset DS_m are the following:

$$gp' = \mathcal{D}_{dp}^p(t,\mu) = (subject, dp_t, object)$$
 FILTER $(object \ cp \ v_{dp})$

$$\begin{split} [[gp']]_{DS_m} &= [[(subject, dp_t, object) \ \text{FILTER}(object \ \texttt{cp} \ v_{dp})]]_{DS_m} \\ &= \{\omega \in [[(subject, dp_t, object)]]_{DS_m} \mid \omega \models object \ \texttt{cp} \ v_{dp}\} \\ &= \{\omega \in [[t']]_{DS_m} \mid \omega \models object \ \texttt{cp} \ v_{dp}\} \end{split}$$

Let $\mathcal{L} = \{(\alpha, b) \mid (\alpha, b) \in dp_t^{\mathcal{I}} \land b \text{ cp } v_{dp}\}$. We consider two premises:

- (a) $\forall \omega \in [[t]]_{DS_m}$: $\exists x, \exists y$, such that $(x, dp_s, y) \in DS_m$, $\omega^s(t) = x$ and $\omega^o(t) = y$. Thus, $(x, y) \in dp_s^{\mathcal{I}}$ and since $dp_s^{\mathcal{I}} = \mathcal{L}$ then $(x, dp_t, y) \in DS_m$ and $y \text{ cp } v_{dp}$, inferring that $\omega \in \pi_{\mathcal{J}}([[gp']]_{DS_m})$.
- (b) $\forall \omega \in \pi_{\mathcal{J}}\left([[gp']]_{DS_m}\right) : \exists x, \exists y, \text{ such that } (x, dp_t, y) \in DS_m, y \text{ cp } v_{dp}, \omega^s(t') = x$ and $\omega^o(t') = y$. Thus, $(x, y) \in \mathcal{L}$ and since $dp_s^{\mathcal{I}} = \mathcal{L}$ then $(x, dp_s, y) \in DS_m$, inferring that $\omega \in [[t]]_{DS_m}$.

$$[[t]]_{DS_m} \equiv \pi_{\mathcal{J}} \left([[gp']]_{DS_m} \right)$$

This concludes the proof that the rewriting step of a Data Triple Pattern, based on a datatype property mapping of its predicate part, is semantics preserving.