Mapping RDF knowledge bases using exchange samples

Carlos R. Rivero [a,*], Inma Hernández [b], David Ruiz [b], Rafael Corchuelo [b]

[a] Rochester Institute of Technology, Department of Computer Science, 102 Lomb Memorial Dr., Rochester, NY 14623, USA
[b] University of Seville, ETSI Informática, Avda. Reina Mercedes s/n, Sevilla E-41012, Spain

A B S T R A C T

Nowadays, the Web of Data is in its earliest stages; it is currently organised into a variety of linked knowledge bases that have been developed independently by different organisations. RDF is one of the most popular languages to represent data in this context, which motivates the need to perform complex integration tasks amongst RDF knowledge bases. These tasks are performed using schema mappings, which are declarative specifications of the relationships amongst a source and a target knowledge base. Generating schema mappings automatically is appealing because this relieves users from the burden of handcrafting them. In the literature, the vast majority of proposals are based on the data models of the knowledge bases to be integrated, that is, on classes, properties, and constraints. In the Web of Data, there exist many data models that comprise very few constraints or no constraints at all, which has motivated some researchers to work on an alternate paradigm that does not rely on constraints. Unfortunately, the current proposals that fit this paradigm are not completely automatic. In this article, we present our proposal to automatically generate schema mappings amongst RDF knowledge bases. Its salient features are that it uses a single input exchange sample and a set of input correspondences, but does not require any constraints to be available or any user intervention; it has been validated and evaluated using many experiments that prove that it is effective and efficient in practice; the schema mappings that it produces are GLAV. Other researchers can reproduce our experiments since all of our implementations and repositories are publicly available.

Keywords:

Web of Data

RDF

Schema mapping

Data exchange

1. Introduction

Currently, there is an increasing interest in publishing, sharing, and exposing data on the Web, so as to evolve it into a Web of Data in which RDF is becoming pervasive [10,16,36]. There are thousands of knowledge bases available, many of which share a common purpose but have been developed by independent organisations in isolation [15,16]. Government, life sciences, geography, media, education, libraries, or scholarly publications range amongst the most popular current information domains [37]. There are many initiatives whose goal is to link these knowledge bases, which is the first step to perform complex integration processes [36].

Integration usually refers to several crucial tasks, such as data integration [68], virtual integration [38], data warehousing [32], model evolution [29], model matching [26], record linkage [42,53], or data exchange [28]. In this article, we focus on data exchange, whose goal is to populate a target knowledge base using data that come from one or more source knowledge bases. Data exchange has been paid much attention in the database context, i.e., relational, nested-relational, or XML [8,9,28,54]. Furthermore, the emergence of RDF is motivating some authors to work on data exchange in the context of the Web of Data [11,51,62,63].

Data exchange is performed by means of schema mappings, which are declarative specifications of the relationships amongst a source and a target knowledge bases [4,5]. A schema mapping can be of the following types [44]: (1) GAV (Global-As-View), which relates one single target entity to many source entities; (2) LAV (Local-As-View), which relates one single source entity to many target entities; and (3) GLAV, which relates many source and target entities.

Generating schema mappings automatically is appealing because this relieves users from the burden of handcrafting them, so researchers have focused on helping users generate them [56]. In the literature, many proposals are based on the data models to be integrated [35,47,48,54,57,61,62]. By data model, we refer to a set of entities (that is, classes and properties) and a set of constraints that establish relationships between some entities (for instance, class *A* is a specialisation of class *B*, property *P* has class *C* as its domain, and so on). In the Web of Data, there are many data models that comprise very few or no constraints at all, which typically results in data models that merely specify a set of entities [21,36,37,43,66]. Therefore, relying on data models with constraints to generate schema mappings is not appealing in the general context of the Web of Data.

* Corresponding author. fax.: +1 585 475 2979.
E-mail addresses: crr@cs.rit.edu (C.R. Rivero), inmahernandez@us.es (I. Hernández), druiz@us.es (D. Ruiz), corchu@us.es (R. Corchuelo).

The literature provides a variety of techniques to generate schema mappings that do not focus on data models. Unfortunately, some of them rely on handcrafting the schema mappings [15,27,45,49–51,58], which is not appealing at all; and a few others rely on exchange samples [1,2,4,56], which make them more appealing, but require user intervention, or are hybrid and require constraints to be available.

In this article, we present a proposal to automatically generate schema mappings between two RDF knowledge bases using a single input exchange sample and a set of input n:m correspondences. An exchange sample comprises a subset of source data and a subset of target data that is the expected result of exchanging the source data. Correspondences are hints that specify which entities in the source and target knowledge bases correspond to each other, i.e., are somewhat related [12,67,69]. These schema mappings can be easily transformed into SPARQL queries. Our proposal does not rely on constraints of the source and target data models and does not require any user intervention, not even to adapt the input exchange example so that it is feasible to generate the schema mappings. We have validated our proposal using ten data exchange problems amongst various knowledge bases. In our validation, the execution time never exceeded one second, and the data exchanged were as expected in every case, which suggests that it is very efficient in practice and that the generated schema mappings are appropriate. Additionally, we have evaluated the performance of our proposal when data exchange problems scale. We used four synthetic data exchange patterns proposed by MostoBM [63], a benchmark for testing data exchange proposals in the context of the Web of Data. We instantiated the synthetic data exchange patterns into 2 000 non-trivial data exchange problems that we used to evaluate our proposal. Our evaluation results suggest that our proposal works well as the data exchange problems scale. We also prove that the schema mappings our proposal generates are GLAV. Furthermore, we prove that the schema mappings output by our proposal fit the input correspondences, i.e., they do not lose any information and take all of the input correspondences into account. Our proposal also uses a best-effort strategy that allows the user to freely define the input exchange sample, so it is not mandatory that this exchange sample fits the output schema mappings. We implemented a research prototype that is publicly available [60], together with our repository of data exchange problems, the scripts to validate and evaluate the performance of our proposal, and our experimental results. Our goal was twofold: on the one hand, this allows other researchers to faithfully reproduce our experiments, which is crucial for the advance of science [30]; on the other hand, our implementation and our repository can be extended to cope with future requirements.

Preliminary results were presented elsewhere [64]. We have improved on them as follows: we present an overall picture to use our proposal; we formalise a conceptual framework that accommodates it; we describe our algorithms within the previous framework; we analyse a number of desired properties, its theoretical complexity, and make its limitations explicit; we have also validated it and evaluated how scalable it is. The rest of the article is organised as follows: in Section 2, we report on several related proposals and compare them with ours; Section 3 presents a general overview to use our proposal; Section 4 introduces our conceptual framework; Section 5 presents the algorithms of our proposal; Section 6 analyses some desired properties of our proposal and its theoretical complexity; Sections 7 and 8 present our validation and scalability evaluation; and Section 9 recaps on our main conclusions. Finally, Appendix A presents some ancillary propositions that we used to support our theorems.

2. Related work

In this article, our focus is on GLAV schema mappings, so we restrict our attention to proposals that generate such mappings. Our brief survey completes the picture that [65] presented elsewhere regarding proposals that generate GAV mappings. Unfortunately, we are not aware of a similar survey regarding LAV proposals.

We have classified the proposals that generate GLAV schema mappings into three groups, namely: (1) handcraft-based proposals, (2) constraint-based proposals, and (3) sample-based proposals. The proposals in the first group require the user to handcraft schema mappings. In the following sections, we first survey the proposals in each group and then discuss on them.

2.1. Handcraft-based proposals

There are a number of proposals that focus on handcrafting schema mappings, which are expressed as queries but can be viewed as implicitly generating schema mappings: in the proposal by Dou et al. [27], queries are represented using Web-PDDL, an ad-hoc language that was designed by the authors; then, a reasoner takes these queries as input to perform data exchange. This is similar in spirit to the proposals by Bizer and Schultz [15], Parreiras et al. [51] and Ressler et al. [58], the difference being the language used to represent the queries: Bizer and Schultz [15], and Ressler et al. [58] use SPARQL, whereas Parreiras et al. [51] use an extension of the Object Constraint Language (OCL) that supports RDF knowledge bases.

Mocan and Cimpian [49] presented a framework to describe schema mappings in terms of first-order logic formulae that can be mapped onto WSML rules very easily. Their proposal is similar in spirit to the one by Omelayenko [50], whose focus was on B2B applications, and the one by Maedche et al. [45], whose focus was on modelling schema mappings in a general-purpose setting.

2.2. Constraint-based proposals

These proposals focus on generating schema mappings building on correspondences and constraints on the source and target data models. These proposals are able to compute subsets of data in the source knowledge base that need to be exchanged as a whole, and subsets of data in the target knowledge base that need to be created as a whole [62]. To compute them, they rely on user-defined constraints and the inherent constraints of certain data models, such as paths from the root to a leaf in a nested-relational data model, or hierarchy relations amongst classes in an RDF data model. Then, several combinations of these subsets of data are used to generate the final schema mappings [54].

Popa et al. [54] presented a seminar proposal regarding the generation of schema mappings amongst nested-relational data models, which was later incorporated into IBM's Clio [35]. This proposal works with 1:1 correspondences, i.e., they only relate one entity in the source with one entity in the target, and was later extended to work with n:1 correspondences by Raffio et al. [57], who used a visual mapping language to represent complex correspondences, including grouping functions, aggregation functions, or dependent correspondences. Mecca et al. [47] extended the previous proposals by computing core schema mappings, a type of schema mapping that generates non-redundant target data when performing data exchange.

In the context of RDF data models, Mergen and Heuser [48] devised an automated proposal that works with a subset of taxonomies. Their algorithm tries to find subsets of correspondences that are involved in the taxonomies, and they are translated into executable scripts that are represented in an ad-hoc script language. The proposals by Rivero et al. [61,62] are able to work with RDF data models whose constraints are interpreted as graphs that are traversed to compute source and target kernels, each of which comprises a subset of the source data model that needs to be exchanged as a whole, and a subset of the target data model that needs to be created as a whole. Kernels are automatically translated into SPARQL queries. Note that both the executable scripts by Mergen and Heuser [48], and

the SPARQL queries by Rivero et al. [61,62] can be seen as representations of schema mappings in specific-purpose languages.

2.3. Sample-based proposals

These proposals aim to generate schema mappings from a set of exchange samples. In the relational or nested-relational contexts, Alexe et al. [1] devised a proposal that helps users understand and maintain generated schema mappings by extracting exchange samples from the source and target knowledge bases. This proposal uses Clio to generate schema mappings based on constraints of the data models, and it illustrates the following: (1) relationships in a specific schema mapping, (2) sample source data that this schema mapping would extract when performing data exchange, and (3) the target data generated by those source data. Alexe et al. [2] presented Muse, which aims to help users generate and understand schema mappings building on exchange samples. Muse assumes that source and target data models, together with their constraints, exist, and it is able to infer grouping functions by analysing the answers to a number of questions it poses to the users.

Alexe et al. [4] devised a proposal to generate a number of schema mappings by means of a finite set of exchange samples. This proposal is able to compute whether or not two input exchange samples have incoherences from a structural point of view, i.e., whether or not these two exchange samples generate schema mappings that shall result in erroneous target data. If the input set of exchange samples have not any incoherences, then it generates the schema mappings. These authors developed a conceptual framework to conduct systematic research on exchange samples in the database context [6]. Furthermore, Qian et al. [56] presented a proposal that is based on exchange samples of target data only. The users are responsible for providing the target data that they wish to be created; then, every piece of data that appears in both source and target knowledge bases represents a correspondence between two entities. Finally, schema mappings are generated by means of these correspondences and the constraints of the source and target data models.

2.4. Discussion

After surveying current proposals to generate schema mappings, we conclude that some of them focus on handcrafting them [15,27,45,49–51,58], which is not appealing since users have to write them, check whether they work as expected or not, make changes if necessary, and restart this cycle [52]. Contrarily, our proposal automatically generates schema mappings without the intervention of the user, and it uses a single exchange sample and a number of correspondences as input.

Regarding constraint-based proposals [35,47,48,54,57,61,62], they are not so appealing in the general context of the Web of Data because RDF allows to represent data building solely on entities, i.e., without any constraints, which is very common in this context [21,36,37,43]. Contrarily to the constraint-based proposals, ours does not rely on constraints of the data models that are integrated to generate schema mappings, but on a single exchange sample and a set of correspondences, which fits perfectly in the general context of the Web of Data.

Regarding sample-based proposals, some of them assume that source and target data models exist, together with their constraints [1,2,56]. Therefore, they rely on a hypothesis that, as was the case for the constraint-based proposals, does not generally hold in the Web of Data. The proposal by Alexe et al. [4] does not have the previous drawback, but it requires the user to provide an exchange sample for each schema mapping to be automatically generated. Furthermore, if this proposal finds the input exchange samples inappropriate to generate schema mappings, the user is responsible for adapting them. Contrarily, our proposal requires the user to provide a single exchange sample and a set of correspondences and adapts it automatically, if necessary.

Finally, to the best of our knowledge, our proposal is the first one that is able to work with n: m correspondences that generalise 1:1 and n:1 correspondences in other proposals.

3. Overview

Fig. 1 presents a real-world data exchange problem that we use to illustrate our proposal. In this problem, our goal is to generate a number of schema mappings to perform data exchange from a part of DBpedia 3.8 [17] to a part of Freebase [19]. On one hand, DBpedia is a community effort to annotate and make the data stored at Wikipedia accessible by means of RDF technologies. On the other hand, Freebase is a knowledge base that models general human knowledge in which the data is collaboratively created and maintained.

The initial step of our proposal consists of specifying the input data, i.e., a single exchange sample and a set of correspondences. The single exchange sample is expected to be an exact sample of the source and target data that the user wishes to exchange. For instance, the exchange sample in Fig. 1 (see d_{EX} in Step 0) comprises a set of source triples regarding Clint Eastwood and one of his marriages, and a set of target triples that specify how these data are structured according to the target entities. For the sake of brevity, we have simplified *Clint_Eastwood* to *CEastwood*, *Dina_Eastwood* to *DEastwood*, and *Clint Eastwood* to *Clint East*. This exchange sample is represented using the N3 notation [13], which is a non-XML serialisation of RDF that has been designed for aiding human-readability. We use the prefixes in Table 1, in which the first row specifies the default URI.

Our proposal takes a number of n:m correspondences over the source and target entities as input. This set indicates the relationships that exist amongst the source and target entities in the data exchange problem that we wish to solve. It is expected that the user has to relate the source entities that should be exchanged as a whole, and the target entities that need to be created as a whole. Note that it is possible to automatically discover these correspondences [12]; however, finding them is orthogonal to the problem of generating schema mappings, which is the reason why we do not discuss this problem further. Fig. 1 (see the right side of Step 0) shows three correspondences, namely: v_1 indicates that there exists a relation between a person in both the DBpedia and Freebase knowledge bases; v_2 states that the name of a person in DBpedia is related to the alias of a common topic in Freebase; and v_3 indicates that a person and her/his spouse in DBpedia are related to a marriage in Freebase.

In the first step, our proposal automatically computes a number of candidate exchange samples, each of which comprises a subset of source data that need to be exchanged as a whole, and a subset of target data that need to be created as a whole. To compute them, we combine all of the pieces of connected data that involve the entities in every correspondence. For instance, Fig. 1 (see Steps 1 and 2) shows some of the candidate exchange samples that result from correspondence v_1, namely: d_{11}, d_{12}, and d_{13}. Note that we group exchange samples by correspondence. These candidate exchange samples are generated by performing the Cartesian product of the following data:

$$\{CEastwood\ rdf{:}type\ dp{:}Person,\ DEastwood\ rdf{:}type\ dp{:}Person\} \times \{CEastwood\ rdf{:}type\ fp{:}person,\ DEastwood\ rdf{:}type\ fp{:}person\}$$

The second step consists of discarding candidate exchange samples that are not useful to generate the final set of schema mappings. We keep candidate exchange samples in which there is, at least, a subset of target data that can be generated using the source data, and

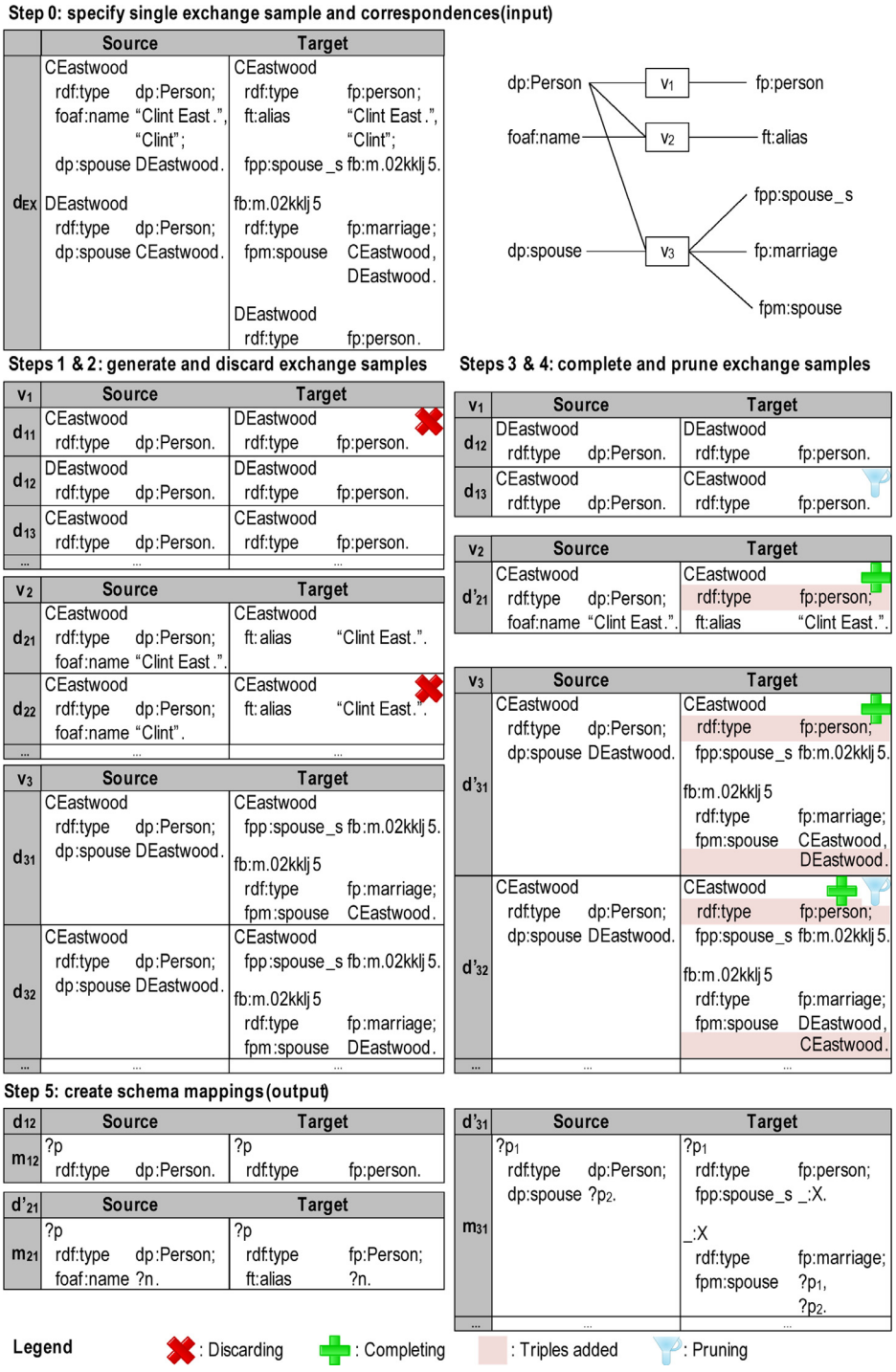


Fig. 1. A running example to illustrate our proposal.

we minimise the target data that do not exist in the source. For instance, in Fig. 1 (See Steps 1 and 2), we discard d_{11} because constant *DEastwood* in the target triple does not appear in the source triples; similarly, we discard d_{22} because constant "*Clint East.*" does not appear in any source triple. Note, however, that exchange samples d_{31} and d_{32} are kept, even though identifier *fb*:*m*.02*kkj*5 is not present in any source triple; we do not require every identifier or literal in the target to be present in the source because we can complete the exchange samples automatically, but require that at least a minimum number of constants are present in both the source and the target so that our procedure can work.

The third step consists of completing exchange samples, i.e., if the same source data can lead to different data in different exchange samples, it is then necessary to complete those exchange samples by adding target data to them. Therefore, we identify the exchange samples that have the same source data but differ in the target data, and we complete them without the user intervention. For instance, in Fig. 1 (see Steps 3 and 4), we complete d_{21} by adding *CEastwood rdf*:*type fp*:*person*. This is due to the fact that exchange sample d_{13} indicates that *CEastwood rdf*:*type Person* can be exchanged as *CEastwood rdf*:*type fp*:*person*. Unless otherwise stated, our proposal interprets that every constant can be generalised to a variable, which

Table 1
Summary of prefixes.

Prefix	URI
	http://dbpedia.org/resource/
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
foaf	http://xmlns.com/foaf/0.1/
dp	http://dbpedia.org/ontology/
fb	http://rdf.freebase.com/ns/
ft	http://rdf.freebase.com/ns/common.topic.
fp	http://rdf.freebase.com/ns/people.
fpm	http://rdf.freebase.com/ns/people.marriage.
fpp	http://rdf.freebase.com/ns/people.person.
fl	http://rdf.freebase.com/ns/location.
fll	http://rdf.freebase.com/ns/location.location.
gw	http://govwild.org/0.6/GWOntology.rdf#

means that d_{13} is interpreted as every identifier of type *Person* can be exchanged as an identifier of type *fp:person*.

In the fourth step, we prune redundant exchange samples, i.e., samples that shall generate the same schema mappings. For instance, in Fig. 1 (see Steps 3 and 4), we prune d_{13} since it shall generate the same schema mapping as exchange sample d_{12}. The same occurs with d'_{31} and d'_{32}.

Finally, the fifth step transforms each exchange sample into a schema mapping, which is built by substituting the source and target constants by variables or blank nodes to generate labelled nulls [28,46]. These schema mappings may be easily transformed into SPARQL queries to exchange data between the integrated knowledge bases. For instance, in Fig. 1 (see Step 5), m_{12} is generated by means of d_{12}, m_{21} results from d'_{21}, and m_{31} results from d'_{31}. Note that we have also used the N3 notation to represent both left and right components of schema mappings.

4. Conceptual framework

In this section, we present the conceptual framework that we use to describe our proposal. We define its foundations, triples, schema mappings, homomorphisms, exchange samples, correspondences, and data exchange problems.

4.1. Foundations

The data model of a knowledge base is composed of entities, which are denoted by means of URIs. Entities can be classified into classes and properties, which can be further classified into data and object properties. We denote the sets of classes, data properties, and object properties as follows:

$$[Class, DataProperty, ObjectProperty]$$

and define the set of entities and properties as follows:

$$Entity == Class \cup Property$$
$$Property == DataProperty \cup ObjectProperty$$

Class, *DataProperty*, and *ObjectProperty* are pairwise disjoint sets, that is:

$$|(Class \cap DataProperty = \emptyset) \wedge (Class \cap ObjectProperty = \emptyset) \wedge (DataProperty \cap ObjectProperty = \emptyset)$$

In addition to a data model, a knowledge base also comprises a set of constants to describe the data. A constant can be an identifier, which refers to a piece of data that can be modelled by means of a URI, or a literal, which denotes a value of a simple data type. Furthermore, it is possible to define blank nodes, which denote anonymous data. We denote these sets as follows:

$$[Identifier, Literal, BlankNode]$$

and define the set of constants as follows:

$$Constant == Identifier \cup Literal$$

Identifier, *Literal*, and *BlankNode* are pairwise disjoint sets [41], that is:

$$|(Identifier \cap Literal = \emptyset) \wedge (Identifier \cap BlankNode = \emptyset) \wedge (Literal \cap BlankNode = \emptyset)$$

Example 1. Fig. 1 (see the left side of Step 0) shows that *dp:Person* is a class, *foaf:name* is a data property, *dp:spouse* is an object property, and they are represented by means of URIs. Furthermore, *CEastwood* is an identifier, and "*Clint East.*" is a literal. *_:X* is an example of a blank node (see m_{31} in Step 5).

4.2. Triples and schema mappings

A knowledge base comprises a set of triples, each of which is a three tuple in which the first element is called subject, the second predicate, and the third object.

$$Triple == Subject \times Predicate \times Object$$
$$Subject == Constant$$
$$Predicate == Property \cup \{rdf:type\}$$
$$Object == Constant \cup Class$$

In our framework, it is mandatory for all triples to be ground, i.e., to not comprise any blank nodes. This is not a shortcoming since blank nodes in a knowledge base can be very easily preprocessed and transformed into regular identifiers.

Note that there is a contradiction between the RDF and the SPARQL recommendations. On the one hand, the RDF recommendation does not allow to use literals in the subject of a triple [41]; on the other hand, the SPARQL recommendation actually allows to use literals in the subject of a triple [55]. Since our goal is to generate schema

mappings that may be easily transformed into SPARQL queries, we decided to include literals in the subject of triples. Note, too, that a predicate can be a property or *rdf:type*, which is a predefined RDF construct that is not included in set *Property*.

In our proposal, schema mappings are conjunctive queries [44]. Such queries consist of a target part that specifies which triples need to be constructed, and a source part that specifies which triples need to be retrieved. These parts can then be represented as sets of triple patterns that are implicitly connected by means of logical ANDs. A triple pattern generalises the concept of triple by allowing the subject, and/or the object to be variables or blank nodes. In the rest of this article, we refer to triple patterns as patterns for the sake of brevity. We denote the set of all variables as follows:

$$[Variable]$$

and define the set of all patterns as follows:

$$Pattern == Subject^? \times Predicate \times Object^?$$

$$Subject^? == Subject \cup Variable \cup BlankNode$$

$$Object^? == Object \cup Variable \cup BlankNode$$

A schema mapping can thus be represented as a two tuple in which the first part corresponds to the set of source patterns, and the second part corresponds to the set of target patterns. We then define the set of all schema mappings as the inner Cartesian product of the power set of patterns, that is:

$$SchemaMapping == \mathbb{P}\, Pattern \times \mathbb{P}\, Pattern$$

Note that these schema mappings may be easily transformed into SPARQL queries. For the sake of convenience, we define an instance of a class as a pattern of the form (*s, rdf:type, c*), in which $s \in Subject^?$, and $c \in Class$; and an instance of a property as a pattern of the form (*s, p, o*), in which $s \in Subject^?$, $p \in Property$, and $o \in Object^?$. Furthermore, we define the following projection functions:

$$\begin{array}{l} subject : Pattern \rightarrow Subject^? \\ predicate : Pattern \rightarrow Predicate \\ object : Pattern \rightarrow Object^? \\ \hline \forall s : Subject^?; p : Predicate; o : Object^?; t : Pattern \mid t = (s, p, o) \bullet \\ \quad subject(t) = s \land \\ \quad predicate(t) = p \land \\ \quad object(t) = o \end{array}$$

Note that $Triple \subseteq Pattern$, which implies that the previous projection functions can be applied to both triples and patterns. We also define the following ancillary functions:

$$\begin{array}{l} constants : \mathbb{P}\, Pattern \rightarrow \mathbb{P}\, Constant \\ entities : \mathbb{P}\, Pattern \rightarrow \mathbb{P}\, Entity \\ classes : \mathbb{P}\, Pattern \rightarrow \mathbb{P}\, Class \\ properties : \mathbb{P}\, Pattern \rightarrow \mathbb{P}\, Property \\ \hline \forall T : \mathbb{P}\, Pattern \bullet \\ \quad constants(T) = \{c : Constant \mid \exists t : Pattern \mid t \in T \bullet \\ \quad c = subject(t) \lor c = object(t)\} \land \\ \quad entities(T) = \{e : Entity \mid \exists t : Pattern \mid t \in T \bullet \\ \quad e = predicate(t) \lor e = object(t)\} \land \\ \quad classes(T) = \{c : Class \mid \exists t : Pattern \mid t \in T \bullet \\ \quad predicate(t) = rdf{:}type \Rightarrow c = object(t)\} \land \\ \quad properties(T) = \{p : Property \mid \exists t : Pattern \mid t \in T \bullet \\ \quad predicate(t) \neq rdf{:}type \Rightarrow p = predicate(t)\} \end{array}$$

Example 2. Fig. 1 (see the left side of Step 0) shows two sample sets of triples. In the left set, triples assert that *CEastwood* and *DEastwood* are instances of the *dp:Person* class. Furthermore, they assert that *CEastwood* has "*Clint East.*" and "*Clint*" as names by means of the *foaf:name* property, and that *CEastwood* is married to *DEastwood* by means of property *dp:spouse*.

Fig. 1 (see Step 5) shows three sample schema mappings in which the left side stands for the set of source patterns and the right side stands for the set of target patterns. Furthermore, $?p$, $?p_1$, and $?p_2$ are variables, and _:*X* is a blank node. m_{12} reclassifies instances of class *dp:Person* into instances of class *fp:person;* m_{21} exchanges instances of class *dp:Person* together with their names (property *foaf:name*) as instances of class *fp:person* and their names by means of property *ft:alias*; finally, m_{31} translates property *dp:spouse* into a more complex structure that comprises properties *fpp:spouse_s* and *fpm:spouse*, and a new instance of the *fp:marriage* class that is generated using a blank node, since this information is not present in the source.

4.3. Homomorphisms and exchange samples

A homomorphism maps the constants, variables, or blank nodes of a set of patterns onto the constants, variables, or blank nodes of another set of patterns. We formally define a homomorphism as a finite map from subjects or variables onto themselves, namely:

$$Homomorphism == Subject^? \twoheadrightarrow Subject^?$$

Homomorphisms specialise into replacements and substitutions. On the one hand, a replacement is a finite map from constants onto constants; on the other hand, a substitution is a finite map from constants onto variables or blank nodes. We formally define them as follows:

$$Replacement == Constant \twoheadrightarrow Constant$$
$$Substitution == Constant \twoheadrightarrow Variable \cup BlankNode$$

Generally speaking, a knowledge base is a set of triples. Regarding our proposal, we restrict our attention to the triples that are of the form (*c, rdf:type, C*), in which *c* is a constant and *C* is a class, or (c_1, *p*, c_2), in which c_1 and c_2 are constants and *p* is a property. Therefore, we define the set of knowledge bases as follows:

$$\begin{array}{l} KnowledgeBase == \{T : \mathbb{P}\, Triple \mid \\ \quad \forall t : Triple; c, c_1, c_2 : Constant; C : Class; p : Property \mid t \in T \bullet \\ \quad (t = (c, rdf{:}type, C) \Rightarrow c \in constants(T) \land C \in classes(T)) \lor \\ \quad (t = (c_1, p, c_2) \Rightarrow \{c_1, c_2\} \subseteq constants(T) \land p \in properties(T)) \end{array}$$

Note that the previous definition is not a shortcoming since we are able to model the most common triples in the general context of the Web of Data [31].

An exchange sample comprises a source knowledge base and a target knowledge base. We define the set of exchange samples as follows:

$$ExchangeSample == KnowledgeBase \times KnowledgeBase$$

For the sake of convenience, we define the following projection functions:

$$\begin{array}{l} source : ExchangeSample \rightarrow KnowledgeBase \\ target : ExchangeSample \rightarrow KnowledgeBase \\ \hline \forall d : ExchangeSample; T_S, T_T : KnowledgeBase \mid d = (T_S, T_T) \bullet \\ \quad source(d) = T_S \land \\ \quad target(d) = T_T \end{array}$$

Example 3. Fig. 1 (see the left side of Step 0) presents an exchange sample that comprises a set of source triples and another set of target triples. Furthermore, {*CEastwood*↦*CEastwood*, "*Clint East.*"↦"*Clint*"} is a sample replacement that maps *CEastwood* to *CEastwood*, and "*Clint East.*" to "*Clint*". Finally, {*CEastwood*↦$?p$, "*Clint East.*"↦$?n$} is a sample substitution that maps *CEastwood* to $?p$, and "*Clint East.* to $?n$.

4.4. Correspondences and data exchange problems

A correspondence is a hint that relates two sets of entities, i.e., they are defined at the data model level of a knowledge base. Note

```
1: algorithm    generateSchemaMappings
2: input        p : DataExchangeProblem
3: output       M : ℙ SchemaMapping
4: variables    D, D′ : ℙ ExchangeSample
5:
6: D := ∅
7: for each v : Correspondence | v ∈ correspondences(p) do
8:     — First step
9:     D′ := createCandidateExchangeSamples(v, sample(p))
10:    — Second step
11:    D := D ∪ discardCandidateExchangeSamples(D′)
12: end for
13: — Third step
14: D := completeExchangeSamples(D)
15: — Fourth step
16: D := pruneExchangeSamples(D)
17: — Fifth step
18: M := createSchemaMappings(D)
```

Fig. 2. Algorithm to generate schema mappings.

```
1: algorithm    createCandidateExchangeSamples
2: input        v : Correspondence; d : ExchangeSample
3: output       D : ℙ ExchangeSample
4: variables    G_S, G_T : ℙ KnowledgeBase; T_S, T_T : KnowledgeBase
5:
6: D := ∅
7: — Compute the triples that are related
8: — to correspondence v
9: G_S := computeRelatedTriples(source(v), source(d))
10: G_T := computeRelatedTriples(target(v), target(d))
11: — Compute the subknowledge bases that contain
12: — the triples that are related to v
13: for each T_S : KnowledgeBase | T_S ∈ ∏ G_S ∧
14:       #computeConnectedComponents(T_S) = 1 do
15:    for each T_T : KnowledgeBase | T_T ∈ ∏ G_T ∧
16:       #computeConnectedComponents(T_T) = 1 do
17:       D := D ∪ {(T_S, T_T)}
18:    end for
19: end for
```

Fig. 3. Algorithm to create candidate exchange samples.

that this entails that the correspondences with which we can deal are *n*:*m* correspondences. We define the set of correspondences as follows:

$$Correspondence == \mathbb{P}\, Entity \times \mathbb{P}\, Entity$$

For the sake of convenience, we define the following projection functions:

$$source : Correspondence \rightarrow \mathbb{P}\, Entity$$
$$target : Correspondence \rightarrow \mathbb{P}\, Entity$$

$$\forall c : Correspondence; E_S, E_T : \mathbb{P}\, Entity \mid c = (E_S, E_T) \bullet source(c) = E_S \land target(c) = E_T$$

A data exchange problem comprises a single exchange sample and a number of correspondences that relate entities in this exchange sample. We define the set of all data exchange problems as follows:

$$DataExchangeProblem == \{d : ExchangeSample; V : \mathbb{P} Correspondence \mid \forall v : Correspondence \mid v \in V \bullet source(v) \subseteq entities(source(d)) \land target(v) \subseteq entities(target(d))\}$$

For the sake of convenience, we define the following projection functions:

$$sample : DataExchangeProblem \rightarrow ExchangeSample$$
$$correspondences : DataExchangeProblem \rightarrow \mathbb{P} Correspondence$$

$$\forall d : ExchangeSample; V : \mathbb{P} Correspondence; p : DataExchangeProblem \mid p = (d, V) \bullet sample(p) = d \land correspondences(p) = V$$

Example 4. Fig. 1 (see the right side of Step 0) presents three sample correspondences that relate the entities that appear in the exchange sample, namely: v_1 relates a source entity with a target entity, v_2 relates two source entities with one target entity, and v_3 relates two source entities with three target entities. Both the single exchange sample and the correspondences form the data exchange problem that we use to illustrate our proposal (see Section 3).

5. Generating schema mappings

Our proposal takes a data exchange problem as input and generates a number of schema mappings to solve that problem without the intervention of a user. Fig. 2 presents the main algorithm of our proposal, which comprises five steps. The first step takes a correspondence and a single exchange sample as input, and it is responsible for generating a number of candidate exchange samples as output, each of which comprises a subset of source data that need to be exchanged as a whole and a subset of target data that need to be created as a whole. In the second step, our proposal ensures that for every candidate exchange sample there is, at least, a subset of target data that can be generated using the source data, and it minimises the target data that do not exist in the source. The third step takes the previous set of exchange samples as input and completes them, i.e., it ensures that it is possible to generate schema mappings using them. To perform this, our proposal adds new triples to the input exchange samples to guarantee that the same target data is generated by means of the same source data. The fourth step prunes those exchange samples that are redundant and, therefore, it is expected that they generate the same schema mappings. Finally, the fifth step takes the final set of exchange samples as input, and it transforms them into schema mappings by substituting source and target constants by fresh variables and/or blank nodes.

5.1. First step: create candidate exchange samples

Fig. 3 shows our algorithm to create candidate exchange samples from a given correspondence and a single exchange sample. Each candidate exchange sample comprises a subset of source data that need to be exchanged as a whole, and a subset of target data that need to be created as a whole. To compute them, for each correspondence, we combine all connected triples that comprise the entities in the correspondence.

First, we compute the triples that are related to correspondence v for the single exchange sample d, i.e., those triples that comprise the entities related by v. The results are stored in a set of knowledge bases. Then, we compute the distributive Cartesian product over the triples that are related to $source(v)$, and over the triples that are related to $target(v)$. This Cartesian product, which we denote as $\prod$, comprises all of the possible combinations in the set of knowledge bases, and it results in another set of knowledge bases. For instance, for the following set of knowledge bases $\{\{a, b, c\}, \{d, e\}, \{f\}\}$, where a, b, c, d, e, and f denote triples, the distributive Cartesian product is $\{\{a, d, f\}, \{a, e, f\}, \{b, d, f\}, \{b, e, f\}, \{c, d, f\}, \{c, e, f\}\}$.

We iterate over each set of source and target knowledge bases, and we transform them into exchange samples only if each knowledge base comprises a single connected component, i.e., every triple is related to every other triple, directly or indirectly.

We present the algorithm to compute the triples that are related to a set of entities in Fig. 4. It works on an input set of entities E and an input knowledge base T, and returns a set of knowledge bases that

```
algorithm   computeRelatedTriples
input       E : ℙ Entity; T : KnowledgeBase
output      G : ℙ KnowledgeBase
variables   T′ : KnowledgeBase

G := ∅
— Iterate over the entities
for each e : Entity | e ∈ E do
    T′ := ∅
    — Iterate over the knowledge base
    for each t : Triple | t ∈ T do
        — Add triple t if its predicate or object is equal to e
        if (e ∈ Property ∧ predicate(t) = e) ∨
                (e ∈ Class ∧ object(t) = e) then
            T′ := T′ ∪ {t}
        end if
    end for
    — Add the knowledge base to the final set
    G := G ∪ {T′}
end for
```

Fig. 4. Algorithm to compute the triples that are related to a set of entities.

```
algorithm   computeConnectedComponents
input       T : KnowledgeBase
output      CC : ℙ ℙ Constant
variables   N : ℙ Constant; A : ℙ (Constant × Constant)

N := ∅
A := ∅
— Iterate over the knowledge base
for each t : Triple | t ∈ T do
    — Add a node if t is a class
    if object(t) ∈ Class then
        N := N ∪ {subject(t)}
    — Add two nodes and an arc if t is a property
    else if predicate(t) ∈ Property then
        N := N ∪ {subject(t), object(t)}
        A := A ∪ {(subject(t), object(t))}
    end if
end for
— Find the connected components of graph (N, A)
CC := findConnectedComponents(N, A)
```

Fig. 5. Algorithm to compute the connected components of a knowledge base.

has all of the triples in T whose entities are in E. To perform this, we iterate over E and, for each entity, we iterate over T and check whether or not each triple is related to the current entity, i.e., if the entity is a class, it must be the object of the triple, whereas if it is a property, it must be the predicate of the triple. This set of triples is added to the final set.

The algorithm to compute the connected components of a set of triples is presented in Fig. 5. In the first part of the algorithm, we create an undirected graph in which set N stores the vertices, and set A stores the arcs of the graph. For each class instance, we add a vertex with the subject of the triple, and for each property instance, we add two vertices (the subject and the object), and an arc between them both. In the end, we use the *findConnectedComponents* algorithm to compute the connected components of this graph. We do not provide any additional details on this algorithm since it is well-known in the literature [40].

Example 5. To illustrate the generation of candidate exchange samples, we focus on correspondence v_2 in our running example. Its source entities are *dp*:*Person* and *foaf*:*name*. The triples that comprise *dp*:*Person* are the following:

(t_1) *CEastwood* *rdf*:*type* *dp*:*Person*

```
algorithm   discardCandidateExchangeSamples
input       D : ℙ ExchangeSample
output      D′ : ℙ ExchangeSample
variables   m : ℤ

D′ := ∅
— Compute the minimum number of
— uncovered constants
m := min{n : Z; d : ExchangeExample | d ∈ D ∧
    n = #(constants(target(d)) \ constants(source(d))) • n}
— Discard some exchange samples
for each d : ExchangeSample | d ∈ D do
    if #(constants(target(d)) ∩ constants(source(d))) > 0 ∧
        #(constants(target(d)) \ constants(source(d))) = m
    then
        D′ := D′ ∪ {d}
    end if
end for
```

Fig. 6. Algorithm to discard candidate exchange samples.

(t_2) *DEastwood* *rdf*:*type* *dp*:*Person*

Furthermore, the triples that comprise *foaf*: *name* are the following:

(t_3) *CEastwood* *foaf*:*name* *"Clint East."*

(t_4) *CEastwood* *foaf*:*name* *"Clint"*

Algorithm *computeRelatedTriples* outputs the following set in this case: $G_S = \{\{t_1, t_2\}, \{t_3, t_4\}\}$; the distributive Cartesian product of this set is $\prod G_S = \{\{t_1, t_3\}, \{t_1, t_4\}, \{t_2, t_3\}, \{t_2, t_4\}\}$. The target entity of v_2 is *ft*:*alias*, and the triples that comprise it are the following:

(t_6) *CEastwood* *ft*:*alias* *"Clint East."*

(t_7) *CEastwood* *ft*:*alias* *"Clint"*

Note that $G_T = \{\{t_6, t_7\}\}$, and $\prod G_T = \{\{t_6\}, \{t_7\}\}$. Additionally, in sets $\{\{t_2, t_3\}, \{t_2, t_4\}\} \subseteq \prod G_S$, each set has two connected components, since there is no triple that does not have any triple in common with at least another triple. Therefore, we discard these sets of triples.

Finally, candidate exchange samples are generated by combining the source triples in $\prod G_S$ and the target triples in $\prod G_T$. Some of them are depicted in our running example, e.g., $d_{21} = (\{t_1, t_3\}, \{t_6\})$, or $d_{22} = (\{t_1, t_4\}, \{t_6\})$.

5.2. Second step: discard candidate exchange samples

Fig. 6 shows our algorithm to discard candidate exchange samples. An exchange example is kept or discarded according to its number of covered and uncovered constants. A constant in the target is said to be covered if there is, at least, a triple in the source that involves that constant; otherwise, it is said to be uncovered. The algorithm first computes the minimum number of uncovered constants in the input set of exchange samples; it then iterates over this set and discards every exchange sample that does not have at least a covered constant or has more uncovered constants than the minimum.

The intuition behind this step is that we keep only the exchange samples that provide the maximum information to generate the target data, i.e., when these exchange samples are transformed into schema mappings, they shall comprise as less blank nodes as possible. Furthermore, this step allows us to disambiguate schema mappings, i.e., if several schema mappings may be generated for the same correspondence, we only generate those schema mappings that give us the maximum information to generate the target. Schema mapping disambiguation is a well-known issue when generating schema mappings [2], and we have devised the previously described simple, yet effective technique to solve it.

```
1: algorithm   computeReplacements
2: input       T1, T2 : KnowledgeBase
3: output      H : ℙ Replacement
4: variables   G : ℙ ℙ Equivalence; h : Replacement
5:
6: — Compute the equivalences between
7: — both knowledge bases
8: G := computeCandidateEquivalences(T1, T2)
9: H := ∅
10: — Iterate over the set of equivalences
11: for each Q : ℙ Equivalence | Q ∈ ∏ G ∧
12:     ∀c1, c2 : Constant | (c1, c2) ∈ Q •
13:         ∄c'1, c'2 : Constant | (c'1, c'2) ∈ Q •
14:             c1 = c'1 ⇒ c2 ≠ c'2 do
15:     h := ∅
16:     — Transform the equivalences into a
17:     — candidate replacement
18:     for each c1, c2 : Constant | (c1, c2) ∈ Q do
19:         h := h ∪ {c1 ↦ c2}
20:     end for
21:     — Apply the candidate replacement and
22:     — add it to the final set if the resulting
23:     — triples are included in T2
24:     if applyHomomorphism(h, T1) ⊆ T2 then
25:         H := H ∪ {h}
26:     end if
27: end for
```

Fig. 7. Algorithm to compute the replacements between two knowledge bases.

Example 6. In our running example, we discard exchange sample d_{11}, which results from correspondence v_1, since it has zero covered constants. Furthermore, the minimum number of uncovered constants in the exchange samples of v_2 is equal to zero, since every constant in d_{21} is covered; therefore, we discard exchange sample d_{22} because it has one uncovered constant: "*Clint East.*". Furthermore, the minimum number of uncovered constants in the exchange samples that result from correspondence v_3 is equal to one, since *fb*:*m*.02*kklj*5 is not present in the source of any exchange sample.

5.3. Third step: complete exchange samples

This step consists of completing exchange samples, i.e., if the same source data generate different target data in different exchange samples, it is necessary to complete those samples by adding target data. Therefore, we identify those exchange samples that have the same source data but differ in the target data, and we complete them.

Replacements lie at the heart of this step, so we present how we compute replacements in Fig. 7. This algorithm takes knowledge bases T_1 and T_2 as input and it first computes the candidate equivalences between them. Each candidate equivalence stands for a constant in T_1 that is related to another constant in T_2, and we formally define them as the following set of pairs:

$$Equivalence == Constant \times Constant$$

After computing these equivalences, we compute their distributive Cartesian product. If every candidate equivalence relates a given constant in T_1 to the same constant in T_2, we then transform the set of candidate equivalences into a candidate replacement h. Finally, if the triples that result from applying h to T_1 are included in T_2, we then add h to the output set, since this means that it is actually a replacement from T_1 to T_2.

Fig. 8 shows the algorithm to compute the candidate equivalences between two input knowledge bases T_1 and T_2. We iterate over the whole set of triples in T_1 and, for each triple t_1 in T_1, we use two different sets: in the first set, E_S, we store the candidate equivalences that relate the subject of t_1 to another constant in T_2; in the second set, E_O, we store those candidate equivalences that relate the object of t_1 to another constant in T_2. Then, for each triple t_2 in T_2 that has the same predicate as t_1, we update both sets of candidate equivalences if t_2 is a property instance, but only the first set if t_2 is a class instance. Finally, we update the output set with one or both sets.

```
1: algorithm   computeCandidateEquivalences
2: input       T1, T2 : KnowledgeBase
3: output      G : ℙ ℙ Equivalence
4: variables   ES, EO : ℙ Equivalence
5:
6: G := ∅
7: — Iterate over knowledge base T1
8: for each t1 : Triple | t1 ∈ T1 do
9:     ES := ∅
10:    EO := ∅
11:    — Iterate over knowledge base T2
12:    for each t2 : Triple | t2 ∈ T2 ∧
13:            predicate(t1) = predicate(t2) do
14:        — If their predicates are the same and both
15:        — triples are property instances
16:        if predicate(t2) ∈ Property then
17:            — Add two equivalences to each set
18:            ES := ES ∪ {(subject(t1), subject(t2)),
19:                        (subject(t1), object(t2))}
20:            EO := EO ∪ {(object(t1), object(t2)),
21:                        (object(t1), subject(t2))}
22:        — If their predicates and objects are the same
23:        else if object(t1) = object(t2) then
24:            — Add an equivalence to a set only
25:            ES := ES ∪ {(subject(t1), subject(t2))}
26:        end if
27:    end for
28:    — Update the output set
29:    if ES ≠ ∅ then
30:        G := G ∪ {ES}
31:    end if
32:    if EO ≠ ∅ then
33:        G := G ∪ {EO}
34:    end if
35: end for
```

Fig. 8. Algorithm to compute the candidate equivalences between two knowledge bases.

We apply a homomorphism to a knowledge base using the algorithm in Fig. 9, in which, for every triple, we transform its subject if it belongs to the domain of the homomorphism, we keep the predicate as is, and we transform the object if it belongs to the domain of the homomorphism. Note that this algorithm can be applied to both replacements and substitutions (see Section 4.3).

Example 7. To illustrate the computation of replacements, we first compute the candidate equivalences between *source*(d_{12}) and *source*(d_{21}) in our running example, which is the following set: $Q_1 = \{\{(DEastwood, CEastwood)\}\}$. Furthermore, $\prod Q_1 = \{\{(DEastwood, CEastwood)\}\}$ is the resulting distributive Cartesian product, which is automatically transformed into a candidate replacement since it relates the same source constant to the same target constant. This candidate replacement is: {*DEastwood*↦*CEastwood*}.

To illustrate the application of replacements, we apply the previous candidate replacement to *source*(d_{12}) in our running example, and we get the following triple: *CEastwood rdf*:*type dp*:*Person*, which is present in *source*(d_{21}); therefore, we conclude that {*DEastwood*↦*CEastwood*} is a replacement between *source*(d_{12}) and *source*(d_{21}).

The algorithm in Fig. 10 takes a set of exchange samples as input and outputs a number of complete exchange samples. It computes if the input set needs to be completed because the same source data

```
1: algorithm   applyHomomorphism
2: input       h : Homomorphism; T : KnowledgeBase
3: output      T' : ℙ Pattern
4: variables   s : Subject?; o : Object?
5:
6: T' := ∅
7: — Iterate over the knowledge base
8: for each t : Triple | t ∈ T do
9:    s := subject(t)
10:   o := object(t)
11:   — Update the subject
12:   if s ∈ dom h then
13:      s := h(s)
14:   end if
15:   — Update the object
16:   if o ∈ dom h then
17:      o := h(o)
18:   end if
19:   — Update the output
20:   T' := T' ∪ {(s, predicate(t), o)}
21: end for
```

Fig. 9. Algorithm to apply a homomorphism to a knowledge base.

```
1: algorithm   completeExchangeSamples
2: input       D : ℙ ExchangeSample
3: output      O : ℙ ExchangeSample
4: variables   H_S : ℙ Replacement; T_T : KnowledgeBase; restart : Boolean
5:
6: O := D
7: restart := true
8: — Iterate until no new triple is added
9: while restart do
10:   restart := false
11:   — Iterate over the whole set of exchange samples
12:   for each d1 : ExchangeSample | d1 ∈ O ∧ ¬restart do
13:      — Iterate again over the whole set of exchange samples
14:      for each d2 : ExchangeSample | d2 ∈ O ∧ d1 ≠ d2 ∧ ¬restart do
15:         — Compute the replacements between both exchange samples
16:         H_S := computeReplacements(source(d1), source(d2))
17:         — Iterate over the replacements
18:         for each h : Replacement | h ∈ H_S ∧ ¬restart do
19:            — Apply the replacement
20:            T_T := applyHomomorphism(h, target(d1))
21:            — If the new triples are not included
22:            if target(d2) ⊈ T_T then
23:               — Complete the exchange sample
24:               O := O \ {d2}
25:               O := O ∪ {(source(d2), target(d2) ∪ T_T)}
26:               restart := true
27:            end if
28:         end for
29:      end for
30:   end for
31: end while
```

Fig. 10. Algorithm to complete exchange samples.

generates different target data. To perform this, we compute the replacements between the exchange samples that have the same source data and, if they have some missing triples, we automatically add them to complete the target data. In this case, *restart* indicates if new triples have been added to the exchange samples, and we iterate until no new triple is added. We extract two different exchange samples d_1 and d_2 from the input set. We then compute the replacements between their source triples, and we apply each replacement to the target triples of d_1; if the resulting triples are not present in the target triples of d_2, we have to add them.

```
1: algorithm   pruneExchangeSamples
2: input       D : ℙ ExchangeSample
3: output      D' : ℙ ExchangeSample
4:
5: D' := ∅
6: — Iterate over the exchange samples
7: for each d1 : ExchangeSample | d1 ∈ D do
8:    — Compute the replacements between them
9:    if #D' = 0 ∨
10:     (∀d2 : ExchangeSample | d2 ∈ D' •
11:        computeReplacements(source(d1), source(d2)) = ∅ ∨
12:        computeReplacements(target(d1), target(d2)) = ∅ ∨
13:        computeReplacements(source(d2), source(d1)) = ∅ ∨
14:        computeReplacements(target(d2), target(d1)) = ∅)
15:   then
16:      — Add the exchange sample since it is
17:      — not redundant
18:      D' := D' ∪ {d1}
19:   end if
20: end for
```

Fig. 11. Algorithm to prune exchange samples that are redundant.

Example 8. To illustrate the completion process, we use exchange samples d_{12} and d_{21} from our running example. The unique replacement that exists between *source*(d_{12}) and *source*(d_{21}) is {*DEastwood*↦*CEastwood*} (see Example 7). Applying it to *target*(d_{12}) results in the following triple: *CEastwood rdf*:*type fp*:*person*, which is not included in *target*(d_{21}). Therefore, it is necessary to add this triple to *target*(d_{21}), and the exchange sample is completed as d'_{21}. The intuition behind this is that we have mapped an instance of *dp*:*Person* as *fp*:*person* in exchange sample d_{12}; however, in exchange sample d_{21}, we have an instance of *dp*: *Person* that is not mapped onto an instance of *fp*:*person*.

5.4. Fourth step: prune redundant exchange samples

Fig. 11 shows our algorithm to prune exchange samples that are redundant, i.e., they lead to the same schema mappings.

Replacements are used to detect if two exchange samples are redundant. Given two exchange samples d_1 and d_2, they are redundant if there exists four replacements from the source triples of d_1 to the source triples of d_2, from the target triples of d_1 to the target triples of d_2, from the source triples of d_2 to the source triples of d_1, and from the target triples of d_2 to the target triples of d_1, respectively. The algorithm depicted in Fig. 11 removes exchange samples that are redundant using the previous idea.

Example 9. In our running example, we have two different cases in which we prune redundant exchange samples. The first case is d_{12} and d_{13}, which have two replacements from *source*(d_{12}) to *source*(d_{13}) and viceversa, and other two replacements from *target*(d_{12}) to *target*(d_{13}) and viceversa. Therefore, we prune one of them randomly, e.g., d_{13}. The same occurs with d'_{31} and d'_{32}, and we prune one of them, e.g., d'_{32}.

5.5. Fifth step: transform exchange samples into schema mappings

Fig. 12 shows our algorithm to transform each exchange sample into a schema mapping, which is built by substituting source and target data by variables or blank nodes, depending on whether the target data is known or not.

To perform this, for each exchange sample, we retrieve its source and target constants. Then, we compute a source and a target substitution as follows: for every constant in the source, we add a fresh variable to both substitutions. For every constant in the target that is not in the source, we add a fresh blank node to the target substitution.

```
 1: algorithm    createSchemaMappings
 2: input        D : ℙ ExchangeSample
 3: output       M : ℙ SchemaMapping
 4: variables    C_S, C_T : ℙ Constant; h_S, h_T : Substitution;
 5:              T_S, T_T : ℙ Pattern; x : Variable
 6:
 7: M := ∅
 8: for each d : ExchangeSample | d ∈ D do
 9:    — Retrieve source and target constants
10:    C_S := constants(source(d))
11:    C_T := constants(target(d))
12:    — Compute source and target substitutions
13:    h_S := ∅
14:    h_T := ∅
15:    for each c : Constant | c ∈ C_S do
16:       x := freshVariable()
17:       h_S := h_S ∪ {c ↦ x}
18:       h_T := h_T ∪ {c ↦ x}
19:    end for
20:    for each c : Constant | c ∈ C_T \ C_S do
21:       h_T := h_T ∪ {c ↦ freshBlankNode()}
22:    end for
23:    — Apply substitutions
24:    T_S := applyHomomorphism(h_S, source(d))
25:    T_T := applyHomomorphism(h_T, target(d))
26:    — Update output set
27:    M := M ∪ {(T_S, T_T)}
28: end for
```

Fig. 12. Algorithm to create schema mappings from exchange samples.

Finally, we apply both substitutions to the source and target triples to generate the source and target patterns of the schema mapping.

Note that our proposal is able to generate more than one schema mapping for each input correspondence. These schema mappings are complementary, i.e., they comprise several combinations of triple patterns that generate valid target data when exchanging data, therefore, it is mandatory to take them all into account.

Example 10. To illustrate the transformation of exchange samples into schema mappings, in our running example, we transform exchange sample d'_{21} into schema mapping m_{21}. Our proposal computes the same source and target substitution, which is the following: {*CEastwood*↦?*p*, "*Clint East.*"↦?*n*}. Note that this algorithm does not include any additional items to the target substitution in this case since all of the target constants are already present in the source substitution; therefore, no blank nodes are generated.

We transform exchange sample d'_{31} into schema mapping m_{31}. Our proposal computes the following source substitution: {*CEastwood*↦?p_1, "*DEastwood*"↦?p_2}. Our proposal computes the following target substitution: {*CEastwood* ↦ ?p_1, "*DEastwood*" ↦ ?p_2, *fb*:*m*.02*kklj*5 ↦ _:*X*}, which comprises a blank node since constant *fb*:*m*.02*kklj*5 is not present in the source.

6. Analysis of our proposal

In this section, we analyse our proposal to prove that it has some desired properties, namely: it outputs GLAV mappings and they fit the input correspondences. We also characterise an upper-bound to its worst-case complexity. Finally, we discuss on some limitations that are interesting from a theoretical point of view, even though they are not a serious shortcoming in practice. Refer to Appendix A for the ancillary propositions.

6.1. Generation of GLAV schema mappings

We first prove that the schema mappings that our proposal generates are GLAV, which is important insofar this guarantees that they lead to universal solutions when they are used to perform data exchange. Universal solutions are the most general solutions to a data exchange problem.

Theorem 1 (Generation of GLAV schema mappings)**.** *The schema mappings that the createSchemaMappings algorithm outputs are GLAV.*

Proof. According to Alexe et al. [6], Fagin et al. [28], and Lenzerini [44], a GLAV schema mapping is a mapping whose logical interpretation is a first-order logic formula of the following form:

$$\forall \alpha \bullet \varphi(\alpha) \Rightarrow \exists \beta \bullet \psi(\alpha, \beta)$$

where $\varphi(\alpha)$ is a conjunction of atoms over the source knowledge base, each variable in α occurs in at least one atom in $\varphi(\alpha)$, and $\psi(\alpha, \beta)$ is a conjunction of atoms over the target knowledge base with variables from both α and β. By atom over a knowledge base they mean a formula $\gamma(\delta_1, \ldots, \delta_m)$, where γ is a predicate of the knowledge base, and $\delta_1, \ldots, \delta_m$ are variables, not necessarily distinct.

The schema mappings that Algorithm *createSchemaMappings* produce are of the form (T_S, T_T), where T_S and T_T are sets of patterns. According to Proposition 1, source patterns can be of the following forms:

- ?*x rdf*:*type C*, where ?*x* denotes a variable, and *C* denotes a source class.
- ?*x p* ?*y*, where ?*x* and ?*y* denote variables, and *p* denotes a source property.

Furthermore, according to Proposition 2, target patterns can be of the following forms:

- μ *rdf*:*type C*, where μ denotes a variable or a blank node, and *C* denotes a source class.
- $\mu\ p\ \nu$, where μ and ν denote variables or blank nodes, and *p* denotes a source property.

Patterns are interpreted in first-order logic as follows:

- μ *rdf*:*type C* is interpreted as $C(\mu)$, that is, classes are unary relations.
- $\mu\ p\ \nu$ is interpreted as $p(\mu, \nu)$, that is, properties are binary relations.

Let T'_S be the first-order logic interpretation of T_S, which is a conjunction of atoms over the source knowledge base. The variables that appear in T'_S are implicitly assumed to be universally quantified since they range over the source constants. Similarly, let T'_T be the first-order logic interpretation of T_T, which is a conjunction of atoms over the target knowledge base.

Let α be the set of variables in T'_S and α' the set of variables in T'_T. Then, $\alpha = \alpha'$ because every source constant that is present in the target is substituted by the same variable, as it can be seen in lines 15 and 19 in Algorithm *createSchemaMappings*.

Let β be the set of blank nodes in T'_T, which we can assume to be existentially quantified because they represent anonymous target data. Each blank node is assigned a fresh labelled null when these schema mappings are used to exchange data.

As a conclusion, our schema mappings can be represented as:

$$\forall \alpha \bullet \varphi(\alpha) \Rightarrow \exists \beta \bullet \psi(\alpha, \beta)$$

where $\varphi(\alpha)$ is T'_S and $\psi(\alpha, \beta)$ is T'_T. □

Corollary 1 (Generation of universal solutions)**.** *The schema mappings output by the createSchemaMappings algorithm produce universal solutions when they are used to exchange data.*

Proof. The proof follows very straightforwardly from the results by Fagin et al. [28]. According to them, if T'_S denotes the first-order logic interpretation of a source knowledge base T_S and M' the first-order logic interpretation of a GLAV schema mapping *M*, then a naive application of the Chase procedure generates a universal solution for

T_S with respect to M. Using the logical first-order interpretation that we propose in Theorem 1, it easily follows that the schema mappings output by Algorithm *createSchemaMappings* produce universal solutions when they are used to exchange data. □

6.2. Fitting correspondences

We prove that the schema mappings our proposal generates fit the input correspondences, which is important insofar this guarantees that they do not lose any information, i.e., they have all of the input correspondences into account [4]. A schema mapping m fits a correspondence v if it has a single connected component in the source, another single connected component in the target, for every entity e_s in $source(v)$, there is a triple pattern t_s in $source(m)$ such that e_s belongs to $entities(\{t_s\})$, and for every entity e_t in $target(v)$, there is a triple pattern t_t in $target(m)$ such that e_t belongs to $entities(\{t_t\})$.

Theorem 2 (Fitting correspondences)**.** *The schema mappings output by Algorithm generateSchemaMappings fit the input correspondences.*

Proof. Let p be a data exchange problem and M be a set of schema mappings output by $generateSchemaMappings(p)$. According to Propositions 3 and 4, each of the schema mappings in M comprises a unique connected component in the source and in the target. Furthermore, for every schema mapping $m \in M$ there is a correspondence $v \in correspondences(p)$ such that m fits $correspondences(p)$ since Algorithm *createCandidateExchangeSamples* extracts all of the possible combinations of those triples in the input exchange sample that belong to the entities of v. Finally, Algorithm *createSchemaMappings* transforms these triples into triple patterns. □

6.3. Analysis of complexity

In the following theorem and propositions, we characterise an upper-bound to the worst-case complexity of our proposal. The worst case is a data exchange problem in which each source triple of the input exchange sample introduces a new entity, and each target triple of the input exchange sample introduces a new entity.

In our proofs, we assume that simple set operations like invoking a projection function, checking for membership, merging two sets, or constructing a tuple can be implemented in $O(1)$ time with regard to the other operations. We also implicitly assume that data exchange problems must be finite, i.e., the sets of triples, entities, and correspondences involved are finite.

Theorem 3 (Generate schema mappings, see Fig. 2)**.** *Let p be a data exchange problem. An upper bound to the time a call to generateSchemaMappings(p) requires to terminate is $O(2^{t_s+t_t}\, t_s^{t_s}\, t_t^{t_t})$, where t_s and t_t denote the number of source and target triples in p, respectively. Furthermore, $2^{t_s+t_t}\, t_s^{t_s}\, t_t^{t_t}$ is an upper bound to the number of schema mappings that it outputs.*

Proof. Algorithm *generateSchemaMappings* iterates over the whole set of input correspondences. In each iteration, it calls Algorithms *createCandidateExchangeSamples* and *discardCandidateExchangeSamples* which, according to Propositions 5 and 8, require $O(e_s\, t_s + e_t\, t_t + t_s^{e_s} + t_s^{e_s}\, t_t^{e_t})$ time and $O(d)$ time, respectively, where e_s and e_t denote the number of entities of the input correspondences in p, t_s and t_t denote the number of source and target triples in p, and d denotes the number of exchange samples output by Algorithm *createCandidateExchangeSamples*. After this loop, it calls Algorithms *completeExchangeSamples*, *pruneExchangeSamples*, and *createSchemaMappings* in sequence, which, according to Propositions 12, 13, and 14 require $O(d^4\,(2t_s)^{t_s}\,(t_s^4 + t_t))$ time, $O(d^2\,(t_s^4\,(2t_s)^{t_s} + t_t^4\,(2t_t)^{t_t}))$ time, and $O(d\,(t_s + t_t))$ time, respectively.

The following formula is an upper bound to the worst-case time complexity of Algorithm *generateSchemaMappings*: $O(v\,(e_s\, t_s + e_t\, t_t + t_s^{e_s} + t_s^{e_s}\, t_t^{e_t} + d) + d^4\,(2t_s)^{t_s}\,(t_s^4 + t_t) + d^2\,(t_s^4\,(2t_s)^{t_s} + t_t^4\,(2t_t)^{t_t}) + d\,(t_s + t_t))$.

In the worst case, $e_s = t_s$ and $e_t = t_t$, which entails that every source and target triple comprises a different entity. Additionally, $d = v\, t_s^{t_s}\, t_t^{t_t}$, since Algorithm *createCandidateExchangeSamples* generates $t_s^{t_s}\, t_t^{t_t}$ exchange samples for each correspondence, in the worst case. Furthermore, $v = 2^{t_s+t_t} - 2^{t_s} - 2^{t_t} - 1$ since the worst case is a case in which the correspondences relate every possible combinations of source and target entities; it is easy to check that $v < 2^{t_s+t_t}$.

As a conclusion, the previous upper bound can be simplified as follows: $O(2^{t_s+t_t}\, t_s^{t_s}\, t_t^{t_t})$. Additionally, an upper bound to the number of schema mappings that this algorithm outputs is equal to $2^{t_s+t_t}\, t_s^{t_s}\, t_t^{t_t}$. □

6.4. Limitations

Despite the fact that we deal with *n*:*m* correspondences, our proposal cannot deal with more than one instance of the same class, e.g., an *Employee* that is related to another *Employee* by the *boss* property. A similar problem occurs when the same property has to be used more than once in the same schema mapping. These limitations are due to the fact that correspondences do not state if an entity should appear one or more times in each schema mapping. However, in the real-world data exchange problems that we have evaluated, we have not found this to be a practical limitation.

Another limitation of our proposal is that it does not generate schema mappings that include patterns with regular expressions [7]. This implies that we are not able to deal with RDF collections, such as bags, lists, or sequences. However, this limitation must not hinder the applicability of our proposal in practice since these constructs are not recommended when publishing RDF data as Linked Data [39]. Furthermore, according to Glimm et al. [31], RDF collections do not range amongst the most used constructs in the Web of Data.

Another limitation is that, in its current form, our proposal is not able to incorporate literals in the schema mappings, which are mandatory for certain data exchange problems. For instance, if we wish to exchange people that was born in Spain, then it is necessary to include Spain as a literal in the schema mappings. Algorithm *createSchemaMappings* transforms every source literal into a variable (see Fig. 12), therefore, to deal with this issue, this algorithm can be modified to take a list of immutable literals as input, which are not transformed into variables.

In the literature, some approaches generate schema mappings only if they fit the input exchange samples [4]. This entails that, when we exchange the source data of the exchange samples using these schema mappings, the generated target data is equivalent to the target data of the exchange samples. If the input exchange samples are not appropriate, the user has to modify them to generate schema mappings. In our proposal, we allow the user to freely define the input exchange sample, so it is not mandatory that this exchange sample fits the output schema mappings. Our proposal implements a best-effort strategy; if the resulting schema mappings do not fit the input exchange sample, then the implementation warns the user, who has to decide whether the input must be modified or not. This warning is triggered when the input exchange sample comprises several source triples for the same property that are not present in the target. For instance, assume that we remove the following target triple of the input exchange sample in our running example: *CEastwood ft*:*alias* "*Clint*". Our proposal generates exactly the same schema mappings, which entails that every source triple that comprises property *foaf*:*name* has to be exchanged as a target triple that comprises property *ft*:*alias*. Note that this behaviour is not explicitly specified in the input exchange sample, but our proposal assumes that it is the user intention.

7. Validity evaluation

To validate our proposal, we used some data exchange problems for which we handcrafted schema mappings in previous projects. We run our proposal on them and then compared the subsets of source and target triples that were exchanged by our previous handcrafted schema mappings and the ones output by our proposal. We ensure the reproducibility of our experimental results by making our RDF dumps, the script to preprocess them, our repository, the implementation of our algorithms, the scripts to perform the evaluation process, and a user manual publicly available [60]. Thanks to this, our results can be reproduced and tested by third parties, which is crucial for the advance of science [30]. Furthermore, other researchers can extend our results to cope with future requirements.

7.1. Repository

To the best of our knowledge, little effort has been paid to devising repositories to validate the automatic generation of schema mappings. Alexe et al. [3] devised a benchmark that provides eleven data exchange patterns, each of which can be instantiated into a number of data exchange problems using a number of parameters. Unfortunately, their focus was on nested-relational data models, which renders them not appropriate for our context. In the RDF context, Rivero et al. [62] used a repository that comprises four real-world data exchange problems and 3 780 synthetic data exchange problems that were generated using MostoBM [63]. Unfortunately, these problems focus on the automatic generation of schema mappings based on source and target data models, therefore, the authors did not provide any exchange examples.

To address this issue, we have setup a repository of ten real-world data exchange problems on which we worked in the past. For each data exchange problem, our repository provides a set of handcrafted schema mappings that are expected to perform data exchange appropriately and source data to perform data exchange. The knowledge bases that we use in our repository are the following:

- Freebase [19]: In our repository, we use a dump provided by the authors [33]. Its structure is very rich, for instance, to model that a person is married to another person, property *fpp*:*spouse_s* relates an instance of class *fp*:*person* with an instance of class *fp*:*marriage*, which contains the data of a marriage, such as the spouses, the date of the ceremony, the type of the union (e.g., marriage or domestic partnership), or the location of the ceremony. The consequence of this rich structure when exchanging data is that it is necessary to perform a deep navigation to have access to the desired data. Another consequence is that, when Freebase is the target in a data exchange problem, if the source knowledge base does not have such a rich structure, then it is necessary to generate blank nodes.
  Furthermore, Freebase uses a large number of properties to model data that are very little reused. For instance, properties *fpp*:*place_of_birth* and *fll*:*people_born_here* model the same data, but the first property relates an instance of class *fp*:*person* with an instance of class *fl*:*location*, and the second property models the inverse. The consequence of this large number of properties when exchanging data is that we have to define several correspondences to generate schema mappings.
- DBpedia [17]: It comprises a number of different versions due to a number of changes in its conceptualisation. In our repository, we use the dump of DBpedia 3.8 that comprises Ontology Infobox Types and Ontology Infobox Properties [25]. The structure of DBpedia is not as rich as the structure of Freebase. For instance, property *dp*:*spouse* is used to state that a person is married to another person. The consequence of this structure when exchanging data is that it is not necessary to perform a deep navigation to have access to the desired data.
  Additionally, DBpedia has several domain and/or range classes for each property. This entails that the same property may relate instances of very different classes. For instance, property *dp*:*author* relates instances of class *dp*:*Person* with instances of class *dp*:*Book*, which stands for "a person is the author of a book"; the same property is also used to relate instances of class *dp*:*Software* with instances of class *dp*:*Organisation*, which means that "an organisation has developed a piece of software". Note that both examples convey the idea of "authorship", but they are used in quite different contexts. The consequence of several domain and/or range classes when exchanging data is that it is necessary to consider the classes of the related instances when dealing with properties.
- GovWILD [18]: It is a public RDF knowledge base that interconnects government data and provides a web-based application that enables exploring them. GovWILD comprises economic data from the USA and other European governments. In our repository, we use the dump provided by the authors [34].
  The structure of GovWILD is not as rich as the structure of Freebase, which entails that it is not necessary to perform deep navigation to have access to the desired data when performing data exchange. There is only one exception: date objects are modelled as individual instances that have several properties, such as *gw*:*day*, *gw*:*month*, and *gw*:*year*. Therefore, GovWILD comprises a date instance for each date that is present in its data. The consequence of this when exchanging data is that it is necessary to use blank nodes to create these dates when GovWILD is used as the target.

The data exchange problems that our repository comprise are the following:

- DF-P, FD-P: They stand for "DBpedia to Freebase (People)" and "Freebase to DBpedia (People)". The goal is to exchange data that are related to people, such as their names, children, birth dates, birth places, nationalities, marriages, or death causes.
- DF-TS, FD-TS: They stand for "DBpedia to Freebase (Television Shows)" and "Freebase to DBpedia (Television Shows)". The goal is to exchange data that are related to television shows, such as episodes, stations, networks, directors, actors, producers, writers, seasons, or fictional characters.
- DF-F, FD-F: They stand for "DBpedia to Freebase (Films)" and "Freebase to DBpedia (Films)". The goal is to exchange data that are related to films, such as film music composers, producers, directors, actors, producers, writers, film budgets, editors, film runtimes, or fictional characters.
- DF-U, FD-U: They stand for "DBpedia to Freebase (Universities)" and "Freebase to DBpedia (Universities)". The goal is to exchange data that are related to universities, such as sport teams, university colours, mascots, students, faculty people, staff people, cities, or endowments.
- DG, GD: They stand for "DBpedia to GovWILD" and "GovWILD to DBpedia". The goal is to exchange data that are related to politicians, such as their names, nationalities, political parties, education, or religion.

It is important to notice that we do not use the whole dump of each knowledge base as source data, but only a part of the dump that comprises the classes and properties involved in the correspondences. Furthermore, without an exception, the knowledge bases in this context comprise a number of errors, e.g., DBpedia and Freebase reference web pages whose URLs are "<None>", which are malformed URLs. Therefore, for each data exchange problem, we preprocess the source knowledge base by cleaning the data, and extracting only the triples that contain the classes and properties that are involved in that problem. We have implemented a script to preprocess

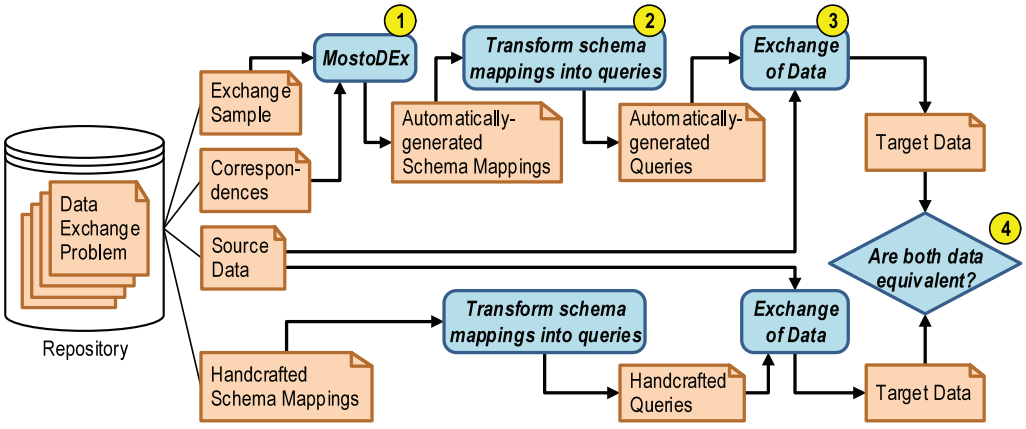


Fig. 13. Process to validate our proposal.

```
CONSTRUCT {
  ?p1  rdf:type        fp:person;
       fpp:spouse_s _:X.

  _:X  rdf:type        fp:marriage;
       fpm:spouse      ?p1,
                       ?p2.
} WHERE {
  ?p1  rdf:type        dp:Person;
       dp:spouse       ?p2.        }
```

Fig. 14. Sample SPARQL query.

DBpedia, Freebase, and GovWILD RDF dumps using Java 1.6, Sesame 2.6.10 [20], and OWLIM Lite 4.2 [14].

7.2. Evaluation process

Fig. 13 presents the evaluation process of our validation, which comprises four steps, namely:

1. We use our proposal to automatically generate a set of schema mappings based on the single exchange sample and the set of correspondences of a specific data exchange problem. We implemented a research prototype, to which we refer to as MostoDEx, using Java 1.6 and Jena 2.7.3 [23]. Furthermore, we used Guava 13.0.1 to implement ancillary set operations, and JGraphT 0.8.3 to compute the connected components of a set of patterns.
2. We transform each schema mapping into a query mapping in SPARQL. This step is mandatory since our goal is to use a query engine to perform data exchange. Fig. 14 shows a sample transformation from schema mapping m_{31} in our running example into a SPARQL query, in which we use the target patterns of the schema mapping as the CONSTRUCT clause, and the source patterns as the WHERE clause.
3. We exchange the source data by means of both automatically-generated and handcrafted queries. Exchanging data between RDF knowledge bases comprises five steps [63], namely: (1) a mandatory step that consists of loading the source knowledge base from persistent storage into the appropriate internal data structures; (2) an optional step that consists of making the knowledge explicit in the source knowledge base; (3) a mandatory step that consists of executing the queries over the source knowledge base to produce a target knowledge base; (4) an optional step that consists of making the knowledge explicit in the target knowledge base; (5) a mandatory step that consists of saving the target knowledge base to a persistent storage. In this article, we omit steps (2) and (4) since we deal with plain RDF, without taking RDFS or OWL entailments into account. Furthermore, steps (1) and (5) largely depend on the technology being used, and our goal in this article is the generation of schema mappings; therefore, we focus only on step (3).
4. We validate whether or not both the target data output by the automatically-generated and handcrafted schema mappings are equivalent, i.e., if every triple in the first set of target data is present in the second set of target data, and vice versa. Note that anonymous data are implemented as blank nodes in the context of SPARQL. Since both target data are stored in different files, blank nodes cannot be compared because they are local to the corresponding files [22]. To address this problem, there are approaches in the literature that use graph isomorphisms to check if two RDF knowledge bases with blank nodes are equal. The Jena framework has an efficient implementation of one such approach [22], therefore, we used Jena TDB 0.9.3, which supports large scale storage and uses the file system to store RDF knowledge bases.

We have implemented the evaluation process of our validation in a script that uses Java 1.6, Jena TDB 0.9.3, Sesame 2.6.10, and OWLIM Lite 4.2. This process was run on a virtual computer that was equipped with a four-threaded Intel Xeon 3.00 GHz CPU and 16GB RAM, running on Windows Server 2008 (64-bits).

7.3. Evaluation results

Table 2 summarises our experimental results. The columns represent the data exchange problems of our repository, and the rows a number of measures; the first group of measures provides an overall idea of the size of each data exchange problem, i.e., the number of source and target triples of the initial examples, the correspondences between the entities, and the number of classes and properties involved in the correspondences. The second group of measures provides information about the schema mappings, i.e., the number of automatically-generated schema mappings, and the time that our proposal took to generate them in seconds. Finally, the third group of measures provides an overall idea about the exchange of data by means of these schema mappings, i.e., the number of source triples in millions, the number of target triples generated in millions, and the time the automatically-generated schema mappings took to perform the data exchange in minutes.

The target data generated by the automatically-generated schema mappings were the same as the target data generated by handcrafted schema mappings in every data exchange problem. This reveals that the schema mappings that our proposal generates agree with the schema mappings that we handcrafted in the past to solve these data exchange problems. The time our proposal took to generate the

Table 2
Experimental results of our proposal.

	DF-P	FD-P	DF-TS	FD-TS	DF-F	FD-F	DF-U	FD-U	DG	GD
Input data (single exchange sample and correspondences)										
Source triples	33	36	28	38	19	48	23	46	27	13
Target triples	36	33	38	28	48	19	46	23	13	27
Correspondences	20	20	26	27	17	19	14	13	15	18
Source classes	14	8	27	17	17	7	14	6	18	13
Source data properties	9	9	9	9	5	4	8	5	13	14
Source object properties	9	9	15	25	11	21	6	13	9	3
Target classes	14	13	17	28	17	10	11	8	11	21
Target data properties	9	9	9	9	5	4	8	5	13	14
Target object properties	11	9	24	15	28	13	17	8	1	11
Output data (schema mappings)										
Number	18	16	23	26	15	18	14	13	15	11
Time	0.67s	0.59s	0.53s	0.59s	0.64s	0.58s	0.58s	0.56s	0.52s	0.47s
Application of our schema mappings (data exchange)										
Source triples	14.28M	56.89M	12.75M	53.23M	12.57M	50.51M	51.91M	60.96M	14.37M	7.48M
Target triples	3.94M	16.00M	0.55M	5.06M	2.49M	2.90M	0.33M	0.84M	3.15M	1.27M
Time	2.16m	12.06m	0.70m	23.86m	1.13m	16.32m	0.38m	0.71m	2.12m	0.64m

DF-P = DBpedia to Freebase (People); FD-P = Freebase to DBpedia (People); DF-TS = DBpedia to Freebase (Television Shows); FD-TS = Freebase to DBpedia (Television Shows); DF-F = DBpedia to Freebase (Films); FD-F = Freebase to DBpedia (Films); DF-U = DBpedia to Freebase (Universities); FD-U = Freebase to DBpedia (Universities); DG = DBpedia to GovWILD; GD = GovWILD to DBpedia.

schema mappings was less than one second in every case; since timings are imprecise in nature, we repeated each experiment 25 times and selected the maximum value within the results.

We also measured the time that automatically-generated schema mappings took to exchange data. Although these timings depend largely on the technology being used, i.e., the RDF technologies used to persist triples and the SPARQL query engine used to exchange data, we think that presenting them is appealing insofar they suggest that the queries can be executed on reasonably-large knowledge bases in a sensible time. Recall that we only focus on the time to execute the queries and not on the loading, reasoning, or unloading time since they are not related to the focus of our proposal.

8. Scalability evaluation

This section analyses the scalability of our proposal. In Section 6.3, we have characterised an upper bound to its complexity, according to which it is computationally complex. In this section, we prove that this does not hinder its applicability in practice. We ensure the reproducibility of our experimental results by making our repository, the scripts to perform the evaluation process, our experimental results, and a user manual publicly available [60]. Thanks to this, our scalability results can be reproduced and tested by third parties.

8.1. Repository

To the best of our knowledge, little effort has been paid to evaluating the scalability of schema mapping proposals in the context of RDF. On the one hand, LODIB [59] is a benchmark to evaluate the performance of exchanging data in the context of Linked Data. Unfortunately, it provides three data exchange problems whose entities and correspondences cannot be scaled, only the data they comprise. On the other hand, MostoBM [63] provides seven data exchange patterns that are instantiated into a number of data exchange problems using some parameters, and we decided to use them to evaluate our proposal.

The data exchange patterns that MostoBM provides are the following: Lift Properties, Sink Properties, Extract Subclasses, Extract Superclasses, Extract Related Classes, Simplify Specialisation, and Simplify Related Classes. Extract Related Classes and Simplify Related Classes cannot be used to perform our evaluation since they use *n*:*m* correspondences that use transformations functions, which are not supported by our proposal. Furthermore, the goal of Simplify Specialisation is to test the performance of making the knowledge explicit in the source and target knowledge bases, which is not supported in our proposal. The rest of the data exchange patterns are described as follows:

- Lift Properties: The data properties of a set of subclasses are moved to a common superclass.
- Sink Properties: The data properties of a superclass are moved to a number of subclasses.
- Extract Subclasses: A source class is split into several subclasses and the domain of the target data properties is selected amongst the subclasses.
- Extract Superclasses: A class is split into several superclasses, and data properties are distributed amongst them.

Data exchange patterns are instantiated into data exchange problems using seven parameters that allow to scale both the entities and the data of a knowledge base. Since our intention is to evaluate the behaviour of our proposal when entities and correspondences scale, we only focus on a subset, namely:

- Levels of classes (L): Number of relationships (specialisations or object properties) amongst one class and the rest of the classes in

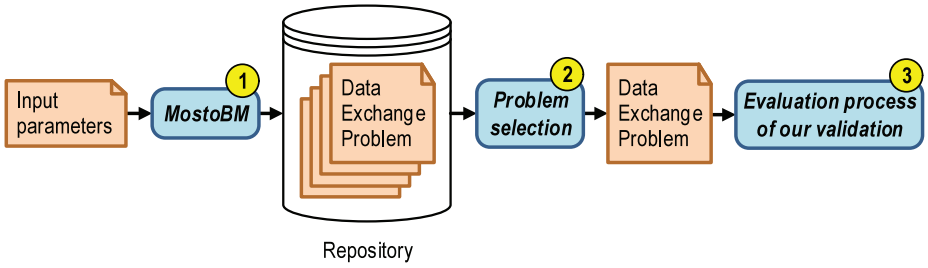
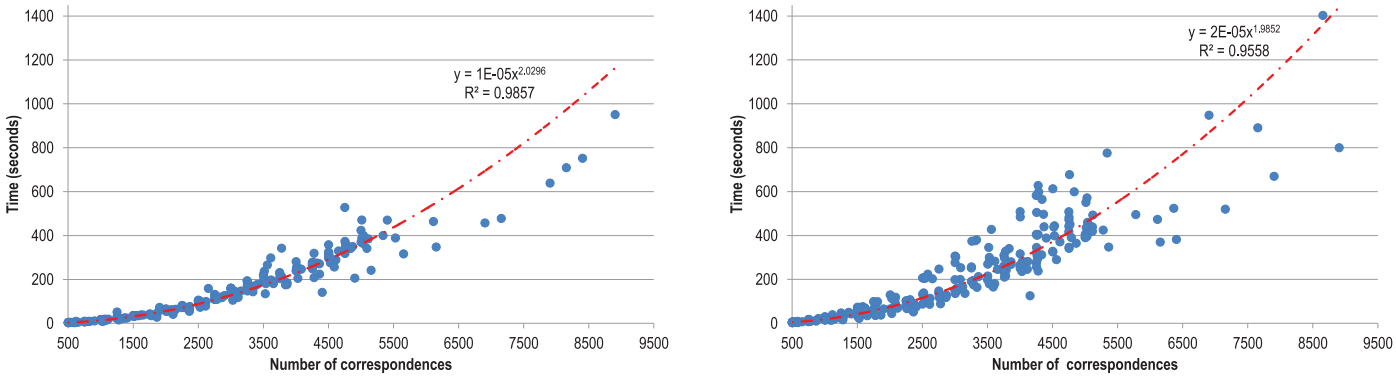


Fig. 15. Process to evaluate the scalability of our proposal.



(a) Lift Properties. (b) Sink Properties.

Fig. 16. Scalability results of two data exchange patterns.

the source or target knowledge bases. L allows to scale the structure of these knowledge bases in depth.

- Number of related classes (C): Number of classes related to each class by specialisation or object properties. C allows to scale the structure of these knowledge bases in breadth.
- Number of data properties (D) of the source and target knowledge bases.

In our experiments, we used the following values for the previous parameters to instantiate our data exchange problems: $L = \{1, 2, 3, 4, 5\}$, $C = \{1, 2, 3, 4, 5\}$, and $D = \{250, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500, 2750, 3000, 3250, 3500, 3750, 4000, 4250, 4500, 4750, 5000\}$. We set the rest of the parameters that deal with the scaling of the data to the following values: $I = 5000$, $I_T = 5$, $I_D = 5$, and $I_O = 5$. As a conclusion, we instantiated 500 data exchange problems for each data exchange pattern. Each of these data exchange problems comprise a number of schema mappings to perform data exchange, a number of correspondences, and a populated source knowledge base with synthetic data. Therefore, we added a new functionality to MostoBM to generate a source and a target exchange sample to evaluate our proposal. To perform this, we generate the source exchange sample by generating a single instance for each source entity and, using the schema mappings generated by MostoBM, we get the target exchange sample by exchanging the source data.

To provide an overall idea of the size of these data exchange problems, the number of source and target classes ranges between 3 and 7812, the number of source and target data properties ranges between 250 and 5000, and the number of source and target object properties ranges between 2 and 7811. The conclusion is that these problems are synthetic but not trivial in general.

8.2. Evaluation process

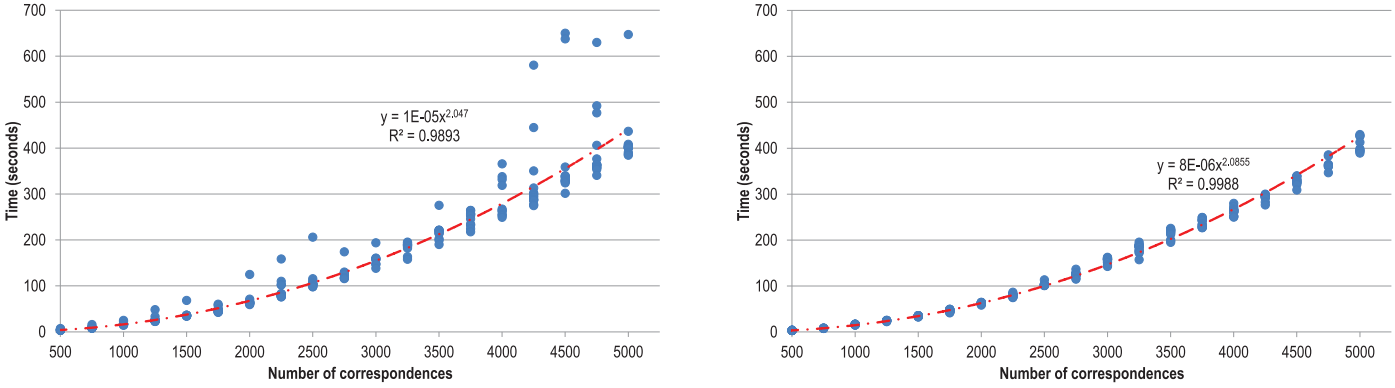
Fig. 15 presents the evaluation process of the scalability of our proposal, which comprises three steps, namely:

1. We use MostoBM to generate the repository of data exchange problems, which results from instantiating a set of data exchange patterns with several values of the input parameters.
2. After generating the repository, it is necessary to run the data exchange problems that it comprises. Running a single data exchange problem may take hours or even days to complete, since it is executed 25 times (see below). This makes it necessary to use the Monte Carlo method to select a subset of data exchange problems to execute randomly. The issue that remains is to determine the size of this subset. To provide a precise figure on the number of data exchange problems to run, we rely on Cochran's formula [24], which is based on the variance of the performance variable, i.e., the time that our proposal takes to output schema mappings. In the worst case, when the variance being studied is very high, this number is equal to $|P|/2$, in which $|P|$ is the total number of data exchange problems for a given data exchange pattern. Therefore, we execute $250 = 500/2$ problems for each data exchange pattern.
3. Finally, we perform the evaluation process of our validation to actually run the data exchange problems (see Section 7.2).

We have implemented this evaluation process in a script that uses Java 1.6, and Jena TDB 0.9.3, which is documented by means of a user manual. Our experiments were run on a virtual computer that was equipped with a four-threaded Intel Xeon 3.00 GHz CPU and 16GB RAM, running on Windows Server 2008 (64-bits).

8.3. Evaluation results

Figs. 16 and 17 present our evaluation results, in which we compare the time our proposal took to generate the schema mappings in the data exchange problems to the number of correspondences in each data exchange problem. Note that, in these problems, the number of source and target triples of the single exchange sample are equal to the number of source and target entities, and they are also equal to the number of correspondences, which are the variables in



(a) Extract Subclasses. (b) Extract Superclasses.

Fig. 17. Scalability results of two data exchange patterns.

our theoretical complexity study (see Section 6.3). Since the number is the same, we only focus on correspondences in this evaluation. Note also that the scaling of correspondences is not linear since the correspondences in the data exchange problems that MostoBM generates are not linear regarding the parameters.

Since timings are imprecise in nature, we repeated each experiment 25 times and averaged the results after discarding roughly 0.01% outliers using the well-known Chevischev's inequality. For each data exchange problem, we checked that the target data that result from exchanging data using the schema mappings of MostoBM were equivalent to exchanging data with our automatically generated schema mappings.

From our experimental results we can draw the following conclusions:

- The behaviour of Lift Properties and Sink Properties is similar, as it was also the case for Extract Subclasses and Extract Superclasses.
- We also computed the minimum squared error tendency line, that is, the one that maximises the R^2 coefficient, and found out that the behaviour is nearly quadratic in every case.

9. Conclusions

In the literature, there are a number of proposals to generate schema mappings that rely on data models that must provide entities and constraints. They are not appealing in the general context of the Web of Data because some of them rely on handcrafting the mappings; some others rely on exploiting the constraints in the data models and the usual knowledge bases in this context have a few or no constraints at all; and a few others rely on exchange samples but require user intervention, or are hybrid and require constraints to be available.

In this article, we present a proposal to automatically generate schema mappings amongst RDF knowledge bases using a single exchange sample and a set of n: m correspondences. It does not rely on constraints of the source and target data models and does not require any user intervention. We have validated our proposal using ten data exchange problems amongst the DBpedia, Freebase, and GovWILD knowledge bases. The time to execute this validation never exceeded one second, and the data exchanged were as expected in every case, which suggest that it is very efficient in practice and that the generated schema mappings are appropriate. We have also evaluated the performance of our proposal when data exchange problems scale. To perform this, we have used four synthetic data exchange patterns proposed by MostoBM, a benchmark for testing data exchange proposals in the context of the Web of Data. Synthetic data exchange patterns are instantiated into 2 000 data exchange problems that we have used to evaluate our proposal. Our evaluation results suggest that it works quite well as the data exchange problems it faces scale.

We have proved that the schema mappings we produce are GLAV. Furthermore, we have proved that the schema mappings output by our proposal fit the input correspondences. We have implemented a research prototype that is publicly available, together with our repository of data exchange problems, the scripts to validate and evaluate the scalability of our proposal, and all of the experimental data regarding the validation and the performance presented in this article. Our goal is twofold: on the one hand, it allows other researchers to faithfully reproduce our experiments, which is crucial for the advance of science; on the other hand, our implementation, repository, and experimental data can be extended to cope with future requirements.

Acknowledgements

The work that we present in this article was supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes with the following Grants: TIN2007-64119, P07-TIC-2602, P08-TIC-4100, TIN2008-04718-E, TIN2010-21744, TIN2010-09809-E, TIN2010-10811-E, TIN2010-09988-E, TIN2011-15497-E, and TIN2013-40848-R.

Appendix A. Ancillary propositions

In this section we present some ancillary propositions that we used to support our theorems.

A1. Generation of GLAV schema mappings

Proposition 1 (Types of source patterns)**.** *A source pattern output by Algorithm createSchemaMappings can only be of the following forms:*

- *?x rdf:type C, where ?x denotes a variable, and C denotes a source class.*
- *?x p ?y, where ?x and ?y denote variables, and p denotes a source property.*

Proof. Lines 15 and 19 in Algorithm *createSchemaMappings* create a substitution for each constant in the source input triples that transforms it into a fresh variable. Then, in line 24, this substitution is applied to the source triples. Our proposal only works with two types of triples (see Section 4.3), which are the following:

- *c rdf:type C*, where *c* denotes a constant, and *C* denotes a source class. This triple is substituted by *?x rdf:type C*, where *?x* denotes a variable.

- $c_1\ p\ c_2$, where c_1 and c_2 denote constants, and p denotes a source property. This triple is substituted by $?x\ p\ ?y$, where $?x$ and $?y$ denote variables. □

Proposition 2 (Types of target patterns)**.** *A target pattern output by Algorithm createSchemaMappings can only be of the following forms:*

- *μ rdf:type C, where μ denotes a variable or a blank node, and C denotes a source class.*
- *$\mu\ p\ \nu$, where μ and ν denote variables or blank nodes, and p denotes a source property.*

Proof. Lines 15 and 19 in Algorithm *createSchemaMappings* create a substitution for constants that are shared by the source and target triples, which are substituted by the same fresh variables. Furthermore, lines 20 and 22 create a substitution for target constants that are not present in the source, which are substituted by the same blank nodes. Our proposal only works with two types of triples (see Section 4.3), which are the following:

- *c rdf:type C*, where c denotes a constant, and C denotes a source class. This triple is substituted by *μ rdf:type C*, where μ denotes a variable if c is present in the source, or a blank node if it is not present.
- $c_1\ p\ c_2$, where c_1 and c_2 denote constants, and p denotes a source property. This triple is substituted by $\mu\ p\ \nu$, where μ and ν denote variables if c_1 and c_2 are present in the source, or blank nodes if they are not present. □

A2. Fitting correspondences

Proposition 3 (Source connected components)**.** *A schema mapping output by Algorithm generateSchemaMappings comprises a unique connected component in the source.*

Proof. Algorithm *createCandidateExchangeSamples* generates a number of exchange samples, each of which comprises a unique connected component in the source, which is guaranteed by enforcing this condition (see Line 14). □

Proposition 4 (Target connected components)**.** *A schema mapping output by Algorithm generateSchemaMappings comprises a unique connected component in the target.*

Proof. Algorithm *createCandidateExchangeSamples* generates a number of exchange samples, each of which comprises a unique connected component in the target, which is guaranteed by enforcing this condition (see Line 16). □

A3. Analysis of complexity

Proposition 5 (Create candidate exchange samples, see Fig. 3)**.** *Let v and d be a correspondence and an exchange sample, respectively. An upper bound to the time a call to createCandidateExchangeSamples(v, d) requires to terminate is $O(e_s\, t_s + e_t\, t_t + t_s^{e_s} + t_s^{e_s}\, t_t^{e_t})$, where t_s denotes the number of source triples in d, t_t denotes the number of target triples in d, e_s denotes the number of source entities in v, and e_t denotes the number of target entities in v. In the worst case, the number of exchange samples that this algorithm computes is equal to $t_s^{e_s}\, t_t^{e_t}$.*

Proof. Algorithm *createCandidateExchangeSamples* performs two calls to Algorithm *computeRelatedTriples*, which, according to Proposition 6, terminates in $O(e\,t)$ time, where e denotes the number of entities and t denotes the number of triples. Therefore, these two calls terminate in $O(e_s\, t_s + e_t\, t_t)$ time. Then, it iterates $t_s^{e_s}$ times over the distributive Cartesian product of G_S. In each iteration, we check if the set of triples has a single connected component, which, according to Proposition 7, terminates in $O(t_s)$ time. Additionally, this algorithm iterates over the distributive Cartesian product of G_T and, in each iteration, it checks if the set of triples has a single connected component. As a conclusion, $O(e_s\, t_s + e_t\, t_t + t_s^{e_s}(t_s + t_t^{e_t}\, t_t)) = O(e_s\, t_s + e_t\, t_t + t_s^{e_s} + t_s^{e_s}\, t_t^{e_t})$ is an upper bound to the time a call to this algorithm requires to terminate.

In the worst case, we add a new exchange sample in each iteration of the two loops, therefore, we add $t_s^{e_s}\, t_t^{e_t}$ exchange samples to the output set. □

Proposition 6 (Compute related triples, see Fig. 4)**.** *Let E be a set of entities, and T be a knowledge base. A call to computeRelatedTriples(E, T) terminates in $O(e\,t)$ time, where e denotes the number of entities in E, and t denotes the number of triples in T. In the worst case, this algorithm outputs a set of knowledge bases in which the principal set contains e sets, and the internal knowledge bases contain t triples each.*

Proof. Algorithm *computeRelatedTriples* iterates over the whole set of entities and, for each entity, it iterates over the triples of the input knowledge base. As a conclusion, it terminates in $O(e\,t)$ time.

In the worst case, for each entity in E, it adds a new knowledge base to the final set. Additionally, in the worst case, every triple in T is related to each entity in E, which adds every triple in T to the internal knowledge base. Therefore, the principal set that Algorithm *computeRelatedTriples* outputs contains e sets, and the internal knowledge bases contain t triples in the worst case. □

Proposition 7 (Compute connected components, see Fig. 5)**.** *Let T be a knowledge base. $O(t)$, where t denotes the number of triples in T, is an upper bound to the time a call to computeConnectedComponents(T) requires to terminate.*

Proof. Algorithm *computeConnectedComponents* iterates over the whole set of input triples, and it creates a graph in which, in the worst case, the number of vertices is equal to $2t$, and the number of arcs is equal to t. According to Hopcroft and Tarjan [40], Algorithm *findConnectedComponents* terminates in $O(max\{n, a\})$ time, where n and a are the number of nodes and arcs of the input graph, respectively. In our case, a is equal to $2t$ and n is equal to t; therefore, Algorithm *findConnectedComponents* terminates in $O(2t)$ time. As a conclusion, $O(t + 2t) = O(t)$ is an upper bound to the time a call to *computeConnectedComponents* requires to terminate. □

Proposition 8 (Discard candidate exchange samples, see Fig. 6)**.** *Let D be a set of exchange samples. Algorithm discardCandidateExchangeSamples(D) terminates in $O(d)$ time, where d denotes the number of exchange samples in D.*

Proof. Algorithm *discardCandidateExchangeSamples* iterates two times over the whole input set of exchange samples, and the operations performed can be safely assumed to terminate in $O(1)$ time. As a conclusion, it terminates in $O(d)$ time. □

Proposition 9 (Compute replacements, see Fig. 7)**.** *Let T_1 and T_2 be two knowledge bases. $O(t_1^4\,(2t_2)^{t_1})$ is an upper bound to the time a call to computeReplacements(T_1, T_2) requires to terminate, where t_1 and t_2 denote the number of triples in T_1 and T_2, respectively. In the worst case, this algorithm outputs $(2t_2)^{2t_1}$ replacements.*

Proof. The *computeReplacements* algorithm performs a unique call to Algorithm *computeCandidateEquivalences* for which, according to Proposition 10, $O(t_1\, t_2)$ is an upper bound of its worst-case complexity. It iterates over the distributive Cartesian product of the set output by Algorithm *computeCandidateEquivalences*, in which, according to Proposition 10, the principal set contains $2t_1$ sets, and the internal knowledge bases contain $2t_2$ equivalences; therefore, an upper bound to the number of iterations is $(2t_2)^{2t_1}$. Additionally, in each iteration, it iterates four times over the equivalences in the internal knowledge bases to check if every candidate equivalence relates a given constant in T_1 to the same constant in T_2 (see lines 12–14); note that each set in

$\prod G$ has $2t_1$ elements, therefore, lines 12–14 require $(2t_1)^4$ iterations. The second iteration (see lines 18–20) iterates a maximum of $(2t_1)^2$ times.

Finally, this Algorithm calls Algorithm *applyHomomorphism* once, which, according to Proposition 11, terminates in $O(t_1)$ time. We get the following expression: $O(t_1\, t_2 + (2t_2)^{2t_1}((2t_1)^4 + (2t_1)^2 + t_1))$, which can be simplified as follows: $O(t_1^4\,(2t_2)^{t_1})$.

Furthermore, if every candidate replacement is transformed into a replacement, we add one replacement in every iteration of the main loop, which generates $(2t_2)^{2t_1}$ replacements in the worst case. □

Proposition 10 (Compute candidate equivalences, see Fig. 8)**.** *Let T_1 and T_2 be two knowledge bases. $O(t_1\, t_2)$ is an upper bound to the time a call to computeCandidateEquivalences(T_1, T_2) requires to terminate, where t_1 and t_2 denotes the number of triples in T_1 and T_2, respectively. In the worst case, this algorithm outputs a set of equivalences in which the principal set contains $2t_1$ sets, and the internal sets contain $2t_2$ equivalences.*

Proof. Algorithm *computeCandidateEquivalences* has to iterate over knowledge base T_1. In the worst case, the predicates in the triples of T_1 are the same as the predicates in the triples of T_2, which entails that it is necessary to iterate over the whole knowledge base T_2. As a conclusion, $O(t_1 t_2)$ is an upper bound to the time a call to *computeCandidateEquivalences* requires to terminate.

In the worst case, for each triple in T_1, we add two sets to the final set. Furthermore, in the worst case, every triple in T_1 has the same predicate as every triple in T_2, which adds two equivalences to the internal set. Therefore, the set that Algorithm *computeCandidateEquivalences* outputs contains $2t_1$ sets, and the internal sets contain $2t_2$ equivalences in the worst case. □

Proposition 11 (Applying homomorphisms, see Fig. 9)**.** *Let h be a homomorphism, and T a knowledge base. Algorithm applyHomomorphism(h, T) terminates in $O(t)$ time, where t denotes the number of triples in T.*

Proof. Algorithm *applyHomomorphism* has to iterate through the whole set of triples T. As a conclusion, Algorithm *applyHomomorphism*(h, T) terminates in $O(t)$ time. □

Proposition 12 (Complete exchange samples, see Fig. 10)**.** *Let D be a set of exchange samples. $O(d^4\,(2t_s)^{t_s}\,(t_s^4 + t_t))$ is an upper bound to the time a call to Algorithm completeExchangeSamples(D) requires to terminate, where d denotes the number of exchange samples in D, and t_s and t_t denote the maximum number of source and target triples in the exchange samples of D, respectively.*

Proof. Algorithm *completeExchangeSamples* iterates over the whole set of input exchange samples (see lines 12–30). Then, in the worst case, in which we have to compare every input exchange sample, it iterates again over the exchange samples except for one (see lines 14–29). The while loop, in the worst case, iterates the same number of times, so the first three loops iterate $(d(d-1))^2 < d^4$ times in the worst case. According to Proposition 9, $O(t_1^4\,(2t_2)^{t_1})$ is an upper bound to the time a call to Algorithm *computeReplacements* requires to terminate, where t_1 and t_2 denotes the number of triples in the input knowledge bases. Assume that t_s and t_t denote the maximum number of source and target triples in the exchange samples of D; so this call to *computeReplacements* terminates in $O(t_s^4\,(2t_s)^{t_s})$. According to Proposition 9, Algorithm *computeReplacements* outputs $(2t_2)^{2t_1}$ replacements in the worst case; therefore, this call generates $(2t_s)^{2t_s}$ replacements through which the next loop has to iterate (see lines 18–28). In each iteration, we call Algorithm *applyHomomorphism*, which terminates in $O(t_t)$ time according to Proposition 11. As a conclusion, $O(d^4\,(t_s^4\,(2t_s)^{t_s} + (2t_s)^{2t_s}\,t_t)) = O(d^4\,(2t_s)^{t_s}\,(t_s^4 + t_t))$ is an upper bound to the time a call to this algorithm requires to terminate. □

Proposition 13 (Prune exchange samples, see Fig. 11)**.** *Let D be a set of exchange samples. An upper bound to the time a call to pruneExchangeSamples(D) requires to terminate is $O(d^2\,(t_s^4\,(2t_s)^{t_s} + t_t^4\,(2t_t)^{t_t}))$, where d denotes the number of input exchange samples, and t_s and t_t denote the maximum number of source and target triples in the exchange samples of D, respectively.*

Proof. Algorithm *pruneExchangeSamples* iterates over the whole set of input exchange samples. Then, it iterates again to filter some of them out. In each iteration, in the worst case, we call Algorithm *computeReplacements* four times, which terminates in $O(t_1^4\,(2t_2)^{t_1})$ time in the worst case according to Proposition 9, where t_1 and t_2 denote the number of triples in the input knowledge bases. Assume that t_s and t_t denote the maximum number of source and target triples in the exchange samples of D, therefore, these four calls terminates in $O(2t_s^4\,(2t_s)^{t_s} + 2t_t^4\,(2t_t)^{t_t})$. As a conclusion, $O(d^2\,(t_s^4\,(2t_s)^{t_s} + t_t^4\,(2t_t)^{t_t}))$ is an upper bound to the time a call to this algorithm requires to terminate. □

Proposition 14 (Create schema mappings, see Fig. 12)**.** *Let D be a set of exchange samples. $O(d\,(t_s + t_t))$ is an upper bound to the time a call to createSchemaMappings(D) requires to terminate, where d denotes the number of input exchange samples, and t_s and t_t denote the maximum number of source and target triples in the exchange samples of D, respectively.*

Proof. Algorithm *createSchemaMappings* iterates over the whole set of input exchange samples. Then, it iterates over the whole set of source constants, which is, in the worst case, double the maximum number of source triples in the input exchange samples, i.e., every triple comprises two constants. Additionally, it iterates over the whole set of target constants that are not present in the source, which is, in the worst case, double the maximum number of target triples in the input exchange samples, i.e., every triple comprises two constants that are not present in the source. Finally, the source and target substitutions that this algorithm computes are applied to generate the final schema mappings. According to Proposition 11, applying a substitution terminates in $O(t)$ time. As a conclusion, $O(d\,(3t_s + 3t_t)) = O(d\,(t_s + t_t))$ is an upper bound to the time a call to *createSchemaMappings* requires to terminate. □

References

[1] B. Alexe, L. Chiticariu, W.C. Tan, SPIDER: a schema mapping debugger, in: Proceedings of International Conference on Very Large Data Bases, VLDB, 2006, pp. 1179–1182.

[2] B. Alexe, L. Chiticariu, R.J. Miller, W.C. Tan, Muse: mapping understanding and design by example, in: Proceedings of IEEE International Conference on Data Engineering, ICDE, 2008, pp. 10–19.

[3] B. Alexe, W.C. Tan, Y. Velegrakis, STBenchmark: towards a benchmark for mapping systems, PVLDB 1 (1) (2008) 230–244.

[4] B. Alexe, B.t. Cate, P.G. Kolaitis, W.C. Tan, Designing and refining schema mappings via data examples, in: Proceedings of SIGMOD Conference, 2011, pp. 133–144.

[5] B. Alexe, B.t. Cate, P.G. Kolaitis, W.C. Tan, EIRENE: interactive design and refinement of schema mappings via data examples, PVLDB 4 (12) (2011) 1414–1417.

[6] B. Alexe, B.t. Cate, P.G. Kolaitis, W.C. Tan, Characterizing schema mappings via data examples, ACM Transactions on Database Systems 36 (4) (2011) 23.

[7] F. Alkhateeb, J.-F. Baget, J. Euzenat, Extending SPARQL with regular expression patterns (for querying RDF), Journal of Web Semantics 7 (2) (2009) 57–73.

[8] S. Amano, L. Libkin, F. Murlak, XML schema mappings, in: Proceedings of ACM Symposium on Principles of Database Systems, PODS, 2009, pp. 33–42.

[9] M. Arenas, L. Libkin, XML data exchange: consistency and query answering, The Journal of the ACM 55 (2) (2008) Article 7.

[10] J. Ashraf, E. Chang, O.K. Hussain, F.K. Hussain, Ontology usage analysis in the ontology lifecycle: a state-of-the-art review, Knowledge-Based Syst. 80 (2015) 34–47.

[11] P. Barceló, J. Pérez, J.L. Reutter, Schema mappings and data exchange for graph databases, in: Proceedings of International Conference on Database Theory, ICDT, 2013, pp. 189–200.

[12] Z. Bellahsene, A. Bonifati, E. Rahm (Eds.), Schema Matching and Mapping, Springer, 2011.

[13] T. Berners-Lee, D. Connolly, Notation3 (N3): a readable RDF syntax, W3C, 2011 Technical report.URL: http://www.w3.org/TeamSubmission/n3/ (accessed 09/2015).

[14] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, R. Velkov, OWLIM: a family of scalable semantic repositories, Journal of Web Semantic 2 (1) (2011) 33–42.
[15] C. Bizer, A. Schultz, The R2R framework: publishing and discovering mappings on the Web, in: Proceedings of International Workshop on Consuming Linked Data, COLD, 2010.
[16] C. Bizer, T. Heath, T. Berners-Lee, Linked data: the story so far, International Journal on Semantic Web and Information Systems 5 (3) (2009) 1–22.
[17] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, S. Hellmann, DBpedia: a crystallization point for the Web of Data, Journal of Web Semantics 7 (3) (2009) 154–165.
[18] C. Böhm, M. Freitag, A. Heise, C. Lehmann, A. Mascher, F. Naumann, V. Ercegovac, M.A. Hernández, P. Haase, M. Schmidt, GovWILD: integrating open government data for transparency, in: Proceedings of the 21st International Conference Companion on World Wide Web, WWW, 2012, pp. 321–324.
[19] K.D. Bollacker, C. Evans, P. Paritosh, T. Sturge, J. Taylor, Freebase: a collaboratively created graph database for structuring human knowledge, in: Proceedings of SIGMOD Conference, 2008, pp. 1247–1250.
[20] J. Broekstra, A. Kampman, F. van Harmelen, Sesame: a generic architecture for storing and querying RDF and RDF Schema, in: Proceedings of International Semantic Web Conference, ISWC, 2002, pp. 54–68.
[21] S. Campinas, T. Perry, D. Ceccarelli, R. Delbru, G. Tummarello, Introducing RDF graph summary with application to assisted SPARQL formulation, in: Proceedings of International Conference on Database and Expert Systems, DEXA Workshops, 2012, pp. 261–266.
[22] J.J. Carroll, Matching RDF graphs, in: Proceedings of International Semantic Web Conference, ISWC, 2002, pp. 5–15.
[23] J.J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, K. Wilkinson, Jena: Implementing the Semantic Web recommendations, in: Proceedings of the 21st International Conference Companion on World Wide Web, WWW, 2004, pp. 74–83.
[24] W.G. Cochran, Sampling Techniques, John Wiley & Sons, 1977.
[25] DBpedia 3.8. DBpedia downloads, 2012. URL: http://wiki.dbpedia.org/Downloads38 (accessed 09/2015).
[26] M.G. de Carvalho, A.H.F. Laender, M.A. Gonçalves, A.S.d. Silva, An evolutionary approach to complex schema matching, Information Systems 38 (3) (2013) 302–316.
[27] D. Dou, D.V. McDermott, P. Qi, Ontology translation on the semantic web, Journal on Data Semantics 2 (2005) 35–57.
[28] R. Fagin, P.G. Kolaitis, R.J. Miller, L. Popa, Data exchange: semantics and query answering, Theoretical Computer Science 336 (1) (2005) 89–124.
[29] G. Flouris, D. Manakanatas, H. Kondylakis, D. Plexousakis, G. Antoniou, Ontology change: classification and survey, Knowledge Engineering Review 23 (2) (2008) 117–152.
[30] J. Freire, P. Bonnet, D. Shasha, Computational reproducibility: state-of-the-art, challenges, and database research opportunities, in: Proceedings of SIGMOD Conference, 2012, pp. 593–596.
[31] B. Glimm, A. Hogan, M. Krötzsch, A. Polleres, OWL: yet to arrive on the Web of Data? in: Proceedings of the Workshop on Linked Data on the Web, LDOW, 2012.
[32] M. Golfarelli, F. Mandreoli, W. Penzo, S. Rizzi, E. Turricchia, OLAP query reformulation in peer-to-peer data warehousing, Information Systems 37 (5) (2012) 393–411.
[33] Google, Freebase data dumps. URL http://download.freebase.com/datadumps/ (accessed 02.12.12).
[34] GovWILD. GovWILD dumps. URL http://govwild.hpi-web.de/project/govwild-data.html (accessed 23.03.12).
[35] L.M. Haas, M.A. Hernández, H. Ho, L. Popa, M. Roth, Clio grows up: from research prototype to industrial tool, in: Proceedings of SIGMOD Conference, 2005, pp. 805–810.
[36] T. Heath, How will we interact with the Web of Data? IEEE Internet Computing 12 (5) (2008) 88–91.
[37] T. Heath, C. Bizer, Linked Data: Evolving the Web into a Global Data Space, Morgan & Claypool, 2011.
[38] I. Hernández, C.R. Rivero, D. Ruiz, R. Corchuelo, CALA: an unsupervised URL-based web page classification system, Knowledge-Based Systems 57 (2014) 168–180.
[39] A. Hogan, J. Umbrich, A. Harth, R. Cyganiak, A. Polleres, S. Decker, An empirical survey of linked data conformance, Journal of Web Semantics 14 (2012) 14–44.
[40] J.E. Hopcroft, R.E. Tarjan, Efficient algorithms for graph manipulation [H] (Algorithm 447), Communications of the ACM 16 (6) (1973) 372–378.
[41] G. Klyne, J.J. Carroll, Resource description framework (RDF): concepts and abstract syntax, W3C, 2004 Technical report.URL http://www.w3.org/TR/rdf-concepts/
[42] H. Köpcke, E. Rahm, Frameworks for entity matching: a comparison, Data & Knowledge Engineering 69 (2) (2010) 197–210.
[43] G. Lausen, M. Meier, M. Schmidt, SPARQLing constraints for RDF, EDBT (2008) 499–509.
[44] M. Lenzerini, Data integration: a theoretical perspective, in: Proceedings of ACM Symposium on Principles, 2002, pp. 233–246.
[45] A. Maedche, B. Motik, N. Silva, R. Volz, MAFRA: a mapping framework for distributed ontologies, in: Proceedings of Conference on Knowledge Engineering and Knowledge Management, EKAW, 2002, pp. 235–250.
[46] A. Mallea, M. Arenas, A. Hogan, A. Polleres, On blank nodes, in: Proceedings of International Semantic Web Conference, ISWC, 2011, pp. 421–437.
[47] G. Mecca, P. Papotti, S. Raunich, Core schema mappings, in: Proceedings of SIGMOD Conference, 2009, pp. 655–668.
[48] S.L.S. Mergen, C.A. Heuser, Data translation between taxonomies, in: Proceedings of International Conference on Advanced Information Systems Engineering, CAiSE, 2006, pp. 111–124.
[49] A. Mocan, E. Cimpian, An ontology-based data mediation framework for semantic environments, International Journal on Semantic Web & Information Systems 3 (2) (2007) 69–98.
[50] B. Omelayenko, Integrating vocabularies: discovering and representing vocabulary maps, in: Proceedings of International Semantic Web Conference, ISWC, 2002, pp. 206–220.
[51] F.S. Parreiras, S. Staab, S. Schenk, A. Winter, Model driven specification of ontology translations, in: Proceedings of International Conference on Conceptual Modeling – ER, 2008, pp. 484–497.
[52] M. Petropoulos, A. Deutsch, Y. Papakonstantinou, Y. Katsis, Exporting and interactively querying web service-accessed sources: the CLIDE system, ACM Transactions on Database Systems 32 (4) (2007) Article 22.
[53] X.H. Pham, J.J. Jung, Recommendation system based on multilingual entity matching on linked open data, Journal of Intelligent and Fuzzy Systems 27 (2) (2014) 589–599.
[54] L. Popa, Y. Velegrakis, R.J. Miller, M.A. Hernández, R. Fagin, Translating web data, in: Proceedings of International Conference on Very Large Data Bases, VLDB, 2002, pp. 598–609.
[55] E. Prud'hommeaux, A. Seaborne, SPARQL query language for RDF, W3C, 2008 Technical report.URL http://www.w3.org/TR/rdf-sparql-query/
[56] L. Qian, M.J. Cafarella, H.V. Jagadish, Sample-driven schema mapping, in: Proceedings of SIGMOD Conference, 2012, pp. 73–84.
[57] A. Raffio, D. Braga, S. Ceri, P. Papotti, M.A. Hernández, Clip: a visual language for explicit schema mappings, in: Proceedings of IEEE International Conference on Data Engineering, ICDE, 2008, pp. 30–39.
[58] J. Ressler, M. Dean, E. Benson, E. Dorner, C. Morris, Application of ontology translation, in: Proceedings of International Semantic Web Conference, ISWC, 2007, pp. 830–842.
[59] C.R. Rivero, A. Schultz, C. Bizer, D. Ruiz, Benchmarking the performance of linked data translation systems, in: Proceedings of Linked Data on the Web Workshop, LDOW, 2012.
[60] C.R. Rivero, I. Hernández, D. Ruiz, R. Corchuelo, Research prototype, repositories, scripts, and experimental results. URL: http://tdg-seville.info/carlosrivero/MostoDEx (accessed 09/2015).
[61] C.R. Rivero, I. Hernández, D. Ruiz, R. Corchuelo, MostoDE: a tool to exchange data amongst semantic-web ontologies, Journal of Systems and Software 86 (6) (2013) 1517–1529.
[62] C.R. Rivero, I. Hernández, D. Ruiz, R. Corchuelo, Exchanging data amongst linked data applications, Knowledge and Information Systems 37 (3) (2013) 693–729.
[63] C.R. Rivero, I. Hernández, D. Ruiz, R. Corchuelo, Benchmarking data exchange among semantic-web ontologies, IEEE Transactions on Knowledge and Data Engineering 25 (9) (2013) 1997–2009.
[64] C.R. Rivero, I. Hernández, D. Ruiz, R. Corchuelo, MostoDEx: a tool to exchange RDF data using exchange samples, Journal of Systems and Software 100 (2015) 67–79.
[65] B.t. Cate, V. Dalmau, P.G. Kolaitis, Learning schema mappings, in: Proceedings of International Conference on Database Theory, ICDT, 2012, pp. 182–195.
[66] J. Völker, M. Niepert, Statistical schema induction, in: Proceedings of European Semantic Web Conference, ESWC, 2011, pp. 124–138.
[67] J. Wang, H. Liu, H. Wang, A mapping-based tree similarity algorithm and its application to ontology alignment, Knowledge-Based Systems 56 (2014) 97–107.
[68] J. Xu, R. Pottinger, Integrating domain heterogeneous data sources using decomposition aggregation queries, Information Systems 39 (1) (2014) 80–107.
[69] N. Zong, S. Nam, J. Eom, J. Ahn, H. Joe, H. Kim, Aligning ontologies with subsumption and equivalence relations in linked data, Knowledge-Based Systems 76 (2015) 30–41.