Peer-to-Peer Semantic Integration of Linked Data

Mirko Michele Dimartino
London Knowledge Lab
Birkbeck, University of London
mirko@dcs.bbk.ac.uk

Andrea Calì*
London Knowledge Lab
Birkbeck, University of London
andrea@dcs.bbk.ac.uk

Alexandra Poulovassilis
London Knowledge Lab
Birkbeck, University of London
ap@dcs.bbk.ac.uk

Peter Wood
London Knowledge Lab
Birkbeck, University of London
ptw@dcs.bbk.ac.uk

ABSTRACT

We propose a framework for peer-based integration of linked data sets, where the semantic relationships between data at different peers are expressed through mappings. We provide the theoretical foundations for such a setting and we devise an algorithm for processing graph pattern queries, discussing its complexity and scalability.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*distributed databases, query processing*

General Terms

Algorithms

Keywords

Rewriting, SPARQL, RDF, Peer-to-Peer, Semantic Web

1. INTRODUCTION

In recent years the World Wide Web has gradually expanded from a simple network of hyper-linked documents to a more complex structure where both documents and data are easily published, consumed and reused. As a result of this rapid transformation, new techniques are required in order to integrate these heterogeneous data into a single global data space, the so-called Linked Open Data (LOD) cloud [2], building on Web infrastructure (URIs and HTTP), Semantic Web standards (such as the Resource Description Framework (RDF) and RDF Schema (RDFS)), and vocabularies. These practices have led to the creation of a worldwide database covering a wide range of domains, varying in type from personal and corporate to statistical and scientific data and reviews [3]. Ideally, users should be able to access an open, global data space with an approach similar to how a local database is queried today, in order to obtain more extensive answers as new data sources appear on the Web. However, linked data poses challenges inherent to integrating and querying highly heterogeneous and distributed data, so the above-stated vision has yet to be entirely realised.

In the LOD environment, it is common for several datasets to describe overlapping domains, often using different standards of data modelling and naming. Therefore a global ontological conceptualisation is impracticable and a more flexible approach for semantic integration is needed. This represents a major research challenge for the web of data.

To cope with these limitations, some work in the literature addresses the problem of answering SPARQL queries over disparate sources, proposing new SPARQL rewriting algorithms that entail semantic mappings between RDF databases [18, 10, 19, 20]. These techniques address query rewriting from one source to another, while the LOD cloud is a dynamic environment that comprises several data sources with arbitrary mapping topologies in a peer-to-peer fashion. In fact, in this scenario, an implementation of the existing rewriting algorithms may lack computability, especially in the presence of mapping cycles. The open problem is then developing new data integration techniques to support SPARQL query answering over several heterogeneous RDF sources whose semantic mappings have arbitrary topologies.

The following example considers a typical use-case of querying Linked Data.

Example 1. Figure 1 illustrates an RDF graph containing triples from three different sources. Sources 1 and 2 contain data about films, while Source 3 describes people and their properties. We can see that URIs representing the same entities (e.g., `DB1:Spiderman` and `DB2:Spiderman2002`, for the film *Spiderman*) are linked by the built-in OWL property `sameAs`[1], which states that the linked URIs represent the same real-world entity (best practices for `owl:sameAs` are given in [15]). It is clear that there is a semantic equivalence mapping between URIs linked by `sameAs`. We can also see that there is a semantic equivalence mapping between pairs of triples of the form `(a starring _z)` and `(_z artist b)` in Source 1 and triples of the form `(a actor b)` in Source 2; both represent the relationship that "*actor b acted in the film*

*Also affiliated to the Oxford-Man Institute of Quantitative Finance, University of Oxford, `andrea.cali@oxford-man.ox.ac.uk` .

[1] `http://sameas.org`

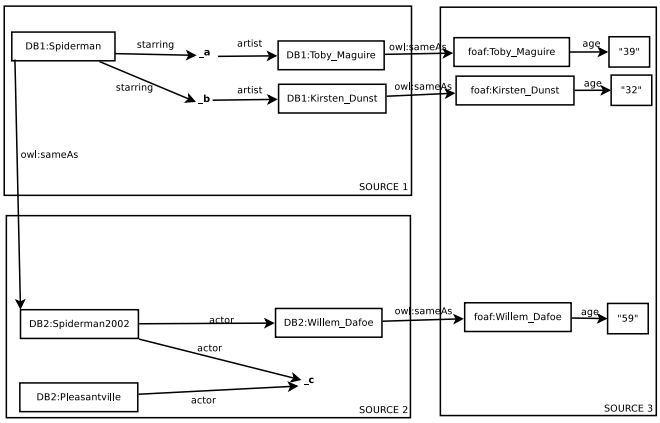

Figure 1: Example of an RDF graph from three data sources.

a". Now assume that a user poses the following SPARQL[2] query:

```
SELECT ?x ?y
WHERE { DB1:Spiderman starring ?z . ?z artist ?x .
        ?x age ?y }
```

This query returns an empty result on the data of Figure 1, since the `sameAs` property is missing from the query, and SPARQL does not automatically exploit semantic mappings between RDF resources. As stated above, adopting existing rewriting techniques to entail the semantic mappings is impractical for this scenario, since there are more than two RDF sources and the mapping topologies are arbitrary. In this regard, we propose a decentralised, easily extensible, RDF-oriented peer data management system. We provide the theoretical foundations for such a setting and we devise an algorithm for processing graph pattern queries, discussing its complexity and scalability.

Related work: As previously stated in this section, few papers in the literature deal with the challenges of answering SPARQL queries over data, leveraging the semantic mappings between similar vocabularies. For instance, the work in [18] presents a query rewriting algorithm over virtual SPARQL views; and, similarly, [10] introduces a SPARQL rewriting algorithm based on the encoding of rules for RDF patterns, involving entity equivalence functions for the semantics of the property `owl:sameAs`. In [19, 20] Makris et al. instead adopt Description Logic rules between overlapping OWL ontologies. These works address query answering over two-tiered architectures, while we wish to explore the most general case, where the number of sources and the mapping topologies are arbitrary.

Several peer-to-peer systems for RDF datasources can be found in the literature. For instance, in [5, 6] the authors describe a distributed RDF metadata storage, querying and subscription service as a structured P2P network. Similarly, work in [23] proposes routing strategies for RDF-based P2P networks. Similar work can be found in [21, 22, 17]. However, all of these papers focus on technical issues relating to peer networks (such as efficiency of query routing, network traffic load, etc.), and so leave a gap between the RDF data model and peer-to-peer semantic integration.

Paper outline: The paper is organised as follows. In Section 2 we present our new framework for RDF peer-to-peer integration. Then, in Section 3 we explore the query answering problem under RDF Peer Systems and we propose a query answering algorithm that terminates in polynomial time. In Section 4 we explore query rewriting techniques in our setting, showing that our mapping rules are not first-order-rewritable. We conclude with a discussion in Section 5.

2. THE FRAMEWORK

In this section, we introduce our framework for peer-to-peer RDF semantic data integration. We present a new peer mapping language suitable for the RDF data model that extends the goals of relational P2P models to achieve semantic integration in accordance with Linked Data technologies. Our goal is to leverage the techniques for specifying semantic mappings between RDF sources, extending them beyond a two-tiered architecture. In our framework, each peer is represented by its peer schema, comprising the set of URIs adopted in the peer to model data. Integration is achieved by means of mappings between these sets of URIs. To formally specify the problem of query answering, we generalize the notion of *certain answers* [1] to our context.

2.1 Graph pattern queries

To formalise the problem, we introduce the notion of graph pattern queries for RDF databases (for details of RDF formalisation, see [14]). Assume there are pairwise disjoint infinite sets I, B, and L (IRIs [11], Blank nodes, and Literals, respectively). A triple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an RDF triple. In this triple, s is the *subject*, p the *predicate*, and o the *object*. Also, we assume the existence of an infinite set of variables V disjoint from $(I \cup B \cup L)$. An *RDF database* is then a set of RDF triples.

A *graph pattern* is defined recursively as follows:

1. A tuple from $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$ is a *graph pattern*. Specifically, it is a *triple pattern*.
2. If GP_1 and GP_2 are graph patterns, then the expression $(GP_1 \text{ AND } GP_2)$ is a *graph pattern*.

We denote by $var(GP)$ the set of variables $V_{GP} \subseteq V$ that appear in the graph pattern GP.

A *graph pattern query* Q of arity n is of the form

$$q(\mathbf{x}) \leftarrow GP$$

where GP is a graph pattern, and $\mathbf{x} = x_1, \ldots, x_n \in var(GP)$ denote the *free variables* of the query. All the elements in $var(GP)$ that are not free variables denote the *existentially quantified variables* of the query.

In order to define the semantics of graph pattern queries, we introduce some terminology from [24, 4] for the evaluation of a graph pattern over an RDF database.

A *mapping* μ from V to $(I \cup B \cup L)$ is a partial function $\mu : V \to (I \cup B \cup L)$. The domain of μ, denoted by $dom(\mu)$, is the subset of V where μ is defined. Given a mapping μ and a variable $v \in dom(\mu)$ we denote by $\mu(v)$ the value in $(I \cup B \cup L)$ obtained by applying the function μ to v. Also,

[2] `http://www.w3.org/TR/rdf-sparql-query/`

abusing notation, for a triple pattern t we denote by $\mu(t)$ the triple obtained by replacing the variables in t according to μ. Two mappings μ_1 and μ_2 are compatible when for all $x \in dom(\mu_1) \cap dom(\mu_2)$, it is the case that $\mu_1(x) = \mu_2(x)$, i.e. when $\mu_1 \cup \mu_2$ is also a mapping.

Let Ω_1 and Ω_2 be sets of mappings. Then the *join* of Ω_1 and Ω_2 is defined as follows [24, 4]:

$$\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ and } \mu_1, \mu_2 \text{ are compatible mappings}\}$$

The semantics of graph patterns is then defined by a function $[\![\cdot]\!]_D$ over a set of RDF triples D, also called an RDF graph or RDF database, which takes a graph pattern as input and returns a set of mappings that matches the database D [4, 24].

Definition 1. (Evaluation of a graph pattern). The evaluation of a graph pattern GP over an RDF dataset D, denoted by $[\![GP]\!]_D$, is defined recursively as follows:

1. If GP is a triple pattern t, then $[\![GP]\!]_D = \{\mu \mid dom(\mu) = var(t) \text{ and } \mu(t) \in D\}$
2. If GP is of the form (GP_1 AND GP_2), then $[\![P]\!]_D = [\![GP_1]\!]_D \bowtie [\![GP_2]\!]_D$.

We are ready to define the semantics of graph pattern queries. We denote by Q^D the set of n-tuples returned by the evaluation of the graph pattern query Q of arity n over the dataset D. We define the semantics of Q^D as follows:

$$Q^D := \{(\mu(x_1), \ldots, \mu(x_n)) \mid \mu \in [\![GP]\!]_D \text{ and } \mu(x_1), \ldots, \mu(x_n) \in (I \cup L)\},$$

where GP is the graph pattern of the query Q and $x_1 \ldots x_n$ are the free variables of Q.

As we can see from the above definition, tuples containing elements in B (blank nodes) are not returned from the evaluation of the query. Blank nodes are used in the RDF triples as placeholders for unknown resources [16]; in other words, they denote variables which may take values in the set of IRIs and literals ($I \cup L$). In this regard, a graph pattern query retrieves only full information, dropping all the tuples containing partial information.

In fact, we define the semantics of blank nodes so as to be equivalent to the semantics of *labelled nulls* in the relational model, which are placeholders for unknown values and are not included in query results. For completeness, we also define a semantics of graph pattern queries which does include blank nodes in the result set. This semantics will be used later on to exploit the expressiveness of equivalence mappings in our RDF Peer System. We denote this semantics by Q^{*D}, where

$$Q^{*D} := \{(\mu(x_1), \ldots, \mu(x_n)) \mid \mu \in [\![GP]\!]_D\}.$$

Note that the graph pattern query language can be seen as a "conjunctive fragment" of SPARQL, so a graph pattern query can always be translated to a conjunctive SPARQL query and vice versa.

2.2 RDF Peer Systems

An *RDF Peer System* (RPS) $\mathcal{P}$ constitutes a set of peers and a set of mappings that specify the semantic relationships between peers. Formally, an RPS $\mathcal{P}$ is defined as a tuple $\mathcal{P} = (\mathcal{S}, G, E)$, where:

- $\mathcal{S}$ is the set of the *peer schemas* in $\mathcal{P}$. Each peer schema $S \in \mathcal{S}$ is simply the set of all the constants $u \in I$ adopted by the corresponding peer to describe data in the form of RDF triples. Informally, the schema of a peer is a subset of I (the set of all the IRIs in Linked Data) comprising only the IRIs adopted by the peer. Two peer schemas then need not be disjoint sets: this is in accordance with real Linked Data sources, where two different RDF databases may share some IRIs in the RDF triples.
- G is a set of *graph mapping assertions*, each of which is an expression of the form $Q \rightsquigarrow Q'$, where Q and Q' are *graph pattern queries* of the same arity, expressed over the schemas S and S', respectively, of two peers in $\mathcal{P}$. Formally, the graph pattern GP in the query Q contains triple patterns from $(S \cup L \cup V) \times (S \cup V) \times (S \cup L \cup V)$, and the graph pattern GP' in the query Q' contains triple patterns from $(S' \cup L \cup V) \times (S' \cup V) \times (S' \cup L \cup V)$.
- E is a set of *equivalence mappings* of the form $c \equiv_e c'$, where $c \in S$ and $c' \in S'$ and $S, S' \in \mathcal{S}$.

2.3 Semantics of RDF Peer Systems

We assume that we are given an instance of the data stored in the peers in the form of a set of RDF triples for each peer in the system. Formally, for each peer defined by its schema $S \in \mathcal{S}$ in $\mathcal{P}$, we have a database d, that is, a set of triples $(s, p, o) \in (S \cup B) \times S \times (S \cup B \cup L)$. Consequently, a *stored database* D of an RPS $\mathcal{P}$ is the union of all the peer databases d of all the peers in $\mathcal{P}$. Then, a *peer-to-peer database* of an RPS $\mathcal{P}$ is simply an arbitrary RDF database containing triples $(s, p, o) \in (S_1 \cup \cdots \cup S_n \cup B) \times (S_1 \cup \cdots \cup S_n) \times (S_1 \cup \cdots \cup S_n \cup B \cup L)$, where $S_1, \ldots, S_n \in \mathcal{S}$ are the peer schemas in $\mathcal{P}$.

We also denote by $subjQ(c)$, $predQ(c)$ and $objQ(c)$ three special *graph pattern queries*:

- $subjQ(c) := q(x_{pred}, x_{obj}) \leftarrow (c, x_{pred}, x_{obj})$
- $predQ(c) := q(x_{subj}, x_{obj}) \leftarrow (x_{subj}, c, x_{obj})$
- $objQ(c) := q(x_{subj}, x_{pred}) \leftarrow (x_{subj}, x_{pred}, c)$

where $c \in (S_1 \cup \cdots \cup S_n \cup L)$.

The evaluation of $subjQ(c)$ over an RDF dataset is the set of pairs of the form $(t.pred, t.obj)$ containing the *predicate* and *object* of all triples in the dataset where the constant c occurs as the *subject*. The queries $predQ(c)$ and $objQ(c)$ are defined similarly, with the constant c now occurring as the *predicate* and the *object* of an RDF triple, respectively.

Below we give formal definitions for a *solution* for an RPS $\mathcal{P}$ and the set of *certain answers* for a query posed against $\mathcal{P}$. Informally, a peer-to-peer database is a solution of an RPS $\mathcal{P}$ if it contains the stored database of $\mathcal{P}$, as well as all triples inferred by the mappings of $\mathcal{P}$. The certain answers to a query against $\mathcal{P}$ are those which appear in all possible solutions of $\mathcal{P}$.

Definition 2. A peer-to-peer database I is said to be a *solution* for an RPS $\mathcal{P}$ based on a stored database D if:

1. For every stored database $d \in D$, we have that $d \subseteq I$.
2. For every graph mapping assertion in G of the form $Q \rightsquigarrow Q'$, we have that $Q^I \subseteq Q'^I$.
3. For every equivalence mapping in E of the form $c \equiv_e c'$, all of the following hold:

$$subjQ(c)^{*I} = subjQ(c')^{*I}$$
$$predQ(c)^{*I} = predQ(c')^{*I}$$
$$objQ(c)^{*I} = objQ(c')^{*I}$$

Definition 3. We define the *certain answers* $ans(q, \mathcal{P}, D)$ to an arbitrary graph pattern query q of arity n, based on a stored database D of an RPS $\mathcal{P}$, as the set of n-tuples $\mathbf{t}$ of constants in $(S_1 \cup \cdots \cup S_n \cup L)$ such that, for every peer-to-peer database I that is a solution for the system $\mathcal{P}$ based on D, we have that $\mathbf{t} \in q^I$.

The query answering problem is defined as follows: given an RPS $\mathcal{P}$, a stored database D and a graph pattern query q, find the certain answers $ans(q, \mathcal{P}, D)$.

3. QUERY ANSWERING

To evaluate the complexity of the query answering problem we show that the problem of finding $ans(q, \mathcal{P}, D)$ is subsumed by CQ answering in data exchange for the relational model. Specifically, we show that a solution of an RPS can be seen as a solution of a special data exchange setting.

A data exchange setting is defined by a *source* relational alphabet $\boldsymbol{S}$, a *target* relational alphabet $\boldsymbol{T}$, a set Σ_{st} of *source-to-target* dependencies and a set Σ_t of *target* dependencies. Instances over $\boldsymbol{S}$ are called *source* instances, while instances over $\boldsymbol{T}$ are called *target* instances. Given a source instance I, the problem is to find a solution J over the target schema such that $I \cup J$ satisfies the source-to-target dependencies and J satisfies the target dependencies [12].

For a given RPS $\mathcal{P} = (\mathcal{S}, G, E)$ and a stored database D for $\mathcal{P}$, we can define a data exchange setting such that a solution for the data exchange problem is a solution for $\mathcal{P}$. We define the relational alphabets $\boldsymbol{R_s} := \{t_s, r_s\}$ and $\boldsymbol{R_t} := \{t_t, r_t\}$, where t_s and t_t are ternary relational symbols and r_s and r_t are unary relational symbols. These relational alphabets describe the RDF triples (t_s) and the identified resources (r_s) stored by the peers in D, and the RDF triples (t_t) and the identified resources (r_t) inferred in a peer-to-peer database of $\mathcal{P}$.

Given a relational alphabet A, we denote by $\mathcal{L}_A$ the set of function-free first-order logic (FOL) formulas whose relation symbols are in A and whose constants are in $(I \cup B \cup L)$. Then, given a graph pattern query Q of the form $q(\mathbf{x}) \leftarrow GP$ we can define the term $Q_{body}(\mathbf{x}, \mathbf{y})$ as the conjunction of the atoms in $\mathcal{L}_{\boldsymbol{R_t}}$ representing triple patterns in GP, where $\mathbf{x} = x_1, \ldots, x_n \in var(GP)$ are the free variables, and $\mathbf{y} = y_1, \ldots, y_m \in var(GP)$ are the existentially quantified variables in Q. For example, given the following graph pattern query

$$Gfather \;:=\; q(x_1, x_2) \leftarrow (x_1, father, y) \text{ AND } (y, father, x_2),$$

$Gfather_{body}(\mathbf{x}, \mathbf{y})$ is the conjunction of atoms

$$t_t(x_1, father, y) \wedge t_t(y, father, x_2),$$

where $\mathbf{x} = x_1, x_2$ and $\mathbf{y} = y$. In this regard, evaluating a graph pattern query Q over an instance of a RPS is equivalent to evaluating the following conjunctive query (CQ) over an interpretation of the relations in $\boldsymbol{R_t}$:

$$\{\mathbf{x} \mid \exists \mathbf{y}\ Q_{body}(\mathbf{x}, \mathbf{y}) \wedge r_t(x_1) \wedge \cdots \wedge r_t(x_n)\}.$$

Given an RPS $\mathcal{P} = (\mathcal{S}, G, E)$, we are now ready to define a data exchange setting whose solution (seen as an RDF database) is also a solution for $\mathcal{P}$. The relational alphabets $\boldsymbol{R_s}$ and $\boldsymbol{R_t}$ are the source and target relational alphabets of the data exchange setting. The source-to-target dependencies express the semantics of item 1 in Definition 2, which states that the peer-to-peer database of the RPS must contain all the triples in the stored database. They are of the form:

$$\forall x \forall y \forall z\ t_s(x, y, z) \rightarrow t_t(x, y, z),$$
$$\forall x\ r_s(x) \rightarrow r_t(x).$$

The target dependencies express the semantics of the graph mapping assertions and the equivalence mappings. For each graph mapping assertion $Q \rightsquigarrow Q' \in G$, we have the dependency

$$\forall \mathbf{x} \exists \mathbf{y}\ Q_{body}(\mathbf{x}, \mathbf{y}) \wedge r_t(x_1) \wedge \cdots \wedge r_t(x_n) \rightarrow \exists \mathbf{z}\ Q'_{body}(\mathbf{x}, \mathbf{z}),$$

and for each equivalence mapping $c \equiv_e c' \in E$, we have the dependencies

$$\forall y \forall z\ t_t(c, y, z) \rightarrow t_t(c', y, z),$$
$$\forall y \forall z\ t_t(c', y, z) \rightarrow t_t(c, y, z),$$
$$\forall x \forall z\ t_t(x, c, z) \rightarrow t_t(x, c', z),$$
$$\forall x \forall z\ t_t(x, c', z) \rightarrow t_t(x, c, z),$$
$$\forall x \forall y\ t_t(x, y, c) \rightarrow t_t(x, y, c'),$$
$$\forall x \forall y\ t_t(x, y, c') \rightarrow t_t(x, y, c).$$

In this regard, query answering under an RPS is equivalent to the problem of CQ answering in data exchange, under the special data exchange setting defined above.

The set of certain answers in the data exchange problem is computed by evaluating queries over the so-called *universal solution* of the data exchange setting. To generate a universal solution, a source database is "chased" using the set of dependencies. Each step of the *chase* "extends" the database so that the chosen dependency is satisfied. For instance, given a dependency $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})$ and a mapping h (from the variables in $\phi(\mathbf{x})$ to constants) for which the dependency is not satisfied, the chase step generates new facts in the target instance in order to satisfy the dependency. The new facts are generated by: (a) extending h to h' such that each existentially quantified variable in $\psi(\mathbf{x}, \mathbf{y})$ is assigned a freshly created constant, a *labelled null*, followed by: (b) taking the image of the atoms of ψ under h' (see [12], Section 3 for more details of the chase procedure).

In our specific data exchange setting, there are no atoms of type $r_t(x)$ in the head of any dependency such that the

variable x is existentially quantified. Therefore, the set of IRIs and literals remains constant during the chase procedure. Thus, the chase generates new blank nodes as labelled nulls. Without loss of generality, we will use the term *newly created blank nodes* when we want to denote labelled nulls.

In an RPS $\mathcal{P} = (\mathcal{S}, G, E)$, only dependencies in G contain existentially quantified variables in the body, therefore they are the only dependencies for which the chase may generate new constants, i.e., newly created blank nodes. Since newly created blank nodes cannot trigger any of these rules, the chase sequence is then bounded by a finite number of steps. This leads to the following theorem:

THEOREM 1. *The problem of finding all certain answers $ans(q, \mathcal{P}, D)$ to an arbitrary graph pattern query q, for a given RDF Peer System $\mathcal{P}$ and a stored database D, has PTIME data complexity.*

Due to lack of space, we omit a formal proof of the theorem, which will appear in an extended version of this paper.

In data exchange, the set of certain answers of a query is then computed by evaluating the query over the universal solution and eliminating all the tuples in the result that contain labelled nulls. In our RDF model, the semantics of a graph pattern query Q^D eliminates all the answer tuples containing blank nodes, so we can generate the certain answers by simply evaluating the graph pattern query over the universal solution. The algorithm that computes the certain answers, Algorithm 1, is listed in the Appendix.

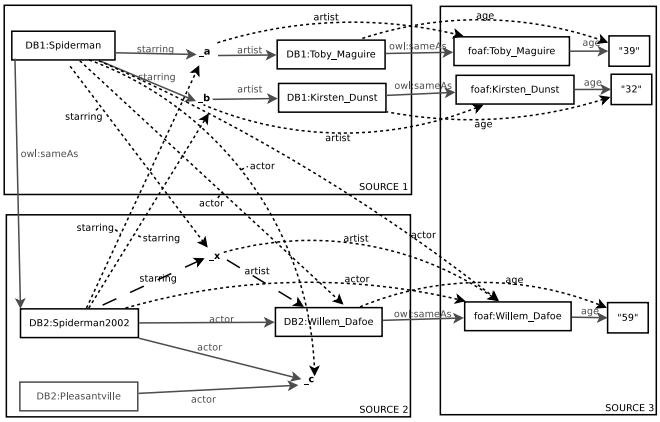

Figure 2: RDF graph of a universal solution for the peer system. Dotted arrows and dashed arrows represent triples inferred by the equivalence mappings and the graph mapping assertions, respectively.

Example 2. Let us consider again the RDF sources of Example 1. We define an RPS $\mathcal{P} = (\mathcal{S}, G, E)$ as follows:

- $\mathcal{S} := \{S_1, S_2, S_3\}$ where S_i is the set of IRIs in the i^{th} source. For example, $S_2 := \{$DB2:Spiderman2002, DB2:Willem_Dafoe, DB2:Pleasantville, actor$\}$ (in this case, we consider `owl:sameAs` triples stored in Source 1 and Source 3).
- G is a single graph mapping assertion of the form $Q_2 \rightsquigarrow Q_1$, where:
  - $Q_1 := q(x, y) \leftarrow (x, starring, z)$ AND $(z, artist, y)$,
  - $Q_2 := q(x, y) \leftarrow (x, actor, y)$.
- E contains an equivalence mapping $c \equiv_e c'$ for each triple of the form $(c, sameAs, c')$.

Figure 2 illustrates an RDF database which is a universal solution for $\mathcal{P}$. Let us consider again the SPARQL query used in Example 1. Now, evaluating the query over the universal solution, we obtain the result in Listing 1. It is important to observe that the user poses a query over Sources 1 and 3 but retrieves additional information also from Source 2 in a transparent way. The RPS, in fact, not only captures the semantics of the `owl:sameAs` property, but also performs integration of similar sources in order to return additional answers to the user. This integration can be performed dynamically as new data sources appear, and requires no input from the user.

```
#Query

SELECT ?x ?y
WHERE { DB1:Spiderman starring ?z .
        ?z artist ?x .
        ?x age ?y }

#Result
DB1:Toby_Maguire "39"
foaf:Toby_Maguire "39"
DB1:Kirsten_Dunst "32"
foaf:Kirsten_Dunst "32"
DB2:Willem_Dafoe "59"
foaf:Willem_Dafoe "59"

#Result without redundancy

DB1:Toby_Maguire "39"
DB1:Kirsten_Dunst "32"
DB2:Willem_Dafoe "59"
```

Listing 1: SPARQL query over the universal solution.

4. QUERY REWRITING

The chase algorithm is a useful tool for query answering, however materialising the universal solution for an RDF peer system may be impractical in the Linked Data scenario due to the large volumes of data involved.

A more efficient approach would involve a rewriting of the original query that, when evaluated directly over the sources, returns the set of certain answers. In other words, given a stored database D, a query q and the set Σ of TGDs that entail the peer mappings, we want to compute a rewriting of q based on Σ, named q_Σ, such that ${q_\Sigma}^D = q^J$, where $J = chase(D, \Sigma)$ is the universal solution for the peer system. In this case q_Σ is a *perfect rewriting* of q since it preserves a sound and complete answer of the original query based on the extensional database D and the "*ontological theory*" Σ.

Several works have addressed query rewriting under TGDs. [8, 9] introduced sets of TDGs, namely *sticky* sets, that enjoy the property of being *FO-rewritable*, i.e., for every query that needs to be evaluated under such dependencies it is

possible to compute a first-order query as a perfect rewriting. The algorithm has as input a *Boolean CQ* q, a database D and a sticky set of TGDs Σ, and it outputs "Accept" if $chase(D, \Sigma) \models q$. We recall that the two problems of CQ and BCQ evaluation under TGDs are LOGSPACE-equivalent.

Stickiness is a sufficient syntactic condition that ensures the so-called sticky property of the chase, which is as follows. For every instance D, assume that during the chase of D under a set Σ of TGDs, we apply a TGD σ that has a variable V appearing more than once in its body; assume also that V maps (via homomorphism) onto the constant z, and that by virtue of this application the atom a is generated by the chase step. In this case, for each atom b in the body of σ, we say that a is derived from b. Then, we have that z appears in a, and in all atoms resulting from some chase derivation sequence starting from a, "sticking" to them (hence the name "sticky sets of TGDs").

The formal definition of sticky sets of TGDs, given in [9], is an efficient testable condition involving variable marking.

Definition 4. Consider a set Σ of TGDs over a relational alphabet $\mathcal{R}$. A *position* $r[i]$ in $\mathcal{R}$ is identified by the predicate $r \in \mathcal{R}$ and its i-th argument (or attribute). We mark the variables that occur in the body of the TGDs of Σ according to the following procedure. First, for each TGD $\sigma \in \Sigma$ and for each variable V in $body(\sigma)$, if there exists an atom a in $head(\sigma)$ such that V does not appear in a, then we mark each occurrence of V in $body(\sigma)$. Given a predicate symbol r, $r[i]$ identifies its i-th argument (or attribute). Now, we apply exhaustively (i.e., until a fixpoint is reached) the following step: for each TGD $\sigma \in \Sigma$, if a marked variable in $body(\sigma)$ appears at a position π, then for every TGD $\sigma' \in \Sigma$ (including the case $\sigma' = \sigma$), we mark each occurrence of the variables in $body(\sigma')$ that appear in $head(\sigma')$ at the same position π. We say that Σ is *sticky* if and only if there is no TGD $\sigma \in \Sigma$ such that a marked variable occurs in $body(\sigma)$ more than once.

Given an RPS $\mathcal{P} = (\mathcal{S}, G, E)$, the set E of TGDs for equivalence mappings enjoys the sticky property of the chase, as well as linearity. Graph mapping assertions in G do not preserve the same property. We can easily show this by applying the variable marking on the following example of a graph mapping assertion:

$$\forall x \forall y \exists z \; t_t(x, A, \hat{z}) \wedge t_t(\hat{z}, B, y) \wedge r_t(x) \wedge r_t(y) \rightarrow t_t(x, C, y),$$

where A,B and C are URIs. Here, applying the variable marking results in the variable z appearing more than once in the body of the TGD. This violates the stickiness condition.

It is important to observe that the set Σ of TGDs in an RPS is neither sticky, nor linear, nor weakly-acyclic [12], nor guarded [7], nor weakly-guarded [7]. In fact our sets of TGDs are incomparable to the above known classes of TGDs under which query answering is decidable. ■

In [13] the authors propose the query rewriting algorithm *TGD-rewrite* which takes as input a BCQ and a set of TGDs each with just one head-atom, containing at most one existentially quantified variable, which occurs only once. The algorithm generates a union of BCQs (i.e., a FO-query) which is a perfect rewriting of the original query. It is shown that query answering under this setting is LOGSPACE-equivalent to query answering under (general) TGDs, and thus the result holds for arbitrary TGDs. The algorithm *TGD-rewrite* guarantees termination under linear, sticky or sticky-join sets of TGDs (a generalisation of sticky TGDs and linear TGDs).

PROPOSITION. 2. *Given an RPS $\mathcal{P} = (\mathcal{S}, G, E)$, a stored database D and a Boolean query q, if G is either linear, sticky, or sticky-join, then we can generate a FO-query $q_{\mathcal{P}}$, such that ${q_{\mathcal{P}}}^D = q^J$, where J is the universal solution for $\mathcal{P}$ based on D.*

Example 3. Consider again the RPS in Example 2. The set G of graph mapping assertions is linear, hence, following from Proposition 2, we can generate an FO-rewriting of a given Boolean query to entail the mapping assertions of the RPS. Listing 2 shows an example rewriting based on the SPARQL query and the RDF stored database shown in the introduction. To compute the set of certain answers of the given query, first we generate the set of all the possible 2-tuples from the stored database. Then we iterate over each 2-tuple t and decide whether or not t is in the set of certain answers, by substituting t into the SPARQL query to obtain a Boolean query (note that this is a polynomial-time reduction of the problem, since there are polynomially many k-tuples from the source database). Each Boolean query can be rewritten as an FO-query according to the mapping assertions in the RPS. In our case, the rewriting generates a union of SPARQL queries. Due to lack of space, we show only one possible step of the query rewriting, which makes use of the dependency

$$\forall y \forall z \; t_t(\textit{foaf:Toby\_Maguire}, y, z) \rightarrow t_t(\textit{DB1:Toby\_Maguire}, y, z)$$

to rewrite the triple pattern `(DB1:Toby_Maguire age "39")`.

```
#Original query

SELECT ?x ?y
WHERE { DB1:Spiderman starring ?z .
?z artist ?x .
?x age ?y }

#Boolean query:
#ask if the tuple (DB1:Toby_Maguire,"39")
#is in the query result.

ASK { DB1:Spiderman starring ?z .
?z artist DB1:Toby_Maguire .
DB1:Toby_Maguire age "39" }

false

#Rewritten query

ASK {{ DB1:Spiderman starring ?z .
?z artist DB1:Toby_Maguire .
DB1:Toby_Maguire age "39" }
UNION
{ DB1:Spiderman starring ?z .
?z artist DB1:Toby_Maguire .
foaf:Toby_Maguire age "39" }}

true
```

Listing 2: SPARQL Boolean query rewriting.

Let us now evaluate the general case.

Consider the TGD $\sigma = A(x,z) \wedge A(z,y) \rightarrow A(x,y)$ and observe that σ is not FO-rewritable since it captures the transitive closure of the relation A, which cannot be done using a finite number of first-order queries. Let us now consider an instance of a RPS $\mathcal{P}$ defined only by the following mapping assertion:

$$\forall x \forall y \exists z \; t_t(x, A, z) \wedge t_t(z, A, y) \wedge r_t(x) \wedge r_t(y) \rightarrow t_t(x, A, y),$$

We assume without loss of generality that the sources in D do not contain blank nodes. Now we transform the mapping assertion into the following set Σ of TGDs:

$$\begin{gathered} \forall x \forall y \; t_t(x, A, y) \rightarrow A(x, y) \\ \forall x \forall y \; A(x, z) \wedge A(z, y) \rightarrow A(x, y) \\ \forall x \forall y \; A(x, y) \rightarrow t_t(x, A, y). \end{gathered}$$

Note that we can drop the atoms $r_t(x)$, $r_t(y)$ in the body of the TGDs because for any D we have that $D \models \forall x \; r_t(x)$, and the same condition holds for every partial instance of the chase in each chase step.

The auxiliary predicates, being introduced only during the above construction, do not match any predicate symbol in any query q, and hence $chase(D, \Sigma) \models q$ if and only if $chase(D, \mathcal{P}) \models q$; therefore query answering under the RPS $\mathcal{P}$ is equivalent to query answering under Σ. Note that Σ now contains TGDs computing the transitive closure, so if we assume that the set of TGDs Σ are FO-rewritable, then the transitive closure is also FO-rewritable which is a contradiction.

This leads to the following result.

PROPOSITION. 3. *The sets of TGDs corresponding to the mapping assertions of RPSs are not FO-rewritable.*

5. DISCUSSION

In this paper we have addressed the problem of integrating RDF data sources in a peer-based fashion, where mappings are defined between arbitrary peers, without a centralised schema. We have proposed a formalisation of the notion of a peer-to-peer semantic integration system, where RDF triples are represented as relational tuples and mappings are expressed as tuple-generating dependencies. Following that, we have shown that answering graph pattern queries on an RDF peer system can be done in polynomial time in terms of data complexity. Finally, we have shown that it is not possible to process queries in general RDF peer systems by rewriting them into first-order queries.

This is a preliminary report which poses several new challenges. As future work, we plan the following.

1. We want to improve the efficiency of the query processing algorithm. Our query answering algorithm is naïve as it generates the whole universal solution under the given dependencies; this is far from ideal, as mappings may be subject to change and we might need to compute the information inferred from the TGDs dynamically. We intend to investigate the possibility of adopting a *combined* approach, where only part of the universal solution is computed, and queries are rewritten according to some of the dependencies only. Another possible approach is to devise a rewriting algorithm that produces rewritten queries in a language more expressive than FO-queries, for instance Datalog.

2. We intend to investigate the query answering problem for more expressive query languages, in particular for larger subsets of SPARQL.

3. We want to be able to discover mappings between peers automatically. We are investigating relevant areas such as probabilistic logics and state-of-the-art techniques for automatic schema/ontology-alignment and for managing uncertain semantic mappings.

4. Finally, we are building a prototype system to validate our techniques on real and synthetic data sets and determine their scalability properties. Our prototype is a SPARQL query engine which provides unified access to the mapped sources. The user poses a query expressed in any vocabulary known by the system, and the query is processed as follows:

   (a) A query rewriting module rewrites the original SPARQL query in order to retrieve all the certain answers.

   (b) A query module performs federated querying over the sources. It stores SPARQL access points of the RDF sources, up-to-date RDF data dumps and other information in order to query federated sources in a transparent way for the user. After query rewriting, sub-queries are posed to the relevant RDF sources and sub-query results are joined, taking into account efficiency of the join operations between the RDF triple patterns. The final result is returned to the user.

Acknowledgments.

Andrea Calì acknowledges support by the EPSRC project "Logic-based Integration and Querying of Unindexed Data" (EP/E010865/1).

6. REFERENCES

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS*, pages 254–263, 1998.

[2] S. Auer, J. Lehmann, and A.-C. N. Ngomo. Introduction to Linked Data and its lifecycle on the web. In *Proc. of RW*, pages 1–75, 2011.

[3] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[4] C. Buil-Aranda, M. Arenas, and O. Corcho. Semantics and optimization of the SPARQL 1.1 federation extension. In *Proc. of ESWC*, pages 1–15, 2011.

[5] M. Cai and M. Frank. RDFPeers: A scalable distributed RDF repository based on a structured peer-to-peer network. In *Proc. of WWW*, pages 650–657, 2004.

[6] M. Cai, M. R. Frank, B. Yan, and R. M. MacGregor. A subscribable peer-to-peer RDF repository for distributed metadata management. *J. Web Sem.*, 2(2):109–130, 2004.

[7] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *Proc. of KR*, pages 70–80, 2008.

[8] A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010.

[9] A. Calì, G. Gottlob, and A. Pieris. Query answering under non-guarded rules in Datalog+/-. In *Proc. of RR*, pages 1–17, 2010.

[10] G. Correndo, M. Salvadores, I. Millard, H. Glaser, and N. Shadbolt. SPARQL query rewriting for implementing data integration over linked data. In *Proc. of EDBT/ICDT*, 2010.

[11] M. Duerst and M. Suignard. RFC 3987: Internationalized Resource Identifiers (IRIs). RFC 3987 (Proposed Standard), January 2005.

[12] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *TCS*, 336(1):89–124, 2005.

[13] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization (extended version). *CoRR*, abs/1112.0343, 2011.

[14] C. Gutierrez, C. Hurtado, and A. O. Mendelzon. Foundations of semantic web databases. In *Proc. of PODS*, pages 95–106, 2004.

[15] H. Halpin, P. Hayes, J. McCusker, D. McGuinness, and H. Thompson. When owl:sameas isn't the same: An analysis of identity in linked data. In *Proc. of ISWC*, volume 6496, pages 305–320, 2010.

[16] P. Hayes and B. McBride. RDF semantics. W3C recommendation, Feb. 2004.

[17] G. Kokkinidis and V. Christophides. Semantic query routing and processing in P2P database systems: The ICS-FORTH SQPeer middleware. In *Proc. of EDBT*, pages 486–495, 2004.

[18] W. Le, S. Duan, A. Kementsietsidis, F. Li, and M. Wang. Rewriting queries on SPARQL views. In *Proc. of WWW*, pages 655–664, 2011.

[19] K. Makris, N. Bikakis, N. Gioldasis, and S. Christodoulakis. SPARQL-RW: transparent query access over mapped RDF data sources. In *Proc. of EDBT*, pages 610–613, 2012.

[20] K. Makris, N. Gioldasis, N. Bikakis, and S. Christodoulakis. Ontology mapping and SPARQL rewriting for querying federated RDF data sources. In *Proc. of OTM*, pages 1108–1117, 2010.

[21] W. Nejdl. Design issues and challenges for RDF- and schema-based peer-to-peer systems. In *Proc. of DBISP2P*, page 1, 2003.

[22] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. S. A. Naeve, M. Nilsson, M. Palmer, and T. Risch. Edutella: A P2P networking infrastructure based on RDF. In *Proc. of WWW*, pages 604–615, 2002.

[23] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing strategies for RDF-based peer-to-peer networks. *WWW*, 1(2):177–186, 2003.

[24] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *TODS*, 34(3):16:1–16:45, 2009.

APPENDIX

Algorithm 1: Using the chase to compute the certain answers $ans(q, \mathcal{P}, D)$.

Data: Graph pattern query q, system $\mathcal{P}$, stored instance D.
Result: The set $\mathbf{t}$ of the certain answers $ans(q, \mathcal{P}, D)$.
Initialize instance $J = \emptyset$;
`/* Chase procedure to generate a universal solution */`
while *some of the mappings of* $\mathcal{P}$ *are not satisfied in* J **do**
  case *(*$d \not\subseteq J$ *for some* $d \in D$*):*
    add d to J;
  case *(for some graph mapping assertion in* $\mathcal{P}$*, we have* $Q^J \not\subseteq Q'^J$*):*
    for *each tuple* $t \in Q^J \setminus Q'^J$ **do**
      generate the boolean query bQ' by substituting t in the free variables Q';
      add triples to J generating new blank nodes, such that $bQ'^J = true$;
    case *(for some equivalence mapping to* $\mathcal{P}$*, we have* $c \not\equiv_e c'$*)*
      switch $subjQ(c)^{*J} \not\subseteq subjQ(c')^{*J}$ **do**
        for *each tuple* $(p, o) \in (subjQ(c)^{*J} \setminus subjQ(c')^{*J})$ **do**
          add the triple (c', p, o) to J;
      switch $subjQ(c')^{*J} \not\subseteq subjQ(c)^{*J}$ **do**
        for *each tuple* $(p, o) \in (subjQ(c')^{*J} \setminus subjQ(c)^{*J})$ **do**
          add the triple (c, p, o) to J;
      switch $predQ(c)^{*J} \not\subseteq predQ(c')^{*J}$ **do**
        for *each tuple* $(s, o) \in (predQ(c)^{*J} \setminus predQ(c')^{*J})$ **do**
          add the triple (s, c', o) to J;
      switch $predQ(c')^{*J} \not\subseteq predQ(c)^{*J}$ **do**
        for *each tuple* $(s, o) \in (predQ(c')^{*J} \setminus predQ(c)^{*J})$ **do**
          add the triple (s, c, o) to J;
      switch $objQ(c)^{*J} \not\subseteq objQ(c')^{*J}$ **do**
        for *each tuple* $(s, p) \in (objQ(c)^{*J} \setminus objQ(c')^{*J})$ **do**
          add the triple (s, p, c') to J;
      switch $objQ(c')^{*J} \not\subseteq objQ(c)^{*J}$ **do**
        for *each tuple* $(s, p) \in (objQ(c')^{*J} \setminus objQ(c)^{*J})$ **do**
          add the triple (s, p, c) to J;
`/* End of chase */`
compute the certain answers $\mathbf{t} := q^J$;
`/* The certain answers are generated by evaluating the query over the universal solution */`
return $\mathbf{t}$;