RDF-F: RDF Datatype inFerring Framework

Towards Better RDF Document Matching

Irvin Dongo¹ · Judith Cardinale² · Richard Chbeir¹

Received: 1 April 2018 / Revised: 28 May 2018 / Accepted: 29 May 2018 / Published online: 11 June 2018
© The Author(s) 2018

Abstract

In the context of RDF document matching/integration, the datatype information, which is related to literal objects, is an important aspect to be analyzed in order to better determine similar RDF documents. In this paper, we present an **RDF Datatype in Ferring Framework**, called RDF-F, which provides two independent datatype inference processes: 1) a four-step process consisting of (i) a predicate information analysis (i.e., deduce the datatype from existing range property), (ii) an analysis of the object value itself by a pattern-matching process (i.e., recognize the object lexical space), (iii) a semantic analysis of the predicate name and its context, and (iv) generalization of Numeric and Binary datatypes to ensure the integration; and 2) a non-ambiguous lexical-space-matching process, where literal values are inferred by the modification of their representation, following new lexical spaces. We evaluated the performance and the accuracy of both processes with datasets from DBpedia. Results show that the execution time of both indicators is linear and their accuracy can increase up to 97.10 and 99.30%, respectively.

Keywords Datatype · Inference · Semantic Web

1 Introduction

One of the main benefits offered by the Semantic Web initiative is the increased support of data sharing and the description of real resources on the Web, by defining standard data representation models such as RDF, the Resource Description Framework. Particularly, heterogeneous RDF documents can express similar concepts using different vocabularies. Hence, many efforts focus on describing the similarity between concepts, properties, and

relations to support RDF document matching/integration [1, 3, 22].

Indeed, RDF describes resources as triples: $\langle \text{subject}, \text{predicate}, \text{object} \rangle$, where subjects, predicates, and objects are all resources identified by IRIs.¹ Objects can also be literals (e.g., a number, a string), which can be annotated with optional type information, called **datatype**. This latter is a classification of data, which defines types of RDF, adopted from XML Schema [25]. There are two classes of datatypes: simple and complex. Simple datatypes can be primitive (e.g., boolean, float), derived (e.g., long, int derived from decimal), or user defined, which are built from primitive and derived datatypes by constraining some of its properties (e.g., range, precision, length, format). Complex datatypes contain elements defined as either simple or complex datatypes.

The W3C Recommendation (proposed in [24]) points out the importance of the existence of datatype annotations

✉ Irvin Dongo
irvin.dongo@univ-pau.fr

Judith Cardinale
ycardinale@usb.ve

Richard Chbeir
richard.chbeir@univ-pau.fr

¹ Univ Pau & Pays Adour / E2S-UPPA, LIUPPA, EA3000, 64600 Anglet, France

² Departamento de Computación y Tecnología de la Información, Universidad Simón Bolívar, Caracas, Venezuela

¹ Internationalized Resource Identifier. An extension of URIs that allows characters from the Unicode character set.

to detect entailments between objects that have the same datatype, but a different value representation. For example, if we consider two distinct triples containing the objects `''20.000''` and `''20.0''`, these objects are considered as different, because of the missing datatype. However, if they were annotated as follows: `''20.000''^^xml:decimal` and `''20.0''^^xml:decimal`, then one can conclude that both objects are identical. Works on XML Schema matching proved that the presence of datatype information, constraints, and annotations on an object improves the similarity between two documents (up to 14%) [2]. Moreover, recent studies in the context of XML/RDF document matching have performed an analysis of datatypes to increase the compatibility/integration among data [9, 18, 23, 29]. However, a huge quantity of RDF documents is incomplete or inconsistent in terms of datatypes [15, 27]. Hence, when datatypes are missing, datatype inference emerges as a new challenge in order to obtain more accurate RDF document matching results.

Two categories of approaches have been proposed in the literature to cope with datatype inference:

- Several *theoretical* studies were conducted in the context of XML Schema Definition (XSD) [6, 7, 14]. They mainly infer simple datatypes by a hierarchy among the candidate datatypes obtained by a pattern-matching process on the format of the values, i.e., the characters that make unique a datatype, which is called *lexical space* according to the W3C Recommendation [25]. These works consider a limited number of simple datatypes (e.g., date, decimal, integer, boolean, and string) and choose the most specific datatype among the candidates. However, datatypes, as `gYear` (e.g., 1999), cannot be determined (since it is identified as an `integer`). Also, the other theoretical studies in the context of programming languages and OWL have focused on inferring complex datatype through axioms, assigned operations, and inference rules [11, 16, 26], without considering simple datatypes.
- There are many *tools* available on the Web that infer datatypes by mainly a pattern-matching process of the lexical spaces. As unknown inference criteria are used, each tool provides different datatypes for the same XML data.

Thus, in the context of RDF document matching/integration, current works are not suitable mainly for three reasons:

1. Hierarchy-based methods [6, 7, 14] cannot infer all simple datatypes, since they consider a reduced set of datatypes and there are intersections between datatype lexical spaces (e.g., 1999 can be an `integer` or a

`gYear` according to the lexical space of both W3C datatypes);

2. Complex datatype inference methods [4, 5, 11, 16, 31] cannot be applied to simple datatypes, since in RDF, a simple datatype is an atomic value associated with a predicate; and
3. Tools available on the Web [12, 21, 32] have unknown criteria of inference, providing different datatypes for the same data.

In a previous work, we proposed a framework that considers, in addition to the lexical space analysis, the analysis of the predicate information related to the object [10]. It consists of four steps: (i) analysis of predicate information, such as *range property* that defines and qualifies the type of the object value; (ii) analysis of lexical space of the object value, by a pattern-matching process; (iii) semantic analysis of the predicate and its semantic context, which consists in identifying related words or synonyms that can disambiguate two datatypes with similar lexical space; and (iv) generalization of Numeric and Binary datatypes, to ensure a possible integration among RDF documents.

Although our previous approach was able to detect many cases, some are still undetected. For example, according to this proposal, `string`, `decimal`, and `base64Binary` are candidate datatypes for the literal value `"1"`. Moreover, primitive datatypes were only considered in the study, but derived datatypes are also part of simple datatypes; thus, the inference is incomplete for simple datatypes. For that, we extend our inference datatype framework, called **RDF Datatype in Ferring Framework** (RDF-F), by proposing a new process based on the modification of the existing literal values through new non-ambiguous lexical spaces as an alternative to the four-step process to infer simple datatypes (primitive and derived). We focus on eliminating the ambiguity among lexical space representations, due to the high performance obtained in our previous study.

In summary, this study has the following contributions:

1. New lexical space representations based on the ones proposed by W3C;
2. A modification of the *Apache Jena*² sources to support the new lexical representation of datatypes and to keep the interoperability among existing Semantic Web services; and
3. An exhaustive validation of our framework through more experiments for the Semantic Web.

Additionally, we detail our proposal presented in [10], by adding more technical information. This paper is organized as follows: Sect. 2 presents a motivating scenario to

² Apache Jena is a free and open-source Java framework for building Semantic Web and Linked Data applications - <https://jena.apache.org>.

illustrate the importance of datatypes. Section 3 surveys the related literature. Definitions and RDF terminologies are presented in Sect. 4. Section 5 describes our inference approach. A complexity analysis is presented in Sect. 6. Section 7 shows the experiments to evaluate the accuracy and performance of our approach. Finally, we present conclusions in Sect. 8.

2 Motivating Scenario

In order to illustrate the importance of datatype information in RDF document matching, we consider a scenario in which we show the need to integrate three RDF documents with similar concepts (resources), but based on different vocabularies. Figure 1 shows three concepts from three different RDF documents. Figure 1a, b describes the concept `Light Switch`, with a property (predicate) `isLight`, whose datatype is `boolean`. However, they are represented with different lexical spaces: binary lexical space with a value `1` in Fig. 1a and string lexical space with a value `true` in Fig. 1b. In both cases, `isLight` property expresses the state of the light switch (i.e., turned on or turned off). Figure 1c shows the concept `Light Bulb`, with a property `Light`, whose datatype is `float`, and property `weight` with datatype `double`.

For their integration, it is necessary to analyze the information of related concept properties. Intuitively, considering the datatype information, one can say that:

1. Both `Light Switch` concepts from Fig. 1a, b are similar, since their properties are similar: The `isLight` property is `boolean` in both cases, and boolean literals can be expressed either as binary values (0 or 1) or as string values (`true` or `false`) according to the W3C [25].
2. `Light Bulb` concept is different from the other ones. Indeed, the `Light` property is expressed with `float`

values, expressing the light intensity, that has nothing to do with light switch state (i.e., turned on or turned off).

When the datatype information is missing and the integration is made only based on literals, several problems may emerge related to the *ambiguity* of properties. Contrary to our intuition, concepts in Fig. 1a, b are incompatible because of the use of different lexical spaces (i.e., value `1` is not compatible with the value `true`, which can be considered as a `string` datatype instead of `boolean`). Moreover, the integration of concept `Light Switch` from Fig. 1a with concept `Light Bulb` from Fig. 1c will be possible, even though it is incorrect. The `Light` properties of both respective documents are compatible because the lexical spaces of their values are the same (1 and 1250, respectively, can be `integer`). With the presence of datatype information, we can avoid this ambiguity even if the lexical spaces of the values are compatible.

Thus, an approach capable of inferring the datatype from the existing information is needed when one needs to match or integrate RDF documents.

In the following section, we survey existing works on datatype inference. We highlight their limitations and discuss their possible applications on RDF document matching/integration.

3 Related Work

To the best of our knowledge, no prior work manages simple datatype inference for RDF documents. However, datatype inference has been addressed in other contexts, such as (theoretical approaches): XML Schema Definition (XSD) [6, 7, 14], programming languages [4, 5, 11, 16, 31], and OWL [13, 19, 26, 28]. Moreover, we have revised some tools available on the Web for XSD inference

Fig. 1 Three concepts from three different RDF documents. **a** Light switch. Datatype boolean: binary lexical space (0, 1). **b** Light switch. Datatype boolean: string lexical space (`false`, `true`). **c** Light bulb

[12, 21, 32]. To evaluate the existing works, we have identified the following criteria of comparison:

1. Data criteria

- Consideration of *simple datatypes*, since this is the scope of the work;
- Analysis of *local information*, such as object values and predicates;
- Analysis of *external information*, since the Semantic Web allows the integration of external resources;

2. Feature criteria

- *Suitability* for the Semantic Web, the whole method should be objective, complete, and applicable for any domain.

Following sections describe the theoretical approaches and the revised tools.

3.1 Theoretical Approaches

According to similar solutions, we classify the existing works into three groups: *hierarchy-based* approaches, where datatypes are inferred using hierarchies over candidate datatypes obtained by a matching process of the lexical spaces; *function-based* approaches, where axioms, operations, and constructions are used to infer datatypes in the context of programming languages; and *knowledge-based* approaches using probabilities among candidate datatypes and external services to provide an analysis of local information. We describe the groups in the following sections:

3.1.1 Hierarchy-Based Approaches

In the inference of XSD from XML documents, the authors in [14] reduce the datatypes to a small set of values (date, decimal, integer, boolean, and string). They propose a hierarchy between the reduced datatypes according to the *lexical spaces* of the W3C Recommendation (see Fig. 2). The proposal returns the most specific datatype that subsumes the candidate datatypes obtained by the pattern matching of the values. However, a *gYear* value is reduced to *integer*, which is incorrect. In the same context, the author of [6, 7] proposes an inference method based on a hierarchy applied to a set of candidate datatypes. This set contains some derived datatypes of numeric group as *nonNegativeInteger*, *unsignedInteger*, and *unsignedShort*. In this case, the smallest datatype is chosen. For example, for a literal value 1999, whose datatype is *gYear*, the smallest among the candidate datatypes is *unsignedShort*, according to

Fig. 2 Hierarchical structure to infer datatypes. Solid lines describe strictly hierarchical relations, while the dotted line shows a loose relation [14]

Fig. 3 Hierarchical structure to infer datatypes. The most general datatype is *string*, while the most specific one is *unsignedByte* [6]

the hierarchy shown in Fig. 3. Table 1 shows the *lexical spaces* of simple datatypes according to the W3C.

3.1.2 Function-Based Approaches

In the context of programming languages, the authors in [11] focus on inferring complex datatypes, modeling them as a collection of constructor, destructor, and coercion functions. Other works [16, 31] also use axioms and pattern matching over the constructors of the datatype during the inference process. In [4, 5], operations and a syntax associated with datatypes are analyzed to infer complex datatypes. Simple datatypes such as *date* and *integer* are mainly inferred by a pattern-matching process of the value format using the *lexical spaces*. However, several simple datatypes having intersection among their *lexical spaces* as *gYear* and *integer* cannot be inferred using this pattern-matching process.

3.1.3 Knowledge-Based Approaches

In the context of OWL, the authors in [26] propose a rule-based method to heuristically generate datatype information by exploiting axioms in a knowledge base. They assign

Table 1 Lexical space for primitive datatypes (W3C Recommendation [25])

Datatype	Lexical space	Examples
string	Any character	“Example 123”
duration	PnYnMnDTnHnMNS	P1Y2M3DT10H30M
dateTime	CCYY-MM-DDThh:mm:ss-UTC	1999-05-31T13:20:00-05:00
time	hh:mm:ss	13:20:00-05:00
date	CCYY-MM-DD	1999-05-31
gYearMonth	CCYY-MM	1999-05
gYear	CCYY	1999
gMonthDay	–MM-DD	–05-31
gDay	–DD	–31
gMonth	–MM–	–05
boolean	true, false, 1, 0	false
base64Binary	Base64-encoded	0YZZ
hexBinary	Hex-encoded	0FB7
float	32-bit floating point type	12.78e–2, 1999
decimal	Arbitrary precision	12.78e–2, 1999
double	64-bit floating point type	12.78e–2, 1999

datatype probabilities to the assertions. In the domain of health care, the work presented in [28] proposes a datatype recognition approach (inference type) by associating a weight with each predicate, using support vector machines and by building a dictionary to map instances. For [20], the Semantic Web needs an incremental and distributed inference method due to the long ontology size. The authors use a parallel and distributed process (MapReduce) to “reduce” the “map” of new inference rules. The authors in [19] state that DBpedia only provides 63.7% of datatype information. Hence, they propose an approach to discover complex datatypes in RDF datasets by grouping entities according to the similarity between incoming and outgoing properties. They also use a hierarchical clustering and the confidence of types for an entity. Although the use of knowledge and inference rules can infer datatypes where a specific information is known (e.g., type of properties, knowledge database), RDF data are not always available with its respective ontology, which makes impossible the task of detecting rules.

In [13], the authors analyze two types of predicates: object property (semantic type, e.g., `dbpedia:BarackObama`) and datatype property (syntactic type, e.g., `xsd:string`). They propose an approach to infer the semantic type of string literals using the word detection technique called Stanford CoreNLP³ to identify the principal term and the UMBC⁴ semantic similarity service to

discover the semantic class. However, a semantic type is not always related to the same datatype, since it depends on the datatype defined in the structure. For example, the same data can be expressed as a `string` or `integer` according to two different ontologies.

3.2 Tools

Several tools that generate XSD from XML documents are provided in the literature to infer the type of data from existing values (*lexical spaces*), such as XMLgrid [32], FreeFormatted [12], and XmlSchemaInference by Microsoft [21]. However, they do not share a standard process to infer datatypes. For example, the attributes `weight` and `isLight` from the following XML document extracted from Fig. 1 have different inferred datatypes according to these three tools:

```
<Light_Bulb>
  <Light>1250</Light>
  <weight>30.00</weight> </Light_Bulb>
<Light_Switch>
  <isLight>1</isLight>
</Light_Switch>
```

- XMLgrid infers `weight` as `double` and `isLight` as `int`;
- FreeFormatted infers `weight` as `float` and `isLight` as `byte`;
- While XmlSchemaInference infers `weight` as `decimal` and `isLight` as `unsignedByte`.

³ CoreNLP is a natural language analysis tool for text that extracts particular relations, datatypes, etc. - <http://stanfordnlp.github.io/CoreNLP/>.

⁴ Semantic similarity service that analyzes semantic relations between words/phrases extracted from WordNet - <http://swoogle.umbc.edu/>.

Table 2 Related work classification

Work	Inference method	Requirements				
		Simple datatypes	Data criteria		Suitability	
			Local	External	XML/XSD	RDF-OWL
[6, 7, 14]	Hierarchy/lexical space	Reduced set	✓	X	✓	X
[4, 5, 11, 16, 31]	Functions (axioms, operations, and constructors)	Only complex	✓	X	✓	X
[19, 26, 28]	Knowledge (inference rules)	Only complex	✓	X	X	✓
[13]	Knowledge (semantic analysis)	Only string	✓	✓	X	✓
Tools: [12, 21, 32]	Not provided	Not provided	✓	X	✓	X
[10]	IRI information, Lexical space, Semantic analysis, Generalization	Only primitive	✓	✓	X	✓

Without known criteria, these existing tools cannot be adopted to infer properly datatypes of RDF resources.

Table 2 summarizes the existing approaches according to our criteria. Note that only our previous work satisfies all the defined requirements. However, derived datatypes were not considered. Following sections describe some RDF terminologies and definitions in order to formalize our datatype inference framework before presenting our proposal in detail.

4 RDF Terminologies and Definitions

RDF is the *common format* to describe resources that represent the abstraction of an entity (document, abstract concept, person, company, etc.) in the real world. RDF uses IRIs, blank nodes, and literal nodes as elements to build triples and provide relationships among resources.

The RDF Schema (RDFS) is a set of classes with certain properties (vocabulary), which are extensions of the basic RDF vocabulary [8]. RDFS defines properties to better describe resources. For example, the `rdfs:domain` property designates the type of subject that can be associated with a predicate and the `rdfs:range` property designates the type of object. The Semantic Web proposes an implicit representation of the datatype property in the

literal object as a description of the value (e.g., `^^value'^'^xml:string`). Definition 1 presents the formal definition of a simple datatype according to W3C [17].

Definition 1 *Simple datatype* (dt): In RDF, a simple datatype, denoted as *dt*, is characterized by:

(i) a value space, denoted as $VS(dt)$, which is a non-empty set of distinct valid values; (ii) a lexical space, denoted as $LS(dt)$, which is a non-empty set of Unicode strings; and (iii) a total mapping from the lexical space to the value space, denoted as $L2V(dt)$ [17].

The datatype `boolean` from Fig. 1a has the following characteristics:

- $VS(boolean) = \{true, false\}$;
- $LS(boolean) = \{“true”, “false”, “1”, “0”\}$;
- $L2V(boolean) = \{“true” \Rightarrow true, “false” \Rightarrow false, “1” \Rightarrow true, “0” \Rightarrow false\}$

Table 3 presents several sets of RDF elements, that we use in our formal approach description in the following sections.

As we mentioned before, RDF links resources by the use of IRIs, blank node, and literal nodes making atomic structures called triples or statements. A triple is defined as follows.

Table 3 Description of sets

Set	Description
<i>I</i>	A set of IRIs defined as: $I = \{i \mid i \text{ is an IRI}\}$
<i>L</i>	A set of literal nodes defined as: $L = \{l \mid l \text{ is a literal node}\}$
<i>BN</i>	A set of blank nodes defined as: $BN = \{bn \mid bn \text{ is a blank node}\}$
<i>DT</i>	A set of datatypes defined as: $DT = \{dt \mid dt \text{ is a datatype}\}$
<i>SDT</i>	The set of simple datatypes proposed by the W3C defined as: $SDT = \{string, duration, dateTime, time, date, gYearMonth, gDay, gMonth, boolean, base64Binary, hexBinary, float, decimal, double\}$

Definition 2 *Triple* (t) [30]: A triple is defined as an atomic structure consisting of a 3-tuple with a subject (s), a predicate (p), and object (o), denoted as $t : \langle s, p, o \rangle$, where:

- $s \in I \cup BN$ represents the subject to be described;
- p is a property defined as an IRI in the form `namespace_prefix : property_name`; `namespace_prefix` is a local identifier of the IRI, where the property (`property_name`) is defined;
- $o \in I \cup BN \cup L$ describes the object.

The predicate (p) is also known as the property of the triple.

The example presented in Fig. 1 underlines four triples with different RDF resources, properties, and literals, as follows:

- t_1 : $\langle \text{Light Switch}, \text{house:is Light}, 1 \rangle$
- t_2 : $\langle \text{Light Switch}, \text{house:is Light}, \text{true} \rangle$
- t_3 : $\langle \text{Light Bulb}, \text{light:Light}, 1250 \rangle$
- t_4 : $\langle \text{Light Bulb}, \text{dbp:weight}, 30.00 \rangle$

In the following section, we describe our datatype inference framework.

5 RDF-F: Our Inference Process Approach

Our datatype inference approach relies on two independent processes: (i) a four-step analysis, where the literal values and the predicates related to them are analyzed syntactically and semantically, and (ii) a non-ambiguous lexical space matching, where the initial literal values are modified according to new lexical space representations and inferred by a matching process. Figure 4 shows the two processes of our inference framework. The input of our framework is an RDF Description which can be represented in different serializations formats (such as RDF/XML, Turtle, N3) and the user parameters (i.e., type of process, inference steps,

and their order). The output is an RDF Description with its respective inferred datatypes.

5.1 RDF-F: Four-Step Process

This process considers the annotations on the predicate, the specific format of literal object values, the semantic context of the predicate, and the generalization of datatype for Numeric and Binary groups. Each step can be applied independently and in different orders according to user parameters.

A description of each step is presented as follows.

5.1.1 Predicate Information Analysis (Step 1)

In a triple $t : \langle s, p, o \rangle$, the predicate p establishes the relationship between the subject s and the object o , making the object value o a characteristic of s . Information (properties) such as `rdfs:domain` and `rdfs:range` can be associated with each predicate to determine the type of subject and object, respectively. To deduce the simple datatype of a particular literal object, we propose to inspect the property `rdfs:range`, when exists. We formally describe this Step 1 with the following definitions and rule.

Definition 3 *Predicate Information (PI)*: Given a triple $t : \langle s, p, o \rangle$, the Predicate Information is a function, denoted as $PI(t)$, that returns a set of triples defined as: $PI(t) = \{t_i \mid t_i = \langle s_i, p_i, o_i \rangle\}$, where:

- $s_i = t \cdot p \mid s_i$ is the subject of a triple t_i ;
- p_i is an RDF defined property $\in \{\text{rdfs:type}, \text{rdfs:label}, \text{rdfs:range}\}$;
- o_i is the object of t_i .

Table 4 shows the set of triples returned by the function Predicate Information, when applied on the property `dbp:weight` presented in Fig. 1c.

Fig. 4 Framework of our RDF inference process

Table 4 Example of the set of triples returned by the Predicate Information (PI) on `dbp:weight`

Subject	Predicate (property)	Object (value)
<code>dbp:weight</code>	<code>rdf:type</code>	<code>owl:DatatypeProperty</code>
<code>dbp:weight</code>	<code>rdfs:label</code>	<code>gewicht (g) (de)</code>
<code>dbp:weight</code>	<code>rdfs:label</code>	<code>gewicht (g) (nl)</code>
<code>dbp:weight</code>	<code>rdfs:label</code>	<code>peso (g) (pt)</code>
<code>dbp:weight</code>	<code>rdfs:label</code>	<code>poids (g) (fr)</code>
<code>dbp:weight</code>	<code>rdfs:label</code>	<code>weight (g) (en)</code>
<code>dbp:weight</code>	<code>rdfs:label</code>	<code>weight (g) (en)</code>
<code>dbp:weight</code>	<code>rdfs:range</code>	<code>xsd:double</code>
<code>dbp:weight</code>	<code>prov:wasDerivedFrom</code>	http://mappings.dbpedia.org/OntologyProperty:weight

Definition 4 *Predicate Range Information (PRI)*: Given a triple $t : \langle s, p, o \rangle$, the Predicate Range Information is a function, denoted as $PRI(t)$, that returns the value associated with the `rdfs:range` property, defined as:

$$PRI(t) = \begin{cases} t_i \cdot o & \text{if } \exists t_i \in PI(t) \mid t_i \cdot p = \text{rdfs:range,} \\ \text{null} & \text{otherwise.} \end{cases}$$

R1 : if $IA(t \cdot p) \Rightarrow dt(t \cdot o) = PRI(t)$.

Rule 1 verifies if the predicate of the triple is an IRI available on the Web (Definition 5), and by Definition 4, it determines if the `rdfs:range` property exists from the set of triples extracted by Definition 3. Algorithm 1 is the pseudo-code of how this rule can be implemented in high-level programming language.

Algorithm 1: Predicate Information Analysis

```

Input: Triple  $t = \langle s, p, o \rangle$ 
Output: Datatype  $dt$ 
1 if  $IA(t.p)$  then
2    $T \text{ triples} = PI(t);$  //Set of triples with information from predicate.
3   foreach triple in triples do
4     if  $triple.p == \text{rdfs:range}$  then
5        $dt = triple.o;$  //If range information exists, the datatype is returned.
6       return  $dt;$ 
7   return unknown; //Range information does not exist.
8 return unknown; //There is not external information available.
```

Applying Definition 4 to the set of Predicate Information (PI) of the property `dbp:weight` (see Table 4), the Predicate Range Information function returns the value `xsd:double`.

Definition 5 *IsAvailable (IA)*: Given a predicate p , *IsAvailable* is a boolean function, denoted as $IA(p)$, that verifies if p is an IRI available on the Web:

$$IA(p) = \begin{cases} \text{true} & \text{if } p \text{ returns code 200;} \\ \text{false} & \text{otherwise.} \end{cases}$$

Using the three previous definitions, we formalize our first inference rule.

Rule 1 *Datatype Inference by Predicate Information Analysis*: Given a triple $t : \langle s, p, o \rangle$, in which $o \in L$, the datatype of o is determined as follows:

As an input, the algorithm receives the triple $t : \langle s, p, o \rangle$, whose object datatype is to be determined. If the IRI representing the predicate exists (line 1—Definition 5), the link is explored to extract all available information as triples (line 2—Definition 3). For example, if `dbp:weight` (more specifically, <http://www.dbpedia.org/ontology/weight>) is the predicate, we can get the list of triples shown in Table 4. (Each row represents a triple.) If among these triples, there is the property `rdfs:range`, then its associated object value, which is the datatype, is returned (lines 3 to 5). Otherwise, an unknown datatype is returned (lines 7—Definition 4).

As shown in Fig. 1c, the output of the algorithm will be the datatype `xsd:double`.

This algorithm examines external information and is independent of the query language. Rule 1 can be implemented as a simple SPARQL query, such as:

Fig. 5 Datatype lexical space intersection

```

select distinct ?datatype where
  {?subject ?predicate ?literal.
   ?predicate rdfs:range ?datatype }

```

where `?subject` `?predicate` `?literal` are the triple to be analyzed; `?predicate` is the analyzed predicate; and `?datatype` is the returned result.

The following step analyzes the lexical space of a literal object in order to infer its datatype.

5.1.2 Datatype Lexical Space Analysis (Step 2)

According to Definition 1, a datatype is a 3-tuple consisting of: (i) a set of distinct valid values, called *value space*; (ii) a set of lexical representations, called *lexical space*; and (iii) a total mapping from the lexical space to the value space. In some cases, the datatype can be inferred from its *lexical space*, when it is uniquely formatted (e.g., value 1999-05-31 matches with the format CCYY-MM-DD, which is the *lexical space* of datatype `date`). However, in other cases (such as `boolean`, `gYear`, `decimal`, `double`, `float`, `integer`, `base64Binary`, and `hexBinary`), the *lexical spaces* of datatypes have common characteristics, leading to a certain ambiguity (e.g., value 1999 matches with *lexical spaces* of `gYear` and `float` – see Table 1). Figure 5 illustrates graphically the *lexical space* intersections of W3C simple datatypes (primitives and integer).

To compare the datatype lexical spaces with the literal values, we establish an order based on the lexical spaces intersections (from a general lexical space to a specific one).

To analyze the *lexical spaces*, we propose the following definition.

Definition 6 *Candidate Datatypes (CDT)*: Given a literal object o , the set of its candidate datatypes is determined by the function *Candidate Datatypes*, defined as: $CDT(o) = \{dt \mid dt \in SDT \wedge LS(o) = LS(dt)\}$

By Definition 6, the set of candidate datatypes of the object literal value 1 presented in Fig. 1a is: $CDT(1) = \{\text{float}, \text{decimal}, \text{double}, \text{hexBinary}, \text{base64Binary}, \text{integer}, \text{boolean}, \text{string}\}$.

Based on this definition, we formally define our second inference rule as follows.

Rule 2 *Datatype Inference by Lexical Space*: Given a triple $t : \langle s, p, o \rangle$, in which $o \in L$, the datatype of o is determined as follows:

$$\begin{aligned}
 &\mathbf{R2} : dt(t \cdot o) \\
 &= \begin{cases} \text{string} & \text{if } |CDT(o)| = 1, \\ cdt_i \mid cdt_i \in CDT(o) \wedge cdt_i \neq \text{string} & \text{if } |CDT(o)| = 2, \\ \text{unknown} & \text{otherwise.} \end{cases}
 \end{aligned}$$

Rule 2 analyzes the number of possible datatypes of a literal object value. The order to analyze the lexical space of each datatype is established by the lexical space intersections. In all cases, the datatype `string` is a candidate datatype, since it has the most general *lexical space* (see Fig. 5); if the number of candidate datatypes is one, then the only datatype, which is `string`, is returned. If the number of candidate datatypes is two, then the other datatype is returned. Otherwise, we have an ambiguous case and any datatype, different from `string`, can be provided. Hence, the inference process remains incomplete due to the ambiguous cases and further analysis is needed.

A pseudo-code following the definition of Rule 2 is proposed in Algorithm 2.

Algorithm 2: Lexical Space Analysis

```

Input: Triple  $t = \langle s, p, o \rangle$ 
Output: Datatype  $dt$ 
1  $DT\ dt\_list = \text{new } [Datatype(String)];$  // Every value is a string (see Fig. 2).
2  $dt\_list = CDT(t.o);$  // Pattern-matching verification.
3 if  $|dt\_list| > 2$  then
4   return unknown; // Ambiguous case.
5 else
6   if  $|dt\_list| == 1$  then
7      $dt = \text{fetch}(dt\_list, 0);$  // Non ambiguous case, it is string.
8   else
9      $dt = \text{fetch}(dt\_list, 1);$  // Non ambiguous case, string and other datatype.
10  return  $dt;$ 

```

The algorithm receives a triple $t : \langle s, p, o \rangle$ and returns the datatype that can be associated with the object. An initial list is initialized with the datatype `string`, because any object value is a `string` (line 1). According to the *lexical spaces* defined by the W3C (see Table 1), the list of candidate datatypes is generated by a pattern-matching process (line 2 in Algorithm 2—Definition 6) following the order obtained from the lexical space intersections. If the number of candidate datatypes is more than 2, we are under an ambiguous case, since the lexical space of the literal value matches with several lexical spaces of the datatypes (lines 3–4 of Algorithm 2). If we have only `string` as a candidate datatype, then this is the returned information (line 7 of Algorithm 2). If we get two candidate datatypes, one of them is a `string` datatype and the other one is the datatype returned for the object value (line 9 of Algorithm 2).

The following step analyzes semantically the predicate of the literal object through the definition of context rules.

5.1.3 Predicate Semantic Analysis (Step 3)

In the presence of ambiguous cases (unknown), a semantic analysis of the predicate needs to be performed. The predicate name can define the context of the information in a scenario where the data are consistent. Regarding the W3C datatype *lexical spaces*, the datatypes `boolean`, `gYear`, `decimal`, `double`, `float`, `integer`, `base64Binary`, and `hexBinary` are ambiguous. However, the ambiguity of `boolean`, `gYear`, and `integer`, in some specific scenarios, can be resolved by examining the context of its predicate according to a knowledge base.

For example, the predicate `dbp:dateOfBirth` has the context `date`, and then it is possible to assume `gYear` as the datatype; the predicate `dbp:era` has the context `period` and the datatype assigned can be `integer`; however, for predicate `dbp:salary`, it is possible to assign datatypes `decimal`, `double`, or `float`; the ambiguous case persists.

In order to describe our inference process in this step, we formalize a knowledge base as follows:

Definition 7 *Knowledge Base (KB)*: A knowledge base (thesaurus, taxonomies, and ontologies) provides a framework to organize entities (words/expressions, generic concepts, etc.) into a semantic space, which is capable of capturing meaning. Our knowledge base has the following defined functions:

- *Similarity (sim)*: Given two entities n and m , Similarity is a function, denoted as $\text{sim}(n, m)$, that returns the value of the relation among both entities:

$$\text{sim}(n, m) = \text{A relation value} \in [0, 1] \text{ between } n \text{ and } m \text{ according to KB.}$$

- *IsPlural (IP)*: Given an entity n , IsPlural is a function, denoted as $IP(n)$, that returns True if the entity n is plural:

$$IP(n) = \begin{cases} \text{True} & \text{if } n \text{ is plural according to KB;} \\ \text{False} & \text{otherwise.} \end{cases}$$

- *IsCondition (IC)*: Given an entity n , IsCondition is a function, denoted as $IC(n)$, that returns True if the entity n is a condition:

$$IC(n) = \begin{cases} \text{True} & \text{if } n \text{ is a condition according to KB;} \\ \text{False} & \text{otherwise.} \end{cases}$$

In this scenario, our knowledge base is reduced to the relations among words.

The semantic context is formalized, based on the knowledge base, as follows:

Definition 8 *Context (ct)*: A context is a related word (synonym), which clarifies or generalizes the domain of a word. It is associated with a relation value according to a Knowledge Base. A context is denoted as a 3-tuple $ct : \langle w, y, v \rangle$, where w is a word; y is a related word of w ; and v is a relation value between w and y , $\text{sim}(w, y) \in [0, 1]$.

Definition 9 *Set of contexts (CT)*: Given a word w , its set of contexts is defined as $CT = \{ct_i \mid ct_i : \langle w, y_i, v_i \rangle \text{ is a context of } w\}$.

For example, from Fig. 1c, the set of contexts of predicate `weight` is: $CT = \{\langle \text{weight}, \text{load}, 0.8 \rangle, \langle \text{weight}, \text{heaviness}, 0.5 \rangle, \langle \text{weight}, \text{obesity}, 0.4 \rangle, \langle \text{weight}, \text{size}, 0.3 \rangle\}$.

Definition 10 *Predicate Context (PC)*: Given a triple $t : \langle s, p, o \rangle$ and a threshold h , Predicate Context is a function, denoted as $PC(t, h)$, that returns a set of contexts defined as:

$$PC(t, h) = \{ct_i \mid ct_i : \langle p, \text{property_name}_i, y_i, v_i \rangle, v_i \geq h\}.$$

The context can determine the datatype for some literal objects through a semantic analysis, and then we assume three scenarios for an ambiguous case:

- If `date` is in the context (e.g., $\langle \text{word}, \text{date}, 0.5 \rangle$, with $h = 0.5$) and the literal value is a number (e.g., 1999), then the datatype is `gYear` because `gYear` (1999) is a part of datatype `date` (1999-05-31);
- If `period` is in the context (e.g., $\langle \text{word}, \text{period}, 0.5 \rangle$, with $h = 0.5$) and the literal value is a number (e.g., 3 months), then the datatype is `integer` because it is about quantity.
- However, if the context is `date`, the word from which we obtain the context cannot be plural, since plural words express quantities. Thus, in this case the word is related to the datatype `integer` according to our scenarios.

Definition 11 generalizes our scenarios to assign a datatype to a literal object, according to the context of its corresponding predicate name.

Definition 11 *Predicate Name Context (PNC)*: Given a triple $t : \langle s, p, o \rangle$, in which $o \in L$, and a threshold h , Predicate Name Context is a function, denoted as $PNC(t, h)$, that returns a datatype defined as:

$$PNC(t, h) = \begin{cases} \text{gYear} & \text{if } \exists ct_i \in PC(t, h) \mid ct_i.y_i = \text{date} \wedge \text{gYear} \in CDT(o) \\ & \wedge \neg IP(p, \text{property_name}); \\ \text{integer} & \text{if } \exists ct_i \in PC(t, h) \mid ct_i.y_i = \text{date} \wedge \text{integer} \in CDT(o) \\ & \wedge IP(p, \text{property_name}); \\ \text{integer} & \text{if } \exists ct_i \in PC(t, h) \mid ct_i.y_i = \text{period} \wedge \text{integer} \in CDT(o); \\ \text{unknown} & \text{otherwise.} \end{cases}$$

In addition, to determine a datatype as `boolean`, we assume that a word is defined as a condition in a knowledge base (e.g., WordNet).

Using the previous definitions, we formally define our third inference rule.

Rule 3 *Datatype Inference by Semantic Analysis*: Given a triple $t : \langle s, p, o \rangle$, in which $o \in L$, and a threshold h the datatype of o is determined as follows:

$$R3 : dt(t \cdot o) = \begin{cases} \text{boolean} & \text{if } \text{boolean} \in CDT(o) \wedge IC(p, \text{property_name}); \\ PNC(t, h) & \text{otherwise.} \end{cases}$$

Rule 3 returns the datatype of the object value when a defined context associated with the predicate exists. If that is not the case, we are still under an ambiguous case. Note that Rule 3 is proposed for a scenario where the data are consistent with the W3C Recommendations (e.g., self-descriptive names).

Algorithm 3 is a pseudo-code of our semantic analysis step. The algorithm receives the triple $t : \langle s, p, o \rangle$ to be analyzed. For the analysis of the predicate name, an external service is required in order to obtain the synonyms of the predicate name, called contexts (line 3 in Algorithm 3). If more than one defined context is available in the set of contexts (Definition 10), the algorithm returns the one which has the highest similarity value (line 20 in Algorithm 3). An unknown datatype is returned if no defined context is present.

Algorithm 3: Predicate Semantic Analysis

Input: Triple $t = \langle s, p, o \rangle$, float h
Output: Datatype dt

```

1 DT dt_list = CDT(t.o);
2 KB knowledge_base;
3 PC contexts = get_context_SERVICE(t.p.property_name, h);
4 DT datatypes = {};
5 if  $date \in contexts$  then
6   if  $knowledge\_base.IP(t.p.property\_name)$  and  $integer \in dt\_list$  then
7     dt = new Datatype(integer);
8     datatypes.add(dt);
9   else if  $!knowledge\_base.IP(t.p.property\_name)$  and  $gYear \in dt\_list$  then
10    dt = new Datatype(gYear);
11    datatypes.add(dt);
12 if  $period \in contexts$  and  $integer \in dt\_list$  then
13   dt = new RDFDatatype(integer);
14   datatypes.add(dt);
15 if  $knowledge\_base.IC(t.p.property\_name)$  and  $boolean \in dt\_list$  then
16   dt = new Datatype(boolean);
17   datatypes.add(dt);
18 if  $|datatypes| > 0$  then
19   datatypes.ORDER_BY_DESC()
20   return fetch(datatypes, 0);
21 return unknown;
```

The following step describes the generalization method for literal values that are part of Numeric and Binary groups.

5.1.4 Generalization of Numeric and Binary Groups (Step 4)

As an alternative to disambiguate the datatypes decimal, double, float, integer, base64Binary, and hexBinary, we propose two groups of datatypes: Numeric and Binary. In each group, we define a total order among the datatypes by considering *lexical space* intersection (see Fig. 5). Hence, for the Numeric group, we have decimal > double > float > integer and in the Binary group, base64Binary > hexBinary. According to these groups, we return the most general datatype, if all candidate datatypes belong only to one of these two groups.

Definition 12 *Generalization (G)*: Given a literal object o , the set of its candidate datatypes is reduced by the function Generalization, defined as:

$$G(o) = \{\text{string}\} \cup \{dt \mid dt \in CDT(o) \wedge (dt = \text{fetch}(\text{Numeric_Group}, 0) \vee dt = \text{fetch}(\text{Binary_Group}, 0))\}$$

Note that datatype string is always part of candidate datatypes. We formally define our fourth inference rule as follows.

Rule 4 *Datatype Generalization*: Given a triple $t : \langle s, p, o \rangle$, in which $o \in L$, the datatype of o is determined as follows:

R4 : $dt(t \cdot o)$

$$= \begin{cases} \text{string} & \text{if } |G(o)| == 1; \\ g_i \mid g_i \in G(o) \wedge g_i \neq \text{string} & \text{if } |G(o)| == 2; \\ \text{unknown} & \text{otherwise.} \end{cases}$$

However, we can have a case where an object value has decimal and base64Binary as candidate datatypes because of similar value representations and our inference approach cannot determinate the most appropriate datatype.

Algorithm 4 is a pseudo-code of a possible implementation of Rule 4. The algorithm receives the triple $t : \langle s, p, o \rangle$ to be analyzed. The list of candidate datatypes is reduced removing specific datatypes and keeping the most general ones (decimal and base64Binary) (line 2 in Algorithm 4). If the list of candidate datatypes has only a value, the datatype is string (line 4 in Algorithm 4); however, if there are two, the datatype is the second one (line 6 in Algorithm 4), since the first one is always string. If there are more than two datatypes, the ambiguity persists and this step is not able to produce a result.

Algorithm 4: Generalization of Numeric and Binary Groups

Input: Triple $t = \langle s, p, o \rangle$
Output: Datatype dt

```

1 DT dt_list = CDT(t.o);           //Candidate Datatypes
2 DT generalDT = G(t.o);           //General datatypes from Numeric and Binary groups.
3 if |generalDT| == 1 then
4   return new Datatype(string);
5 if |generalDT| == 2 then
6   return fetch(generalDT,1);
7 return unknown;                  // Ambiguous case.

```

Our first process of the RDF-F allows to improve the datatype analysis for RDF matching/integration by complying with the identified requirements (see Sect. 3): (i) the use of local available information, as the predicate value in *Step 1* and *Step 3* and the datatype lexical space in *Step 2*, as well as external available information, such the predicate information in *Step 1* and the predicate context in *Step 3*; and (ii) this method is objective and complete for the Semantic Web, since all simple datatypes are considered, which are available in the most common Semantic Web databases.

In the following section, we present our second process where new lexical space representations, for simple datatypes based on the W3C, are proposed.

5.2 RDF-F: Non-ambiguous Lexical-Space-Matching Process

Even though the first process of the RDF-F is performed, some datatypes could remain unknown. However, the inference process can be reduced to a simple matching

between lexical spaces and the format of literal values if we associate with each simple datatype lexical space a different representation. To achieve this, we extend the W3C definitions to provide such representations. Table 5 summarizes the set of representations using regular expressions. For instance, float values have a lexical representation consisting of a consonant “f” before a mantissa followed, optionally, by the character “E” or “e”, followed by an exponent. The exponent must be an integer. The mantissa must be a decimal number. The representations for exponent and mantissa must follow the lexical rules for integer and decimal. If the “E” or “e” and the following exponent are omitted, an exponent value of 0 is assumed. The special values positive and negative infinity and not-a-number have lexical representations INF, -INF, and NaN, respectively. Lexical representations for zero may take a positive or negative sign (e.g., f-1E4, f1267.43233E12, f12, f-0, f0, f-1E4, f1267.43233E12, f12, f-0, f0). The corresponding regular expression of datatype float representation is: $f[+-]?([0-9]*[.])?(E|e)?[0-9]+$.

Table 5 Lexical space representation defined as regular expressions

Simple datatypes		W3C	Proposal
Primitive	boolean	(1 0 true false)	b(1 0 true false)
	gYear	[1-9]{1,4}	y[1-9]{1,4}
	decimal	[+-]?([0-9]*[.])?[0-9]+	(de)[+-]?([0-9]*[.])?[0-9]+
	float	[+-]?([0-9]*[.])?(E e)?[0-9]+	f[+-]?([0-9]*[.])?(E e)?[0-9]+
	double	[+-]?([0-9]*[.])?(E e)?[0-9]+	d[+-]?([0-9]*[.])?(E e)?[0-9]+
	hexBinary	0[xX][0-9a-fA-F]+	hB0[xX][0-9a-fA-F]+
Derived	integer	[+-]?[0-9]+	I[+-]?[0-9]+
	negativeInteger	-[0-9]+	nI(-[0-9]+)
	nonNegativeInteger	0 (\+[0-9]+)	nNI(0 (\+[0-9]+))
	positiveInteger	\+[1-9]+[0-9]*	pI\+[1-9]+[0-9]*
	nonPositiveInteger	0 (-[0-9]+)	nPI(0 (-[0-9]+))
	long	[+-]?[0-9]+	l[+-]?[0-9]+
	int	[+-]?[0-9]+	i[+-]?[0-9]+
	short	[+-]?[0-9]+	s[+-]?[0-9]+
	unsignedLong	[0-9]+	uL[0-9]+
	unsignedInt	[0-9]+	uI[0-9]+
	unsignedShort	[0-9]+	uS[0-9]+

Using this simple solution, the lexical spaces become unique, and thus by a lexical space matching, we can infer the simple datatypes. However, changing the lexical spaces leads to change the processing engine (e.g., Jena) in order to provide compatibility between previous W3C lexical spaces and the new proposed ones.

In the following section, we describe a complexity analysis of our framework.

6 Complexity Analysis

As the framework relies on two independent processes, two different temporal complexity analyses need to be performed. A complexity analysis of the first process of our inference approach indicates a linear order performance in terms of the number of triples ($O(n)$).

- In Step 1, the predicate information of each triple is extracted to search the `rdfs:range` property, since the number of properties associated with the predicate of each triple (Definition 3) is constant, and then its execution order is of $O(n)$.
- In Step 2, for each triple a pattern matching is executed for all simple datatypes (finite number of executions); thus, it is of linear order ($O(n)$).
- In Step 3, for each triple, its set of contexts is extracted to determine the best related work (in a constant time); thus, its time complexity is also $O(n)$.
- Finally, Step 4 reduces the finite set of candidate datatypes (generalization) in a linear order ($O(n)$).

As the four steps are executed sequentially, the whole first inference datatype process exhibits a linear order complexity, $O(n)$.

The second process of our approach also indicates a linear order performance in terms of the number of triples ($O(n)$). This process is similar to the Step 2 performed in the four-step process. Each triple is analyzed by a pattern matching for all simple datatypes (finite number of executions).

The following section evaluates the accuracy and demonstrates the linear order performance of our proposal.

7 Experimental Evaluation

To evaluate and validate our inference approach, an online prototype system, called *RDF2rRDF*,⁵ was developed using PHP and Java. Figure 6 shows the graphic user interface of the prototype, where the processes of inference can be selected according to user preferences.

⁵ <http://rdf2rrdf.sigappfr.org/>.

For our four-step inference process, contexts in Step 3 were implemented using the semantic similarity service UMBC.⁶ Also, we used WordNet⁷ to recognize if a word is plural assuming that every word has a root lemma where the default plurality is singular. Additionally, we assumed in our implementation that a word is a *condition* if it has the prefix “is” or “has”. All these assumptions compose our knowledge base.

For our non-ambiguous lexical-space-matching process, we modified the Jena sources in order to support the new lexical space representations.⁸ The idea is to propose compatibility between previous lexical spaces and the new proposed ones through the modification of the Jena sources. New designs can adopt the proposed lexical spaces without losing their properties in existing services. Additionally, we implemented a tool where automatically the RDF data can be modified according to the new lexical space representations. RDF data have to be annotated with their respective datatypes to produce consistent documents.

Different Semantic Web databases are currently available on the Web (e.g., DBpedia, WordNet, GeoLinked data). However, they do not provide enough variety of datatypes. (WordNet considers only strings, and GeoLinked data consider complex datatypes.) In DBpedia dataset, only nine datatypes are present (integer, gYear, date, gMonthDay, float, nonNegative, double, Integer, and decimal). Consequently, we chose DBpedia as the dataset to perform our experiments.

Experiments were carried out on a MacBook Pro, 2.2 GHz Intel Core(TM) i7 with 16.00 GB, running a MacOS Sierra and using a Sun JDK 1.7 programming environment.

Our prototype was used to perform a large battery of experiments to evaluate the accuracy and the performance (execution time) of our approach in comparison with the related work. To do so, we considered two datasets:

- *Case 1*: 5603 RDF documents gathered from *DBpedia person data*,⁹ in which 1059822 triples, 38292 literal objects, and 8 different datatypes are available.
- *Case 2*: the whole *DBpedia person data* as a unique RDF document with 16842176 triples, in which only datatypes date, gMonthDay, and gYear are presented.

⁶ Semantic Similarity Service Computing, which is based on distributional similarity and Latent Semantic Analysis. UMBC service is available online, and an API is provided - <http://swoogle.umbc.edu/SimService/api.html>.

⁷ WordNet is a large lexical database of English (nouns, verbs, adjectives, etc.).

⁸ <http://rdf2rrdf.sigappfr.org/jena/>.

⁹ Information about persons extracted from the English and Germany Wikipedia, represented by the FOAF vocabulary - <http://wiki.dbpedia.org/Downloads2015-10>.

Fig. 6 Graphic user interface of our prototype RDF2rRDF

RDF Datatype Inference

Input

RDF/XML

Direct Input / [File or folder upload](#)

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:prov="http://www.w3.org/ns/prov#"
  xmlns:dbo="http://dbpedia.org/ontology/"
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
```

☒ Inference ☐ Comparison

This option infers all datatypes missed in the RDF document.

☒ 1. Four-Step Inference Process

- ☒ Predicate Information Analysis
- ☒ Lexical Space Analysis
- ☒ Predicate Semantic Analysis
- ☒ Generalization

☐ 2. Non-Ambiguous Lexical Space Matching Process

☒ New Lexical Space Representations

Start

7.1 Accuracy Evaluation

To evaluate the accuracy of our approach, we calculated the *F*-score, based on the Recall (*R*) and Precision (*PR*). These criteria are commonly adopted in information retrieval and are calculated as follows:

$$PR = \frac{Valid}{Valid + Invalid} \in [0, 1]$$

$$R = \frac{Valid}{Valid + Ambiguous} \in [0, 1]$$

$$F\text{-score} = \frac{2 \times PR \times R}{PR + R} \in [0, 1]$$

where *Valid* is the number of correctly inferred datatypes; *Invalid* is the number of wrongly inferred datatypes; and *Ambiguous* is the number of datatypes not inferred by our inference approach.

For our four-step process, in Case 1, we evaluated the accuracy and performance of each step, all the combinations (*Step 1 + Step 2*, *Step 1 + Step 3*, *Step 2 + Step 3*,

Step 1 + Step 4, *Step 2 + Step 4*, *Step 1 + Step 2 + Step 3*, *Step 1 + Step 2 + Step 4*, *Step 2 + Step 3 + Step 4*), and the whole inference process. The order of the whole inference process was established starting from a general solution (*Step 1*), that can be applied to all simple datatypes, until a specific solution for particular cases (*Step 3* and *Step 4*). Extra experiments were performed in order to evaluate the accuracy with respect to the existing approaches, and to measure the behavior of the process when some datatypes are available in the RDF data. In Case 2, we only evaluated the whole four-step process, since the aim was to evaluate the execution time when having a high number of triples.

For our non-ambiguous lexical space process, the literal values of Case 1 were modified according to the new lexical space representations. To do so, we used the developed tool available on our online prototype. As a high number of triples are available in Case 2, we used this dataset to evaluate the performance of this inference process.

Table 6 Accuracy evaluation of the four-step inference process

Four-step inference process	Accuracy evaluation					
	Valid	Invalid	Ambiguous	Precision (%)	Recall (%)	<i>F</i> -score (%)
Case 1: Step 1	24,033	26	14,233	99.89	62.81	77.12
Case 1: Step 2	16,898	537	20,857	96.92	44.76	61.24
Case 1: Step 3	2480	119	35,812	95.20	6.18	11.62
Case 1: Step 4	16,899	1962	19,431	89.60	46.52	61.24
Case 1: Step 1 + 2	33,771	281	4240	99.17	88.85	93.73
Case 1: Step 1 + 3	26,394	145	11,753	99.45	69.19	81.61
Case 1: Step 2 + 3	19,259	656	18,377	96.71	51.17	66.93
Case 1: Step 1 + 4	33,772	999	3521	97.13	90.56	93.73
Case 1: Step 2 + 4	16,899	1962	19,431	89.60	46.52	61.24
Case 1: Step 1 + 2 + 3	36,132	400	1760	98.91	95.36	97.10
Case 1: Step 1 + 2 + 4	33,772	999	3521	97.13	90.56	93.73
Case 1: Step 2 + 3 + 4	19,260	1811	17,221	91.41	52.79	66.93
Case 1: whole process	36,132	551	1609	97.71	96.50	97.10
Case 2: whole process	2,250,402	710,234	0	76.01	100.00	86.37

7.1.1 Four-Step Inference Process

Test 1: In Table 6, for *Step 1*, 24,059 datatypes were inferred (62.83% of the total, 38,292) with a Precision, Recall, and *F*-score of 99.89%, 62.81%, and 77.12%, respectively. This process inferred 26 invalid simple datatypes due to inconsistencies on the data. In *Step 2*, 17,435 datatypes were inferred (45.53% of the total) with a Precision, Recall, and *F*-score of 96.91%, 44.76%, and 61.24%, respectively. This process inferred 537 invalid datatypes (14 simple and 523 complex datatypes), but could not determine the datatype for 20,857 literal objects. Combining *Step 1* and *Step 2*, the Precision, Recall, and *F*-score values increased considerably (99.17%, 88.85%, and 93.73%, respectively). In *Step 3*, only 2480 datatypes were inferred (Recall 6.18%), since it is proposed for particular cases (context rules). Precision in *Step 4* is less than all other steps; however, the Recall is greater than *Step 2* and it makes a *F*-score similar to *Step 2*. Other combinations as *Step 1* and *Step 3* and *Step 2* and *Step 3* have high Precision but low Recall, because of the Recall of *Step 3* (specific cases). We noted that the combination of *Step 2* and *Step 4* has the same Precision and Recall as the ones of *Step 4*. According to the definition of *Step 4*, it uses the datatype candidates in order to keep the most general datatypes. The candidates are obtained by a lexical-space-matching process, which is the *Step 2*. The same situation is noted between the results of *Step 1*, *Step 4*, and *Step 1*, *Step 2*, *Step 4*. The results of the combinations of three steps (*Step 1* + *Step 2* + *Step 3*, *Step 1* + *Step 2* + *Step 4*, and *Step 2* + *Step 3* + *Step 4*) show that *Step 1* plays an important role

during the inference process due to the lowest *F*-score value obtained when this step is not considered.

Executing the whole process, 37,066 datatypes were inferred (96.80%). The Precision, Recall, and *F*-score were 97.71%, 96.50%, and 97.10%, respectively.

The best *F*-score was obtained with the whole inference process; however, the Precision decreased from 99.89% (*Step 1*) to 97.71% because of *Step 3* and *Step 4* (Precision 95.20% and 89.60%, respectively). Table 7 shows the Precision, Recall, and *F*-score for each datatype available in Case 1. In this table, the datatype `date` was not correctly inferred 7 times; however, according to the W3C Recommendation, its lexical space representation is unique and the datatype can be inferred by a simple lexical space matching; regarding the data, these 7 cases have the format YY-MM-DD instead of CCYY-MM-DD, which is the cause of the incorrect inferences (inconsistencies of the data).

In Case 2, the Precision decreased to 76.01% due to the noise and inconsistencies of the DBpedia datasets [27] (e.g., `dbo:deathDate` should have the datatype property `date`, but in the queried datasets, it was set as `gYear`).

Test 2: We also evaluated the accuracy of our four-step process in comparison with alternative methods and tools, namely Xstruct [14], XMLgrid [32], FreeFormatted [12], and XMLMicrosoft [21]. Since these works infer datatypes in XML documents, we transformed all literal nodes to XML format by using the value and its relation. Table 8 shows the accuracy results obtained for Case 1. Note that our process has the best Precision and *F*-score. Our Recall is less than the other ones because we consider a bigger

Table 7 A detailed inference per datatype (Case 1)—whole four-step process

Datatype	Valid	Invalid	Ambiguous	Precision (%)	Recall (%)	Case 1: <i>F</i> -score (%)
integer	13,567	424	1311	96.37	91.72	93.99
gYear	5067	1	0	99.98	100	99.99
date	16,446	7	0	99.91	100	99.98
gMonthDay	459	0	0	100	100	100
float	0	142	0	0	NaN	NaN
double	266	1	0	100	99.63	99.81
nonNegativeInteger	77	0	0	100	100	100
decimal	0	0	1	NaN	0	NaN
Complex	250	273	0	47.80	100	64.68
Total	36,132	934	1226	97.71	96.50	97.10

Table 8 Accuracy comparison of the four-step inference process with the related work (Case 1)

Work	Precision (%)	Recall (%)	<i>F</i> -score (%)
Xstruct	83.28	100	90.88
XMLgrid	83.61	100	91.07
FreeFormatted	43.32	100	60.45
XMLMicrosoft	43.23	100	60.36
Four-step process	97.71	96.50	97.10

number of datatypes, and thus, there are more ambiguous cases (*lexical space* intersections).

Test 3: For Case 1, we performed an extra experiment to measure the behavior of our four-step inference process when a partial number of datatypes are missed (25%, 50%,

and 75%). Table 9 shows the results obtained for this experiment. Precision, Recall, and *F*-score were measured with respect to the number of missed datatypes. Since each document has at most two same predicates, the results have not increased significantly. However, when a huge number of the same predicates are presented, the known datatype of a literal node is added to all the literal nodes associated with its predicate, leading to a better and easy inference.

7.1.2 Non-ambiguous Lexical-Space-Matching Process

Test 4: In Table 10, almost all simple datatypes were inferred by a high Precision and Recall (100.00% in both cases) for Case 1. However, due to the inconsistency of the data, the datatype *date* was considered as *string* in 7 cases, where the lexical space representation did not match

Table 9 Availability of datatypes for the four-step inference process (Case 1)

Availability of datatypes (%)	Precision (%)	Recall (%)	<i>F</i> -score (%)
0	97.71	96.50	97.10
25	97.78	96.47	97.12
50	97.66	96.66	97.16
75	97.64	96.91	97.27

Table 10 A detailed inference per datatype (Case 1)—non-ambiguous LS-matching process

Datatype	Valid	Invalid	Ambiguous	Precision (%)	Recall (%)	Case 1: <i>F</i> -score (%)
integer	15,302	0	0	100.00	100.00	100.00
gYear	5068	0	0	100.00	100.00	100.00
date	16,446	7	0	99.91	100.00	99.98
gMonthDay	459	0	0	100	100	100
float	142	0	0	100.00	100.00	100.00
double	267	0	0	100	99.63	99.81
nonNegative Integer	77	0	0	100.00	100.00	100
decimal	1	0	0	100.00	100.00	100.00
Complex	0	0	523	47.80	100	64.68
Total	36,132	934	1226	99.98	98.98	99.30

Table 11 Performance evaluation for four-step inference process

Four-step inference process	Performance evaluation	
	Execution time (s)	Cache building time (s)
Case 1: Step 1	31.336	11.582
Case 1: Step 2	15.939	15.939
Case 1: Step 3	243.826	40.764
Case 1: Step 4	17.879	17.879
Case 1: Step 1 + Step 2	33.216	13.966
Case 1: whole approach	53.247	14.236
Case 2: whole approach	–	59.282

with the current W3C lexical spaces. No complex datatypes that are also present in Case 1 were inferred. The total Precision, Recall, and *F*-score values are 99.98%, 98.98%, and 99.30%, respectively. Comparing the obtained values with the four-step process, we can observe a better accuracy, i.e., 97.27% for four-step inference process and 99.98% for non-ambiguous lexical-space-matching process.

7.2 Performance Evaluation

To evaluate the performance of our inference processes, we measured the average time of 10 executions for each test.

7.2.1 Four-Step Inference Process

Test 5: Table 11 shows the results obtained in our four-step inference process performance evaluation. In Case 1, the execution time of *Step 1* was greater than that of *Step 2*, because the use of external calls increased the execution time. However, the execution time of *Step 1* + *Step 2* was similar to *Step 1*, since *Step 1* works as a filter of triples and leaves less analysis for *Step 2*. *Step 3* has the greatest execution time, since it depends on an external service. *Step 4* depends on the list of candidate datatypes; thus, its execution time is greater than that of *Step 2* due to the use of extra operations to reduce the set of datatypes (generalization).

Test 6: Additionally, we implemented in *Step 1* and *Step 3* the use of cache to store predicate information and

predicate contexts, respectively (see Table 11—column 3). This cache is reused for consequential analysis of triples, since the same predicates are available in different triples. In Case 1, the use of cache in *Step 1* reduced the execution time for more than 65% and made the execution time of *Step 1* + *Step 2* less than those of *Step 1* and *Step 2*, separately. The cache in the whole inference approach represented more than 70% of improvement in the performance and an average of 157×10^{-7} s per triple. Moreover, for more than 16 millions of triples (Case 2), the execution time remained in the order of seconds (59.28 s) and the average execution time per triple was reduced to 35×10^{-7} s. We presume that in Case 2 the majority of triples were inferred by *Step 1*, which uses cache.

Figure 7 shows the execution time with respect to the number of triples. The performance obtained confirms the linearity of our inference approach.

Note that the use of cache makes the function stable for high number of triples because of the finite number of predicates available in the DBpedia database.

7.2.2 Non-ambiguous Lexical-Space-Matching Process

Test 7: As a lexical space matching is performed during the parser of the RDF data, we compared the parsing time of the original Jena framework with respect to the one from the modified version. Table 12 shows the execution times for *Case 1*. We observed a minimum increment of the original Jena source with respect to the modified one (11.192 s and 11.955 s, respectively).

Fig. 7 Execution time of the four-step inference process

Fig. 8 Execution time of the non-ambiguous lexical-space-matching process

Table 12 Performance evaluation for non-ambiguous lexical space process

Non-ambiguous LS matching	Performance evaluation	
	Jena sources (s)	Modified Jena sources (s)
Case 1	11.192	11.955

Table 13 Accuracy and performance evaluations for our inference processes

Inference process	Accuracy evaluation			Performance evaluation (s)
	Precision (%)	Recall (%)	F-score (%)	
Four-step [10]	97.71	96.50	97.10	14.236 (cache building)
Non-ambiguous LS matching	99.98	98.98	99.30	11.955

Table 14 Requirements of the study

Inference process	Method	Requirements				
		Simple datatypes	Data criteria		Suitability	
			Local	External	XML/XSD	RDF-OWL
Four-step [10]	IRI information Lexical space Semantic analysis Generalization	Only primitive	✓	✓	X	✓
Non-ambiguous LS matching	Lexical space	Primitive and derived	✓	X	✓	✓

Test 8: In Case 2, we measured the parsing time with respect to the number of triples. Figure 8 confirms the linearity of our non-ambiguous lexical-space-matching process. This test demonstrates that the modification in Jena source has an insignificant impact on the total performance of the parsing.

7.3 Discussion and Comparison

According to our experiments for the Semantic Web, the four-step inference process overcomes existing inference tools, in terms of Precision and Recall. We obtained up to 97.10% of *F*-score value. We suggest the use of the four steps in a particular order starting from a general solution (*Step 1*), that can be applied to all datatypes, until a specific one for particular cases (*Step 4*). Following this steps-order, we obtained the best results during experimentation.

Since *Step 2* showed a high accuracy and performance during experimentation, we worked on the lexical spaces in order to improve the results. The non-ambiguous lexical-space-matching process is behaving better than the four-step inference process in accuracy (99.30% of *F*-score) and performance (11.955 s), but it demands the modification of engines that manage RDF data as triples, as well as the modification of the RDF data itself to support the new lexical space representations. Table 13 summarizes the results obtained by the accuracy and performance evaluations. Note that our second inference process overcomes the one proposed in the work [10] for both evaluations.

In Sect. 3, we have identified a set of criteria of comparison to evaluate the existing works according to the boundaries of this study. Table 14 shows the criteria satisfied by our inference processes. External information is not needed for our non-ambiguous lexical-space-matching

process, since the value representation is only used (local data). This process can be applied to any context, such as XML/XSD and databases, where datatypes are considered. However, the modification of the data and its respective parser is needed.

We modified Jena, one of the most used processing RDFs. This modification allows compatibility among lexical spaces, by accepting literal values that follow the new lexical space representations. For example, the literal values 3.1416 and f3.1416 are equivalent since both formats match with the lexical space representations of `float` and the values themselves are equal. With this process, we demonstrate the feasibility of an appropriate approach, when having non-ambiguous lexical spaces. We proposed simple lexical space modifications, but more sophisticated proposals need to be devised.

8 Conclusions

In this paper, we investigated the issue of datatype inference for RDF documents matching/integration. We proposed a RDF Datatype inFerring Framework based on two independent processes: 1) four-step inference, consisting of (i) the analysis of the predicate information associated with the object value, (ii) analysis of the lexical space of the value itself, (iii) semantic analysis of the predicate name, (iv) and generalization of datatypes; and 2) non-ambiguous lexical space matching, where literal values can be used to infer datatypes through a matching process, following a new lexical space representations. We evaluated the accuracy and performance of our inference process with DBpedia datasets (DBpedia person data). Results show that the inference approach increases the *F*-score up to 97.10% by our four-step process, where no modification of the RDF data is required, and the *F*-score up to 99.30% for our non-ambiguous lexical space process for data aligned to new lexical space representations. We modified the Jena engine to support the new lexical spaces and to provide compatibility among existing tools.

We are currently working on extending this work to include complex datatypes. We also plan to evaluate our approach with other databases of Semantic Web initiatives.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Algergawy A, et al (2008) A sequence-based ontology matching approach. In: Proceedings of European conference on artificial intelligence workshops, pp 26–30
2. Algergawy A, Nayak R, Saake G (2009) On the move to meaningful internet systems. In: Proceedings of OTM 2009: confederated international conferences, CoopIS, DOA, IS, and ODBASE 2009, Vilamoura, Portugal, November 1–6, 2009, part II, chapter XML schema element similarity measures: a schema matching context, pp 1246–1253. Springer, Berlin
3. Algergawy A, Nayak R, Saake G (2009) XML schema element similarity measures: a schema matching context, Berlin, Heidelberg, pp 1246–1253
4. Arts T, Castro LM, Hughes J (2008) Testing Erlang data types with quivq quickcheck. In: Proceedings of the 7th ACM SIGPLAN workshop on ERLANG, ACM, pp 1–8
5. Boulytchev D (2015) Combinators and type-driven transformers in objective caml. *Sci Comput Program* 114:57–73
6. Chidlovskii B (2001) Schema extraction from xml: a grammatical inference approach. In: *KRDB*, vol 45
7. Chidlovskii B (2002) Schema extraction from xml collections. In: Proceedings of the 2Nd ACM/IEEE-CS joint conference on digital libraries, JCDL '02. ACM, New York, pp 291–292
8. Dan Brickley RG RDF Schema 1.1. <https://www.w3.org/TR/rdf-schema/>. Online; Accessed 6 Dec 2016
9. Dongo I, Al Khalil F, Chbeir R, Cardinale Y (2017) Semantic web datatype similarity: towards better RDF document matching. Springer, Cham, pp 189–205
10. Dongo I, Cardinale Y, Al-Khalil F, Chbeir R (2017) Semantic web datatype inference: towards better RDF matching. Springer, Cham, pp 57–74
11. Fluet M, Pucella R (2006) Practical datatype specializations with phantom types and recursion schemes. *Electron Notes Theor Comput Sci* 148(2):211–237
12. Free Formatter—Free Online Tools For Developers (2011) <https://www.freeformatter.com/xsd-generator.html>. Online; Accessed 3 May 2017
13. Gunaratna K, Thirunarayan K, Sheth A, Cheng G (2016) Gleaning types for literals in RDF triples with application to entity summarization. In: Proceedings of the 13th international conference on the SW, pp 85–100
14. Hegewald J, Naumann F, Weis M (2006) Xstruct: efficient schema extraction from multiple and large xml documents. In: Proceedings of the 22nd international conference on data engineering workshops, Washington, DC, p 81
15. Hogan A, Harth A, Passant A, Decker S, Polleres A (2010) Weaving the pedantic web. *LDOW*, p 628
16. Holdermans S (2013) Random testing of purely functional abstract datatypes: guidelines for dealing with operation invariance. In: Proceedings of the 15th symposium on principles and practice of declarative programming, ACM, pp 275–284
17. Jeremy JZP, Carroll J (2006) XML schema datatypes in RDF and OWL, W3C working group note 14 March 2006. <https://www.w3.org/TR/swbp-xsch-datatypes/#sec-values>. Online; Accessed 6 Dec 2016
18. Jiang S, Lowd D, Dou D (2015) Ontology matching with knowledge rules. *CoRR*, abs/1507.03097
19. Kellou-Menouer K, Kedad Z (2015) Discovering types in RDF datasets. In: European semantic web conference. Springer, pp 77–81
20. Liu B, Huang K, Li J, Zhou M (2015) An incremental and distributed inference method for large-scale ontologies based on mapreduce paradigm. *Trans. Cybern* 45(1):53–64

21. Microsoft. Xml Schema Inference—Developer Network. <https://msdn.microsoft.com/en-us/library/system.xml.schema.xmlschemainference.aspx>. Online; Accessed 3 May 2017
22. Mukkala L, Arvo J, Lehtonen T, Knuutila T et al (2015) Current state of ontology matching: a survey of ontology and schema matching. University of Turku, Technical Reports No. 4
23. Ngo D, Bellahsene Z (2016) Overview of YAM++(not) yet another matcher for ontology alignment task. *Web Sem Sci Serv Agents WWW* 41:30–49
24. Patrick P, Patel F-S, Hayes J (2014) RDF 1.1 Semantics, W3C Recommendation 25 February 2014. <https://www.w3.org/TR/rdf11-mt/#literals-and-datatypes>. Online; Accessed 6 Dec 2016
25. Paul AM, Biron V (2004) XML schema part 2: datatypes, 2nd edn, W3C recommendation 28 October 2004. <https://www.w3.org/TR/xmlschema-2/#built-in-datatypes>. Online; Accessed 6 Dec 2016
26. Paulheim H, Bizer C (2013) Type inference on noisy RDF data. In: *International semantic web conference*. Springer, pp 510–525
27. Polleres A, Hogan A, Harth A, Decker S (2010) Can we ever catch up with the web? *Semant Web* 1(1, 2):45–52
28. Sleeman J, Finin T, Joshi A (2015) Entity type recognition for heterogeneous semantic graphs. *AI Mag* 36(1):75–86
29. Thuy PT, Lee Y-K, Lee S (2013) Semantic and structural similarities between xml schemas for integration of ubiquitous healthcare data. *Pers Ubiquitous Comput*. 17(7):1331–1339
30. Ticona-Herrera R, Tekli J, Chbeir R, Laborie S, Dongo I, Guzman R (2015) Toward RDF normalization. Springer, Cham, pp 261–275
31. Wang M, Gibbons J, Matsuda K, Hu Z (2013) Refactoring pattern matching. *Sci Comput Program* 78(11):2216–2242
32. XML Grid—Online XML Editor (2010) <http://xmlgrid.net/xml2xsd.html>. Online; Accessed 3 May 2017