Semantic Web Datatype Inference: Towards Better RDF Matching

Irvin Dongo, Yudith Cardinale, Firas Al-Khalil, Richard Chbeir

Semantic Web Datatype Inference: Towards Better RDF Matching

Irvin Dongo[1], Yudith Cardinale[2] Firas Al-Khalil[3], and Richard Chbeir[1]

[1] Univ Pau & Pays Adour, LIUPPA, EA3000, 64600, Anglet, France
{irvin.dongo, richard.chbeir}@univ-pau.fr
[2] Dpto. de Computación y Tecnología de la Información, U. Simón Bolívar, Venezuela
ycardinale@usb.ve
[3] University College Cork, CRCTC, 13 South Mall, Cork, Ireland
firas.alkhalil@ucc.ie

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 propose a datatype inference process based on four steps: (i) predicate information analysis (i.e., deduce the datatype from existing range property); (ii) analysis of the object value itself by a pattern-matching process (i.e., recognize the object lexical-space); (iii) semantic analysis of the predicate name and its context; and (iv) generalization of numeric and binary datatypes to ensure the integration. We evaluated the performance and the accuracy of our approach with datasets from DBpedia. Results show that the execution time of the inference process is linear and its accuracy can increase up to 97.10%.

Keywords: Datatype Analysis, Datatype Inference, XML, RDF, Semantic Web

1 Introduction

The Semantic Web enables the integration and combination of data from different sources by providing standard models such as RDF, OWL, etc. [22]. 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 (e.g., Linked Open Data integration, ontology matching) [17,4,3].

RDF describes concepts as triples, ⟨`subject`, `predicate`, `object`⟩, where subject, predicate, and object are resources identified by their IRIs. Objects can also be literals (e.g., a number, a string), which can be annotated with optional type information, called datatype. A **datatype** is a classification of data, which defines types for RDF, adopted from XML Schema [19]. 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. A W3C Recommendation [18] points out the importance of the existence of simple datatype annotations to detect entailments between objects that have identical datatype but a value represented in different formats. Moreover, it has been proven that the presence of datatype information, constraints, and annotations on an object improves the similarity measures between two documents (up to 14%) [4].

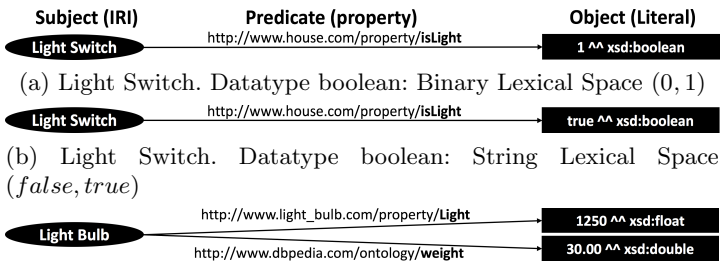
When this information is missing, datatype inference emerges as a new challenge in order to obtain more accurate RDF document matching results. In the context of XSD, works such as [7,11] infer simple datatypes by a pattern-matching process on the format of the values; i.e., the characters that make unique a datatype, which is called *lexical space* according to the W3C Recommendation [19]. These works consider a limited number of simple datatypes (`date`, `decimal`, `integer`, `boolean`, and `string`), thus for other datatypes, as `year` (e.g., `1999`), this method cannot determine its correct datatype. Others works in the context of programming languages and OWL are focused on inferring complex datatype through axioms, assigned operations, and inference rules [9,12,20], which are elements not present in an RDF document for simple datatypes. Thus, in the context of RDF document matching/integration, these works are not suitable mainly for two reasons: (i) lexical space based methods cannot infer all simple datatypes, since 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); and (ii) complex datatype inference methods cannot be applied to simple datatypes, since in RDF, a simple datatype is an atomic value associated to a predicate.

To overcome these limitations, we propose a new approach that considers, in addition to the lexical space analysis, the analysis of the predicate information related to the object. 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. Moreover, we experimentally evaluated the accuracy and performance of our approach by using DBpedia databases. Results show a high accuracy (F-score up to 97.10%) and a linear execution time ($O(n)$).

The rest of the paper is organized as follows: Section 2 presents a motivating scenario. Section 3 surveys the related literature. RDF terminologies are presented in Section 4. Section 5 describes our inference approach. Section 6 shows the experiments. We conclude in Section 7.

2 Motivating Scenario

In order to illustrate the importance of datatype information for RDF documents, we consider a scenario in which we need to integrate three RDF doc-



(a) Light Switch. Datatype boolean: Binary Lexical Space $(0, 1)$

(b) Light Switch. Datatype boolean: String Lexical Space $(false, true)$

(c) Light Bulb

Fig. 1. Three concepts from three different RDF documents

uments with similar concepts (resources) but based on different vocabularies. Fig. 1 shows three concepts from three different RDF documents. Figs. 1a and 1b describe the concept `Light Switch`, with property (predicate) `isLight`, whose datatype is `boolean`. However, they are represented with different lexical spaces: binary lexical space with value `1` in Fig. 1a and string lexical space with value `true` in Fig. 1b. In both cases, `isLight` property expresses the state of the light switch (i.e., turned on or turned off). Fig. 1c shows the concept `Light Bulb`, with property `Efficiency`, whose datatype is `integer`, and property `Light` representing the luminosity of the light (luminous flux), whose datatype is `float`.

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

1. Both `Light Switch` concepts from Figs. 1a and 1b 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 string values (true or false) according to the W3C [19].
2. `Light Bulb` concept is different from the other ones. Indeed, the `Light` property is expressed with `float` values, expressing the light intensity instead of light switch state (i.e., turned on or turned off).

If the datatype information is missing and the integration is made only based on literals, we have problems related to the *ambiguity* of properties. Contrary to our intuition, concepts in Figs. 1a and 1b are incompatible because of the use of different lexical spaces (i.e., value `1` is not compatible with value `true`, which can be considered as a `string` datatype instead of `boolean`). 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 documents are compatible because the lexical spaces of their values are the same (1 and 1275 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.

In this scenario, we can realize the role of datatypes for matching/integration of RDF documents. Thus, when this information is missing, an approach capable of inferring the datatype from the existing information is needed.

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 the

context of XSD, programming languages, and OWL (theoretical approaches and tools). To evaluate the existing works, we have identified the following criteria of comparison: (i) consideration of *simple* or *complex* datatypes; (ii) analysis of *local information*, such the object value, and *external information*, since the Semantic Web allows the integration of resources available on the Web; and (iii) *suitability for the Semantic Web*, the whole method should be objective and complete.

For **theoretical approaches**, we classify the existing works into four groups:

- Lexical space based approaches: In the inference of XSD from XML documents, datatypes are reduced to a small set of values (`date`, `decimal`, `integer`, `boolean`, and `string`) or to only `string` datatypes. The authors in [7,11] propose a hierarchy between the reduced datatypes according to the *lexical spaces* of the W3C Recommendation. The proposal returns the most specific datatype that subsumes the candidate datatypes obtained from the patter-matching of the values. However, a `gYear` value is reduced to `interger`, which is incorrect.

- Axioms, constructors, and operations based approaches: In the context of programming languages, the authors in [9] focus on inferring complex datatypes, modelling them as a collection of constructor, destructor, and coercion functions. Other works [12,24], also use axioms and pattern matching over the constructors of the datatype during the inference process. In [5,6], operations and a syntax associated to 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, some simple datatypes have intersection among their *lexical spaces* as `gYear` and `integer`.

- Inference rules based approaches: In the context of OWL, the authors in [20] propose a method to heuristically generate type information by exploiting axioms in a knowledge base. They assign type probabilities to the assertions. In the domain of health-care, [23] proposes a type recognition approach (inference type) by associating a weight to each predicate, using support vector machines to model types and by building a dictionary to map instances. For [15], the Semantic Web needs an incremental and distributed inference method because of the long ontology size. The authors use a parallel and distributed process (MapReduce) to "reduce" the "map" of new inference rules. The authors in [14] state that DBpedia only provide 63.7% of type 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. The use of inference rules helps to infer datatypes where a specific information is known (e.g., type of properties, knowledge database). However, RDF data is not always available with its respective ontology, which makes impossible the task of formulating inference rules.

- Semantic analysis based approaches: In [10], the authors analyze two types of predicates: object property and datatype property. They propose an approach to infer the semantic type of string literals using the word detection technique called Stanford CoreNLPto identify the principal term. However, a semantic type is not always related to the same datatype, since it depends on

Table 1. Related Work Classification

Work	Inference Method	Requirements				
			Information		Semantic Web	
		Simple Datatypes	Local	External	XML/XSD	RDF/OWL
[7,11]	Lexical Space	Reduced Set	✓	X	✓	X
[9,12,24,5,6]	Axioms, operations, constructors	Only Complex	✓	X	✓	X
[20,23,14]	Inference rules	Only Complex	✓	X	X	✓
[10]	Semantic Analysis	Only `string`	✓	✓	X	✓
Tools: [1,2,16]	Not provided	Not provided	✓	X	✓	X

the datatype defined in the structure. A value can be expressed as a `string` or `integer` according to two different ontologies.

On the other hand, there are **tools** that generate XSD from XML documents, such as XMLgrid [1], FreeFormatted [2], and XmlSchemaInference by Microsoft [16]. However, they do not share a standard process to infer datatypes. For example, the attribute `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`; using FreeFormatted, the datatype for `weight` is `float` and for `isLight` is `byte`; while according XmlSchemaInference `weight` is `decimal` and `isLight` is `unsignedByte`. The criteria used to infer the datatype are unknown since these tools do not describe their algorithms. Thus, the direct application of existing approaches presents limitations in the context of RDF document integration/matching.

Table 1 shows our related work classification. Note that none of the works satisfies all the defined requirements. Before describing how our approach overcomes the limitations of existing works and addresses these requirements, the following section introduces some common terminologies and definitions of RDF.

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 literals 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 to 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`). Def. 1 presents the formal definition of datatype according to W3C [13].

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) [13].* ♦

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

- $VS(\texttt{boolean}) = \{\text{true,false}\}$;
- $LS(\texttt{boolean}) = \{"true", "false", "1", "0"\}$;
- $L2V(\texttt{boolean}) = \{"true" \Rightarrow true, "false" \Rightarrow false, "1" \Rightarrow true, "0" \Rightarrow false\}$.

Table 2 presents abbreviations to denoted several sets of RDF elements, that we use in our formal approach description.

Table 2. Description of sets

Set	Description
I	A set of IRIs is defined as: $I = \{i \mid i \text{ is an } IRI\}$.
L	A set of literal nodes is defined as: $L = \{l \mid l \text{ is a literal node}\}$.
BN	A set of blank nodes is defined as: $BN = \{bn \mid bn \text{ is a blank node}\}$.
DT	A set of Datatypes is defined as: $DT = \{dt \mid dt \text{ is a datatype}\}$.
SDT	The set of simple datatypes proposed by the W3C, is defined as: $SDT = \{\texttt{string}, \texttt{boolean}, \texttt{decimal}, \ldots\}$

Definition 2. ***Triple (t)****: 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 $t :< s, p, o >$, 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:

- t_1: `<Light Switch,house:isLight,1>`
- t_2: `<Light Switch,house:isLight,true>`
- t_3: `<Light Bulb,light:Light,125>`
- t_4: `<Light Bulb,dbp:weight,30.00>`

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

5 Inference Process: Our Proposal

Our datatype inference approach mainly relies on a four step process that 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. Fig. 2 shows the framework of our inference process composed by the four steps. Each step can be applied independently and in different orders according to user parameters.

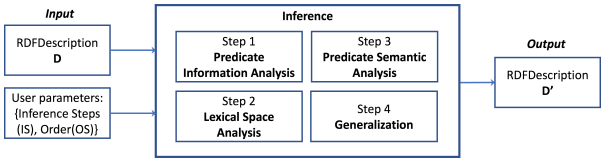


Fig. 2. Framework of our RDF Inference process

The input of our framework is an RDF Description which can be in different serializations (such as RDF/XML, Turtle, N3) and the user parameters (infer-

ence steps and their order). The output is an RDF Description with its respective inferred datatypes. A description of each step is presented as follows:

– **Step 1 – Predicate Information Analysis:** In a triple `t: <s,p,o>`, 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 to each predicate to determine the type of subject and object. We propose as a first step to deduce the simple datatype of a particular literal object, based on the use of the property `rdfs:range`, if this information exists. We formally describe the Step 1 with the following definitions and rule.

Definition 3. ***Predicate Information (*PI*)****:* *Given a triple* $t :< s, p, o >$*, Predicate Information is a function, denoted as* ***PI(t)****, that returns a set of triples defined as:* $PI(t) = \{t_i \mid t_i =< s_i, p_i, o_i >\}$*, where:*
– s_i *is the predicate of t (t.p), acting as a subject on each* t_i *triple;*
– p_i *is an RDF defined property* $\in \{rdfs{:}type, rdfs{:}label, rdfs{:}range, ...\}$*;*
– o_i *is the value of* p_i*.* ♦

Table 3 shows the set of triples (PI), returned by the function Predicate Information, for property `dbp:weight`, which is presented in Fig. 1c.

Table 3. Example of the set of triples of Predicate information (PI) for `dbp:weight`

Subject	Predicate (Property)	Object (Value)
dbp:weight	rdf:type	owl:DatatypeProperty
dbp:weight	rdfs:label	weight (*g*) (en)
dbp:weight	**rdfs:range**	**xsd:double**
dbp:weight	prov:wasDerivedFrom	http://mappings.dbpedia.org/OntologyProperty:weight

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

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

Applying Def. 4 to the set of predicate information (PI) of property `dbp-:weight` (see Table 3), the Predicate Range Information function returns the value `xsd:double`.

Definition 5. Is Available (*IA*)*:* *Given a predicate* ***p****, Is Available is a boolean function, denoted as* ***IA(p)****, that verifies if* ***p*** *is an IRI available on the web:*

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

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

Rule 1. Datatype Inference by Predicate Information Analysis*:*
Given a triple $t :< s, p, o >$*, in which* $o \in L$*, the datatype of o is determined as follows:* ***R1:*** $if\ IA(p) \implies datatype = PRI(t).$

Rule 1 verifies if the predicate of the triple is an IRI available (Def. 5), extracts the set of triples corresponding to the predicate information (Def. 3), and determines if the `rdfs:range` property exists (Def. 4).

Table 4. Lexical Space for several Simple Datatypes (W3C Recommendation [19])

Datatype	Lexical Space	Examples
date	CCYY-MM-DD	1999-05-31
gYear	CCYY	1999
boolean	true, false, 1, 0	false
float	32-bit floating point type	12.78e-2, 1999
decimal	Arbitrary precision	12.78e-2, 1999
integer	[0-9]	1999

– **Step 2** – **Datatype Lexical Space Analysis:** According to a W3C Recommendation, the *lexical space* of a datatype describes the representation format and restricts the use of characters for the object values. Table 4 shows the *lexical spaces* of several simple datatypes according to the W3C. 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 several cases (such as `boolean`, `gYear`, `decimal`, `double`, `float`, `integer`, `base64Binary`, and `hexBinary`), the *lexical spaces* of datatypes have common characteristics, leading to ambiguity (e.g., value `1999` matches with *lexical spaces* of `gYear` and `float` – see Table 4).

Figure 3 illustrates graphically the *lexical space* intersections of W3C simple datatypes. To analyze the *lexical spaces*, we propose the following definition.

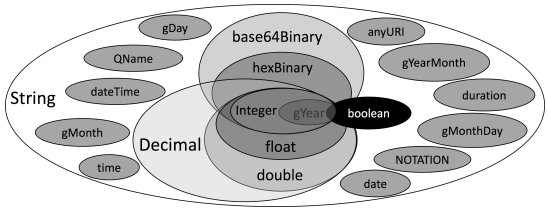


Fig. 3. Datatype Lexical Space Intersection

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:* $CDT(o) = \{dt \mid dt \in SDT \wedge o \in LS(dt)\}$ ♦

By Def. 6, the set of candidate datatypes of the object literal value `1` presented in Fig. 1a is: `CDT(1)={float, decimal, double, hexBinary, base64Binary, integer, boolean, string}`. Based on this definition, we formally define our second inference rule.

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

R2: $$datatype = \begin{cases} string & \text{if } |CDT(o)| = 1, \\ dt \mid dt \in CDT(o) \wedge dt \neq string & \text{if } |CDT(o)| = 2, \\ null & \text{otherwise.} \end{cases}$$

Rule 2 analyzes the number of possible datatypes of a literal object value. In all cases, the datatype `string` is a candidate datatype, since it has the most general *lexical space* (see Fig. 3); 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 cannot provide any decision.

– **Step 3 – Predicate Semantic Analysis:** In presence of ambiguous cases, a semantic analysis of the predicate can be done to resolve ambiguity. Regarding the W3C datatype *lexical spaces*, the datatypes `boolean`, `gYear`, `decimal`, `double`, `float`, `integer`, `base64Binary`, and `hexBinary` datatypes 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*, 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*): *Knowledge bases (thesaurus, taxonomies, and ontologies) provide a framework to organize entities (words/expressions, generic concepts, etc.) into a semantic space. A knowledge base has the following defined functions:*

– **Similarity (*sim*)**: *Given two word values* $\boldsymbol{n}$ *and* $\boldsymbol{m}$*, Similarity is a function, denoted as* ***sim(n,m)****, that returns the similarity value among the words:*
$sim(n, m) = A\ similarity\ value \in [0, 1]\ between\ \boldsymbol{n}\ and\ \boldsymbol{m}\ according\ to\ KB.$

– **IsPlural (*IP*)**: *Given a string value* $\boldsymbol{n}$*, IsPlural is a function, denoted as* ***IP(n)****, that returns True if the word* $\boldsymbol{n}$ *is plural:*

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

– **IsCondition (*IC*)**: *Given a string value* $\boldsymbol{n}$*, IsCondition is a function, denoted as* ***IC(n)****, that returns True if the word* $\boldsymbol{n}$ *is a condition:*

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

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

Definition 8. Context (*ct*): *A context is a related word or synonym, which clarifies or generalizes the domain of a word. It is associated to a similarity value according to a knowledge base. A context is denoted as a 3-tuple* $ct :< w, y, v >$*, where* w *is a word;* y *is a related word of* w*; and* v *is sim(w,y)* $\in [0, 1]$. ♦

Definition 9. Set of Contexts (*CT*): *Given a word* w*, a set of contexts of* w *is defined as* $CT = \{ct_i \mid ct_i :< w, y_i, v_i >\ is\ a\ context\ of\ w\}$. ♦

For example, from Fig. 1c, the set of contexts of predicate `weight` is: `CT = {<weight,load,0.8>, <weight,heaviness,0.5>, ...}`

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

$$PC(t, h) = \{ct_i \mid ct_i :< p.property\_name_i, y_i, v_i >, v_i \geq h\}.$$ ♦

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

– If the context is date ($< word, \textbf{date}, 0.5 >$), the datatype is `gYear` because `gYear` (1999) is a part of datatype `date` (**1999**-05-31);

– If the context is period ($< word, \textbf{period}, 0.5 >$), the datatype is `integer` because it is about quantity.

However, if the context is date, the word from which we obtain the context, can not be plural, since plural words express quantities and it is related to datatype `integer` according to our scenarios. Based on our scenarios, the following definition is formulated:

Definition 11. Predicate Name Context (*PNC*): *Given a triple $t :< s, p, o >$, 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:*

$$PNC(t,h) = \begin{cases} \texttt{gYear} & if \exists ct_i \in PC(t,h) \mid ct_i = date \wedge \texttt{gYear} \in CDT(o); \\ \texttt{integer} & if \exists ct_i \in PC(t,h) \mid ct_i = date \wedge \texttt{integer} \in CDT(o) \\ & \wedge IP(p.property\_name); \\ \texttt{integer} & if \exists ct_i \in PC(t,h) \mid ct_i = period \wedge \texttt{integer} \in CDT(o); \\ null & \text{otherwise.} \end{cases}$$ ♦

In addition, to determine a datatype as `boolean`, we assume that a word is defined as 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 $t :< s, p, o >$, in which $o \in L$, and a threshold **h** the datatype of o is determined as follows:*

$$\textbf{R3:} \qquad datatype = \begin{cases} \texttt{boolean} & if \texttt{boolean} \in CDT(o) \wedge IC(p.name\_property); \\ PNC(t,h) & \text{otherwise.} \end{cases}$$

Rule 3 returns the datatype of the object value when a defined context associated to 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 is consistent with the W3C recommendations (e.g., self-descriptive names).

– Step 4 – Generalization of Numeric and Binary Groups: If we still have ambiguity, 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 an order among the datatypes by considering *lexical space* intersection (see Fig. 3). 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: $G(o) = \{dt \mid dt \in CDT(o) \wedge (dt$ is the most general datatype according to **Numeric** and **Binary** groups$)\}$* ♦

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 $t :< s, p, o >$*, in which* $o \in L$*, the datatype of* o *is determined as follows:*

$$\textbf{\textit{R4:}} \qquad datatype = \begin{cases} dt \mid dt \in G(o) \wedge dt \neq \texttt{string} & \text{if } |G(o)| = 2, \\ null & \text{otherwise.} \end{cases}$$

However, we can have a case where an object value has `decimal` and `base64-Binary` as candidate datatypes and our inference approach cannot determinate the most appropriate datatype.

Our inference approach allows to improve the datatype analysis for RDF matching/integration by complying with the identified requirements (see Section 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 as DBpedia.

Complexity Analysis. A complexity analysis of our inference approach indicates a linear order performance in terms of the number of triples ($O(n)$). For Step 1, the predicate information of each triple is extracted to search the `rdfs:range` property, since the number of properties associated to the predicate of each triple (Def. 3) is constant, then its execution order is $O(n)$. In the case of Step 2, for each triple a pattern-matching is executed for all simple datatypes (finite number of execution) 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 inference datatype process exhibits a linear order complexity, $O(n)$. The following section evaluates the accuracy and demonstrate the linear order performance of our proposal.

6 Experimental Evaluation

To evaluate and validate our inference approach, an online prototype system, called ***RDF2rRDF***[4], was developed using PHP and Java. For *Step 3*, we implemented our assumptions of contexts using the semantic similarity service *UMBC: Semantic Similarity Service Computing*, which is based on distributional similarity and Latent Semantic Analysis (Def. 10). UMBC service is available online and an API is provided[5]. Also, we used Wordnet[6] to recognize if a word is *plural* assuming that every word has a root lemma where the default plurality is singular. Additionally, we assume that a word is a *condition* if it has the prefix *"is"* or *"has"*. All these assumptions compose our knowledge base.

[4] http://rdf2rrdf.sigappfr.org/
[5] http://swoogle.umbc.edu/SimService/api.html
[6] WordNet is a large lexical database of English (nouns, verbs, adjectives, etc.)

Table 5. Semantic Web databases

DataBase	Datatypes
DBpedia	integer, gYear, date, gMonthDay, float, nonNegative, double, Integer and decimal
Wordnet	string
GeoLinked data	point (complex datatype)

Table 5 shows the different datatypes available in several semantic web databases. Note that DBpedia has more variety of datatypes compared with the others, thus our experiments were made with DBpedia database. Experiments were undertaken on a MacBook Pro, 2.2 GHz Intel Core(TM) i7 with 16.00GB, 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: (i) **Case 1**: 5603 RDF documents gathered from *DBpedia person data*[7], in which 1059822 triples, 38292 literal objects, and 8 different datatypes are available, and (ii) **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.

For **Case 1**, we evaluated the accuracy and performance of each step of our datatype inference approach, *Step 1 + Step 2*, *Step 1 + Step 3*, *Step 2 + Step 3*, 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*). In **Case 2,** we only evaluated the whole inference process, since it is mainly used for performance because the high number of triples.

6.1 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.

Test 1: In Table 6, for *Step 1*, 24059 datatypes were inferred (45.35% of the total, 38292) with a Precision, Recall, and F-score of 99.89%, 62.81%, and 77.12% respectively. This process inferred 26 invalid simple datatypes because inconsistencies on the data. In *Step 2*, 17435 datatypes were inferred (45.35% 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) and it could not determine the datatype for 20857 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 6.85%), 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 the Recall of *Step 3* (specific cases).

Executing the whole approach, 37066 datatypes were inferred (96.80%). The Precision, Recall and F-score are 97.71%, 96.50%, and 97.10% respectively.

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

Table 6. Accuracy Evaluation

Inference Process	Accuracy Evaluation					
	Valid	Not Valid	Ambiguity	Precision	Recall	F-score
Case 1: Step 1	24033	26	14233	99.89%	62.81%	77.12%
Case 1: Step 2	16898	537	20857	96.92%	44.76%	61.24%
Case 1: Step 3	2480	119	35812	95.20%	6.85%	11.62%
Case 1: Step 4	16899	1962	19431	89.60%	46.52%	61.24%
Case 1: Step 1 + Step 2	33771	281	4240	99.17%	88.85%	93.73%
Case 1: Step 1 + Step 3	26394	145	11753	99.45%	69.19%	81.61%
Case 1: Step 2 + Step 3	19259	656	18377	96.71%	51.17%	66.93%
Case 1: Whole Approach	36132	551	1609	97.71%	96.50%	97.10%
Case 2: Whole Approach	2250402	710234	0	76.01%	100.00%	86.37%

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). For Case 2, the Precision decreased to 76.01%. It is caused by the noise and inconsistencies of the DBpedia datasets [21] (e.g., `dbo:deathDate` should have the datatype property `date`, but in the queried datasets, it was set as `gYear`).

Table 7. Accuracy Comparison with the Related Work for Case 1

Work	Precision	Recall	F-score
Xstruct	83.28%	100%	90.88%
XMLgrid	83.61%	100%	91.07%
FreeFormated	43.32%	100%	60.45%
XMLMicrosft	43.23%	100%	60.36%
RDF2rRDF	**97.71%**	**96.50%**	**97.10%**

Test 2: We also evaluated the accuracy of our approach in comparison with alternative methods and tools, namely Xstruct, XMLgrid, FreeFormated, and XMLMicrosoft [1,2,11,16]. Since these works infer datatypes in XML documents, we transformed all literal nodes to XML format by using the value and its relation. Table 7 shows the accuracy results obtained for Case 1. Note that our approach has the best Precision and F-score. Our Recall is less than the other ones because we consider a bigger number of datatypes and thus, there are more ambiguous cases (*lexical space* intersections).

6.2 Performance evaluation

To evaluate the performance, we measured the average time of 10 executions for each test. Table 8 shows the results obtained in our performance evaluation.

Test 3: In Case 1, the execution time of *Step 1* was greater than *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 of an external service. *Step 4* depends of the list of candidate datatypes; thus its execution time should be greater than *Step 2* because the use of extra operations to reduce the set of datatypes (generalization).

Table 8. Performance Evaluation

Inference Process	Performance Evaluation	
	Execution Time	Cache Building Time
Case 1: Step 1	31.336s	11.582s
Case 1: Step 2	15.939s	15.939s
Case 1: Step 3	243.826s	40.764s
Case 1: Step 4	17.879s	17.879s
Case 1: Step 1 + Step 2	33.216s	13.966s
Case 1: Whole Approach	53.247s	14.236s
Case 2: Whole Approach	-	59.282s

Test 4: Additionally, we implemented in *Step 1* and *Step 3* the use of cache to store predicate information and predicate contexts, respectively (see Table

8 - column 3). This cache is reused for consequential analysis of triples, since same predicates are available in different triples. For Case 1, the use of cache in Step 1 reduced the execution time in more than 65% and made the execution time of *Step 1 + Step 2* less than *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}sec. per triple. Moreover, for more than 16 millions of triples (Case 2), the execution time remained in the order of seconds (59.28s) and the average execution time per tripe was reduced to 35×10^{-7}sec. We presume in Case 2 that the majority of triples were inferred in *Step 1*, which uses cache.

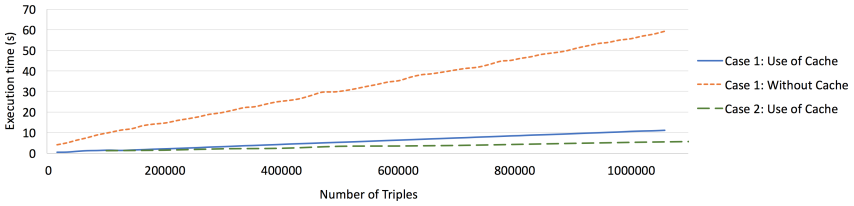


Fig. 4. Execution Time of our Inference Approach

Fig. 4 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 Conclusion

In this paper, we investigated the issue of datatype inference for RDF documents matching/integration. We proposed an approach, consisting of four steps: the analysis of the predicate information associated to the object value, analysis of the lexical space of the value itself, semantic analysis of the predicate name, and generalization of datatypes. 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% (accuracy) and it does not incur in high execution time (performance). We are currently working on extending this work to include other datatypes and propose more context rules to resolve extra ambiguity. We also plan to evaluate our approach with other databases from Semantic Web initiatives.

Acknowledgments: FINCyT/INNOVATE Peru - N 104-FINCyT-BDE-2014.

References

1. XML Grid - Online XML Editor. `http://xmlgrid.net/xml2xsd.html`, 2010. Online; accessed 2017-05-03.
2. Free Formatter - Free Online Tools For Developers. `https://www.freeformatter.com/xsd-genearator.html`, 2011. Online; accessed 2017-05-03.
3. A. Algergawy and et al. A sequence-based ontology matching approach. In *Proc. of European Conference on Artificial Intelligence Workshops*, pages 26–30, 2008.
4. A. Algergawy, R. Nayak, and G. Saake. *XML Schema Element Similarity Measures: A Schema Matching Context*, pages 1246–1253. Berlin, Heidelberg, 2009.

5. T. Arts, L. M. Castro, and J. Hughes. Testing erlang data types with quviq quickcheck. In *Proc. of the 7th ACM SIGPLAN Workshop on ERLANG*, pages 1–8, NY, USA, 2008. ACM.
6. D. Boulytchev. Combinators and type-driven transformers in objective caml. *Science of Computer Programming*, 114:57 – 73, 2015.
7. B. Chidlovskii. Schema extraction from xml collections. In *Proceedings of the 2Nd ACM/IEEE-CS Joint Conference on Digital Libraries*, JCDL '02, pages 291–292, New York, NY, USA, 2002. ACM.
8. R. G. Dan Brickley. RDF Schema 1.1. `https://www.w3.org/TR/rdf-schema/`. Online; accessed 2016-12-06.
9. M. Fluet and R. Pucella. Practical datatype specializations with phantom types and recursion schemes. *Electronic Notes in Theoretical Computer Science*, 148(2):211 – 237, 2006.
10. K. Gunaratna, K. Thirunarayan, A. Sheth, and G. Cheng. Gleaning types for literals in rdf triples with application to entity summarization. In *Proc. of the 13th International Conference on The SW.*, pages 85–100, NY, USA, 2016.
11. J. Hegewald, F. Naumann, and M. Weis. Xstruct: Efficient schema extraction from multiple and large xml documents. In *Proc. of the 22Nd International Conference on Data Engineering Workshops*, pages 81–, Washington, DC, USA, 2006.
12. S. Holdermans. Random testing of purely functional abstract datatypes: Guidelines for dealing with operation invariance. In *Proc. of the 15th Symposium on Principles and Practice of Declarative Programming*, pages 275–284. ACM, 2013.
13. J. Z. P. Jeremy J. Carroll. XML Schema Datatypes in RDF and OWL, W3C Working Group Note 14 March 2006. `https://www.w3.org/TR/swbp-xsch-datatypes/#sec-values`, 2006. Online; accessed 2016-12-06.
14. K. Kellou-Menouer and Z. Kedad. Discovering types in rdf datasets. In *European Semantic Web Conference*, pages 77–81. Springer, 2015.
15. B. Liu, K. Huang, J. Li, and M. Zhou. An incremental and distributed inference method for large-scale ontologies based on mapreduce paradigm. *Transac. on Cybernetics*, 45(1):53–64, 2015.
16. Microsoft. Xml Schema Inference - Developer Network. `https://msdn.microsoft.com/en-us/library/system.xml.schema.xmlschemainference.aspx`. Online; accessed 2017-05-03.
17. L. Mukkala, J. Arvo, T. Lehtonen, T. Knuutila, et al. Current state of ontology matching. a survey of ontology and schema matching. 2015.
18. P. F. P.-S. Patrick J. Hayes. RDF 1.1 Semantics, W3C Recommendation 25 February 2014. `https://www.w3.org/TR/rdf11-mt/#literals-and-datatypes`, 2014. Online; accessed 2016-12-06.
19. A. M. Paul V. Biron. XML Schema Part 2: Datatypes Second Edition, W3C Recommendation 28 October 2004. `https://www.w3.org/TR/xmlschema-2/#built-in-datatypes`, 2004. Online; accessed 2016-12-06.
20. H. Paulheim and C. Bizer. Type inference on noisy rdf data. In *International Semantic Web Conference*, pages 510–525. Springer, 2013.
21. A. Polleres, A. Hogan, A. Harth, and S. Decker. Can we ever catch up with the web? *Semantic Web*, 1(1, 2):45–52, 2010.
22. P. A. Sandro Hawke, Ivan Herman. W3C Semantic Web Activity. `https://www.w3c.org/2001/sw/`, 2001. Online; accessed 2016-12-06.
23. J. Sleeman, T. Finin, and A. Joshi. Entity type recognition for heterogeneous semantic graphs. *AI Magazine*, 36(1):75–86, 2015.
24. M. Wang, J. Gibbons, K. Matsuda, and Z. Hu. Refactoring pattern matching. *Science of Computer Programming*, 78(11):2216 – 2242, 2013.