# Learning Content Patterns from Linked Data

Emir Muñoz[1,2]

[1] Fujitsu Ireland Limited
[2] National University of Ireland, Galway
E-mail: Emir.Munoz@ie.fujitsu.com

**Abstract.** Linked Data (LD) datasets (e.g., DBpedia, Freebase) are used in many knowledge extraction tasks due to the high variety of domains they cover. Unfortunately, many of these datasets do not provide a description for their properties and classes, reducing the users' freedom to understand, reuse or enrich them. This work attempts to fill part of this lack by presenting an unsupervised approach to discover syntactic patterns in the properties used in LD datasets. This approach produces a content patterns database generated from the textual data (content) of properties, which describes the syntactic structures that each property have. Our analysis enables (i) a human-understanding of syntactic patterns for properties in a LD dataset, and (ii) a structural description of properties that facilitates its reuse or extension. Results over DBpedia dataset also show that our approach enables (iii) the detection of data inconsistencies, and (iv) the validation and suggestion of new values for a property. We also outline how the resulting database can be exploited in several information extraction use cases.

## 1 Introduction

Many companies and government agencies are massively publishing data on the Web as result of Open Data initiatives, from public and private sectors, that enable publicly-available data to be easily accessible by other users. Hence, the ability to extract information from those sources is becoming increasingly important in the society for driving innovation, investment and economic growth. In the process, data publishers usually reuse and extend a public ontology/vocabulary to be used when publishing data on the Web in Linked Data shape. The recent update of the Linked Open Data (LOD) Cloud diagram[1] shows that publishers still prefer to interlink their datasets mainly to DBpedia[2] and reuse its vocabulary among other popular ones. Statistics provided by Linked Open Vocabularies (LOV) [17] reported the existence of 446 vocabularies (by July 7th,

---

[1] http://data.dws.informatik.uni-mannheim.de/lodcloud/2014/ISWC-RDB/ (July 2014)
[2] http://dbpedia.org/About

2014) with 10 classes and 20 properties in average. This numbers reveal the cumbersome process for any publisher —person or application— to determine which properties or classes to use at the moment of design and publish new datasets.

In an ideal scenario, each property present in an ontology/vocabulary should have attached its specification of domain and range metadata in order to increase its reuse. But in practice, this is not a reality, vocabularies lack of such definitions or guidelines for users indicating how to reuse a vocabulary. Here, it is worth mentioning that domain and range metadata are categorized as *non-normative* by RDF Schema [3, §4]. Furthermore, even in the cases when such metadata is explicitly mentioned, the LD datasets that use them might contain lexical errors, such as the values `"'''e-book ID:"@en` and `"See text"@en` for the property `http://dbpedia.org/property/isbn`. This kind of errors are hard to detect in an automatic extraction process, such as the ones used by DBpedia. Our main goal is to discover a set of syntactic patterns in the content (value) of each property that provides content-based rules for valid values of properties, and thus reduce errors and increase the quality of LD datasets. Datatypes in Linked Data (e.g., `xsd:integer`, `xsd:gMonthDay`) allow the validation of values that RDF properties can take. For instance, we can expect that the property `dbp:dateCreated` should follows a syntactic pattern, such as `Number-Number-Number` to cover a Year-Month-Day data field. More specifically, a pattern such as `Small_Num-Small_Num-Medium_Num`. This guide us to state our *hypothesis*: In Linked Data, a given property value satisfies a fixed and small set of lexico-syntactic patterns.

The former hypothesis guide us to build a database that contains all the possible content patterns for each RDF property in a Linked Data dataset.

**Organization.** In this paper, we introduce RDF, domain, and range in Section 2. Section 3 introduces the concept of content patterns and the learning process to extract them from LD. Section 4 describes the algorithm for the database construction from DBpedia[3] and its implementation. A discussion of some identified use cases for the patterns database is presented in Section 5. Relevant related work is presented in Section 6, to finally conclude about our work in Section 7.

## 2 Background

In this section, we briefly introduce RDF model and the main issues with the structure of properties in current datasets of the LOD cloud.

### 2.1 RDF Model

The RDF data model is used in the Semantic Web to give a universal structure to the content that enables interoperability and semantics. An RDF triple can be seen as an atomic fact representing the existence of a relationship between a

---

[3] http://dbpedia.org/About

subject resource and an object resource selected from a set of RDF terms. The *RDF terms* set is the union of three pair-wise disjoint sets: **U**, the set of all *URI references*; **B**, an infinite set of *blank nodes*; and **L**, the set of all *literals*. The set of literals is further decomposed into the union of two disjoint sets: $\mathbf{L}_p$ the set of all *plain literals* and $\mathbf{L}_t$ the set of *typed literals*. The mentioned relationship can be formally represented by a tuple $(s, p, o)$ or *RDF triple* where $s \in \mathbf{U} \cup \mathbf{B}$ represents the *subject*, $p \in \mathbf{U}$ represents the *predicate* (instance of the class property), and $o \in \mathbf{U} \cup \mathbf{B} \cup \mathbf{L}$ represents the *object* of the triple.

This work focus on the set **L**, where each string value is analyzed to learn lexico-syntactic patterns [10] that exploit the structure (grammar) of the values for a fixed property. Thus, we do learn content-based rules that models the data (string values) of a property, which we refer as content patterns or simply patterns. As mentioned above, a literal $\ell \in \mathbf{L}$ can be either plain or typed. Plain literals are composed by plain strings, such as `"Hello World"`, and usually are associated to a language tag (e.g., en, es), such as `"Hello World"@en` and `"Hola Mundo"@es` for English and Spanish, respectively. Typed literals are those that next to the lexical string have a datatype, such as `"13"^^xsd:integer`, representing the number 13. These datatypes are generally defined for XML Schema that cover numerics, booleans, dates, times, and so forth. Plain literals without language tags are associated to `xsd:string` values by default. In addition, datatypes define which lexical forms are valid for a datatype. For instance, `"Hello World"^^xsd:integer` is an invalid statement whereas that `"Hello World"^^xsd:string` is valid.

### 2.2 Domain, Range and their issues

Vocabularies and schemas in Linked Data, and RDF specifically, aim to attach semantics to the user-defined classes and properties. RDF Schema (RDFS) [3] is an extension of RDF with four key terms that allow the specification of well-defined relationships between classes and properties. The four introduced elements are: `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain` and `rdfs:range` (please see [15] for details). We focus on the last two elements, namely, `rdfs:domain` and `rdfs:range` to explain how our content patterns database can be used. `rdfs:domain` is used to state that the subject of an RDF triple with property $p$ is a member of a given class $c$. Similarly, `rdfs:range` is used to state that the object of an RDF triple with property $p$ is a member of a given class $c$. This allows us to put constraints, and later validate the actual types of values that are appropriate for a given property. Thus, facilitating a checking of the dataset to discover errors, or to suggest appropriate values for a property.

*Example 1.* The following RDF triples:

```
dbr:17049_Miron dbo:epoch "May 14, 2008"@en
dbr:17049_Miron dbo:apoapsis "401288344481.673828"^^xsd:double
```

describe features of the asteroid `17049 Miron`[4]. The first triple shows the predicate epoch[5] (explained by DBpedia ontology as the "moment in time used as a reference point for some time-varying astronomical quantity"), where the *domain* and *range* of the property `epoch` are defined as: `http://dbpedia.org/ontology/Planet` class for planets, and `xsd:string` for string values, respectively[6]. While, the second triple shows the predicate apoapsis[7] which does not present any description explaining its use in this context.  □

Using LOV SPARQL endpoint[8], we can compute that ca. 70% of the properties have a defined domain and range. Also, that only 1.2% of the properties contains a not empty `dcterms:description`; and 40.9% contains a not empty `rdfs:comment` value. This shows that widely used LD datasets, such as DBpedia, lack of a minimal description about what/when/why to use a given property.

## 3   Content Patterns

The *content* of a property in an RDF triple, i.e. the data string in the object position, is considered as a sequence of characters. A lexical analysis over properties' content generates sequences of tokens, strings generated from an alphabet containing different types of characters: alphabetic, numeric, punctuation, etc. This sequence of tokens defines the structure of the content in an RDF property that is here used to identify patterns.

### 3.1   Learning Patterns from Linked Data

In the following, we present our method to generate the content patterns. In order to generate this patterns, first, we need an algorithm that allows us to learn structural information about string values. For this purpose, we do use DataProG presented by Lerman et al. [11], and designed for wrapper maintenance, wrapper verification and induction. This algorithm takes as input a set of positive examples, and using a word-level representation, or more accurately, a token-level representation, generates a set of lexico-syntactic rules that the tokens follow, from here on referenced as our *content patterns*. A token is considered as a particular instance of a concept or type. For example, the type `Number` can have 1, 25, or 40 as instances. The types of the tokens are associated to syntactic categories as depicted in Figure 1. Each category has its own semantics describing the datatype. For instance, `Number` category is divided in three sub-categories: `Small` (0 - 9), `Medium` (10 - 1000) and `Large` (larger than

---

[4] http://dbpedia.org/resource/17049_Miron

[5] http://dbpedia.org/ontology/epoch

[6] The consideration of an asteroid as member of the class planet stated by the example RDF triple is not a discussion covered in this paper but it is related to the question how a given RDF ontology represents the real world.

[7] http://dbpedia.org/ontology/apoapsis

[8] http://lov.okfn.org/endpoint/lov_aggregator

1000). Here, every string appearing in at least $k$ examples will be represented by a token type (see Example 2). Formally, let $\mathcal{K}_p^{\mathcal{T}} = \{k_1, k_2, \ldots, k_m\}$ be the set of patterns for property $p$ in a LD dataset $\mathcal{T}$, where every pattern $k_i$ is a sequence of syntactic classes w.r.t. the one in Figure 1.
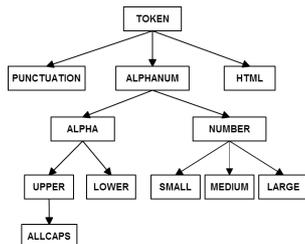


Fig. 1: Portion of the token type syntactic hierarchy [11].

*Example 2.* Given a property $p$ with value in the set $S_p = \{14.1, 362.5, 95.0\}$, the learning process should return a content pattern set: $\mathcal{K}_p^{\mathcal{T}} = \{[\texttt{Number} . \texttt{Number}]\}$. DataProG algorithm over $S_p$ returns two content patterns: $(k_1)$ [`Number Punctuation Number`], and its specification $(k_2)$ [`Medium_Number . Small_Number`], which is fine-grained. So far, $S$ satisfies both patterns $(k_1)$ and $(k_2)$. But, whenever elements in the input set follows a slightly different syntax, the set $\mathcal{K}_p^{\mathcal{T}}$ of patterns changes and becomes more general or coarse-grained. For example, adding the element $5.3i$ (a imaginary number) to the former set $S_p$, we have $S_p' = \{14.1, 362.5, 95.0, 5.3i\}$, which will change the last former syntactic category (i.e. `Number`) in the previous patterns $(k_1)$ and $(k_2)$ for `Alphanum`. □

As it is, our test data source, i.e. DBpedia, only contains "examples" of values for a given property. We do require that the learning patterns algorithm accepts as input only positive examples. DataProG algorithm satisfies this requirement. Then, it is a good choice to be used in this work for the generation of content patterns. (Note that our method is flexible enough to accept other learning pattern algorithms.)

## 4 Database Construction

In this section, we explain our methodology to build a content patterns database from a Linked Data dataset. First, we pre-process our test dataset, i.e. DBpedia, and present a survey of the properties found in it. Second, based on the learning process depicted in Section 3, we introduce our algorithm for pattern extraction in order to build the content patterns database from our Linked Data dataset.

### 4.1 DBpedia Properties Survey

We used DBpedia v3.9, gathered on May 2014, which contains over 2.4 billion RDF triples with instances of 53,230 properties. To analyze each property, the

DBpedia dump was fragmented by properties in order to extract all the patterns. For one property, the analysis can be done using the following SPARQL query:

```
SELECT ?sub ?obj
WHERE { ?sub <http://dbpedia.org/property/placeOfBirth> ?obj . }
```

where we ask for all *subject* (`?sub`) and *object* (`?obj`) elements that appear in the RDF triples where the predicate corresponds to the property `http://dbpedia.org/property/placeOfBirth`.

In this work, we do use HDT (header, dictionary, triples) [7] —a compact data structure and binary serialization format for RDF— for RDF data management. HDT allows us to search and browse DBpedia dataset using `S-P-O`-like[9] queries where we can indicate fixed values for any of these three positions. We then fragment the whole dataset, generating small and manageable indices per property. Table 1 shows the list with the top-20 most frequent properties found in DBpedia, being the `sameAs` property the most frequent. This list is mainly composed by properties coming from the most popular vocabularies: FOAF[10], Dublin Core[11], OWL[12], DBpedia.

Table 1: List of the top-20 most frequent properties in DBpedia.

| Freq. | Property |
|---|---|
| 84,724,705 | `http://www.w3.org/2002/07/owl#sameAs` |
| 61,709,033 | `http://dbpedia.org/ontology/wikiPageUsesTemplate` |
| 44,603,034 | `http://purl.org/dc/terms/subject` |
| 42,057,021 | `http://dbpedia.org/ontology/wikiPageRevisionID` |
| 42,057,021 | `http://dbpedia.org/ontology/wikiPageID` |
| 42,056,971 | `http://www.w3.org/ns/prov#wasDerivedFrom` |
| 37,784,587 | `http://www.w3.org/2000/01/rdf-schema#label` |
| 34,226,925 | `http://xmlns.com/foaf/0.1/primaryTopic` |
| 34,226,925 | `http://xmlns.com/foaf/0.1/isPrimaryTopicOf` |
| 28,440,690 | `http://www.w3.org/1999/02/22-rdf-syntax-ns#type` |
| 20,488,114 | `http://dbpedia.org/ontology/wikiPageExternalLink` |
| 18,674,346 | `http://dbpedia.org/ontology/wikiPageRedirects` |
| 12,629,958 | `http://dbpedia.org/property/hasPhotoCollection` |
| 10,408,296 | `http://dbpedia.org/property/name` |
| 8,924,014 | `http://xmlns.com/foaf/0.1/name` |
| 8,489,275 | `http://purl.org/dc/elements/1.1/rights` |
| 5,176,265 | `http://www.w3.org/2004/02/skos/core#broader` |
| 5,070,656 | `http://dbpedia.org/property/language` |
| 4,803,302 | `http://xmlns.com/foaf/0.1/depiction` |
| 4,803,300 | `http://dbpedia.org/ontology/thumbnail` |

Each index contains RDF triples from DBpedia in a `S-P-O` order optimized for fast query answering. During our analysis we take one by one the property

---

[9] The order for each RDF triple is subject - predicate - object.

[10] http://xmlns.com/foaf/spec/

[11] http://dublincore.org/documents/dcmi-terms/

[12] http://www.w3.org/TR/owl-ref/

indices, where each one contains only RDF triples for one fixed property. Each RDF triple in the index is parsed to extract the object position and determine whether it is a literal, URI or blank node to form the set $\mathbf{V}_p$ of values for property $p$. (Blank nodes do not contribute to our method—since they do not contain any URI or literal—, so they are discarded. On the other hand, for URIs we can still extract some domain patterns.) We further determine if the literals are plain or typed. From the extracted set $\mathbf{L}$ for a property index, we determine that 19.25% correspond to elements in $\mathbf{L}_p$ (plain literals), 18.02% to elements in $\mathbf{L}_t$ (typed literals), and 62.73% elements that do not contain any language or datatype associated (by default linked to `xsd:string`). Among the literals with an existing datatype, the most common datatype found was `xsd:integer`. Table 2 shows the top-10 most frequent datatypes in DBpedia that are mainly related with numbers, dates and time representation.

Table 2: List of the top-10 more frequent datatypes in DBpedia.

| Freq. | Datatype |
|---|---|
| 39,938,610 | `http://www.w3.org/2001/XMLSchema#integer` |
| 3,449,581 | `http://www.w3.org/2001/XMLSchema#double` |
| 3,268,506 | `http://www.w3.org/2001/XMLSchema#date` |
| 1,677,864 | `http://www.w3.org/2001/XMLSchema#float` |
| 1,216,821 | `http://www.w3.org/2001/XMLSchema#gYear` |
| 908,155 | `http://dbpedia.org/datatype/second` |
| 679,934 | `http://www.w3.org/2001/XMLSchema#nonNegativeInteger` |
| 275,376 | `http://www.w3.org/2001/XMLSchema#gMonthDay` |
| 204,753 | `http://dbpedia.org/datatype/squareKilometre` |
| 196,569 | `http://dbpedia.org/datatype/minute` |

### 4.2 The Algorithm

We will now design an algorithm to obtain the sets $\mathcal{K}_p^{\mathcal{T}}$ of patterns for each property $p$ in the dataset $\mathcal{T}$. For each property we extracted the set $\mathbf{V}_p$ of values that is passed as input to the learning patterns algorithm. This action generates the content patterns per property. Algorithm 1 formalizes the approach to construct the content patterns database. The first part of the algorithm (lines 3-6) represents the parsing of the LD dataset $\mathcal{T}$, and generation of the set $\mathbf{V}_p$, filtering values that are not in $\mathbf{L} \cup \mathbf{U}$. Once finished the parsing, the patterns can be generated calling DataProG (line 9); compute their corresponding coverage (line 10), and then write the 3-tuples into the database (line 11).

### 4.3 Implementation

Algorithm 1 was implemented using Java language, and tested over DBpedia dataset. In terms of implementation, due to the size of some sets $\mathbf{V}_p$, we optionally optimized the processing time and memory required for the patterns computation, by truncating the size to 500 elements maximum. This should be executed after line 7 and before line 9. In our following experiments, we applied this optimization to meet the hardware constraints imposed by the machine used.

---

**Algorithm 1** Construction of a Content Patterns Database

---

**Input:** a Linked Data dataset $\mathcal{T}$
**Output:** a content patterns database
 1: Let $\mathbf{W}$ be the set of all properties in $\mathcal{T}$
 2: Let $\mathbf{V}_p$ be set of values for a *property* $p \in \mathbf{W}$
 3: **for all** RDF triple $(s, p, o) \in \mathcal{T}$ **do**
 4:     **if** $o \in \mathbf{L} \cup \mathbf{U}$ **then**
 5:         $\mathbf{V}_p$.add($o$)
 6:     **end if**
 7: **end for**
 8: **for all** $p$ in $\mathcal{U}$ **do**
 9:     $\mathcal{K}_p^{\mathcal{T}} = \text{DataProG}(\mathbf{V}_p)$
10:     Compute coverage ($cov$) for each $k_i \in \mathcal{K}_p^{\mathcal{T}}$
11:     Write the 3-tuple $(p, k_i, cov_{k_i})$ for each property/pattern into the database
12: **end for**

---

However, this is still an optional optimization, so can be skipped or discarded for bigger hardware resources.

The machine used to process DBpedia and build the database was a virtual machine running Ubuntu Linux 12.04 with an Intel i7 processor, and 8GB of RAM memory. The source code with the implementation of the current approach used to generate a content patterns database from a Linked Data dataset can be found on-line in `https://github.com/emir-munoz/ld-patterns`. A dump of the database is available in a tabular separated values (tsv) file, which contains the generated patterns for DBpedia: `property<tab>pattern<tab>overage`, where the metric coverage is used to measure the proportion of the dataset for which the learning algorithm makes a prediction, i.e. all the examples that satisfy the pattern divided by the total number of examples. The coverage metric range is $[0.0, 1.0]$, being 1.0 the full coverage of the data examples.

## 5   Discussion

In this section, we present a discussion about the possible use cases where the generated database can be helpful. In practice, the values in the `rdfs:range` of a property are not uni-type, which does not violate the formal definition of RDF model in Section 2. However, this is translated in properties $p$ whose set $\mathbf{V}_p$ are composed by URIs mixed with literals and even blank nodes[13]. This fact makes more challenging the reuse of many properties. The survey performed on literals in DBpedia knowledge base showed that only ca. 40% falls into the categories of plain and typed literals, leaving the rest, 60%, as default literals linked to the `xsd:string` type. Nevertheless, we consider all of them in our analysis and posterior processing. When applying Algorithm 1 to DBpedia we generate the main output of this work: A database composed by ca. 500,000 content patterns associated to properties in Linked Data. With an average of 17.3 patterns per

---

[13] In LOV, 374 unique properties have a range defined as blank node.

property. In other words, we were able to find content patterns in properties values of a Linked Data dataset. This result comes to validate our initial hypothesis which stated that in Linked Data datasets, a given property satisfies a small set of lexico-syntactic patterns whenever the range of the property is not an empty value or blank node.

We report in Table 3 few examples of content patterns extracted for eight properties. Each pattern is accompanied by its coverage measure. We can notice that the patterns of some properties are easy to deduce, such as `dbp:barcode` or `dbo:address`, but it is not the same in other cases, such as `dbp:admCtrOf`, `dbp:1stishhead` or `dbp:2006MeanHhIncome`—where it is hard even to figure out the context where those properties are used. This lead us to define and discuss a set of use cases where it is handy to count with such database.

Table 3: Examples of content patterns identified.

| Property | Content Patterns | Coverage |
|---|---|---|
| `dbp:barcode` | `LARGE/FLOAT_NUMBER` | 1.0 |
| `dbo:address` | `NUMBER FIRST_UPPERCASE FIRST_UPPERCASE` | 0.318 |
| | `ALPHANUMERIC FIRST_UPPERCASE Road` | 0.056 |
| | `ALPHANUMERIC FIRST_UPPERCASE Street` | 0.044 |
| `dbp:admCtrOf` | `ALPHA of ALL_LOWERCASE significance of FIRST_UPPERCASE` | 0.298 |
| | `FIRST_UPPERCASE District` | 0.272 |
| | `Town of ALPHA significance of FIRST_UPPERCASE` | 0.206 |
| `dbo:editorTitle` | `FIRST_UPPERCASE` | 0.978 |
| | `Editor` | 0.79 |
| | `Editor - in - Chief` | 0.282 |
| `dbo:isbn` | `ALPHANUMERIC - NUMBER - NUMBER - NUMBER` | 0.56 |
| | `NUMBER - NUMBER - NUMBER - NUMBER` | 0.56 |
| | `ALPHANUMERIC 978 - SMALL_NUMBER - NUMBER - NUMBER - SMALL_NUMBER` | 0.046 |
| `dbp:1stishhead` | `vol . SMALL_NUMBER` | 0.54 |
| | `ALPHA . SMALL_NUMBER` | 0.54 |
| | `vol . SMALL_NUMBER cont .` | 0.02 |
| `dbp:2006MeanHhIncome` | `LARGE/FLOAT_NUMBER` | 0.682 |
| | `ALPHANUMERIC / A` | 0.122 |
| | `ALPHANUMERIC Available` | 0.108 |
| `dbp:dateCreated` | `NUMBER` | 0.873 |
| | `MEDIUM_NUMBER` | 0.731 |
| | `MEDIUM_NUMBER - SMALL_NUMBER - SMALL_NUMBER` | 0.233 |

To guide our discussion, we present a list with some identified use cases:

1. The database can facilitate user searches to discover and reuse existing properties. Similar to a search by example, given an example value the user can search for all the patterns that cover that example.
2. As a derivation of the previous use case, this database can facilitate a human-understanding of the lexicon of properties existing in a Linked Data dataset in general beyond the simple and not selft-explanatory `label`.
3. The database can be used to check atypical values (outliers) inside the same knowledge base, based on the most in/frequent patterns. The outliers might correspond to errors because of failures in an automatic extraction, or changes/updates in the lexicon of properties that will require to re-run the Algorithm 1 to generate a new database.
4. Most ambitiously, the database obtained can help to the automatic generation of schemas from tabular data. For example, when trying to convert CSV

to RDF format using the SPARQL-based data mapping language TARQL [5], users need to define how each column will be represented in RDF.

5. In terms of information extraction, this database can be used for instance in the table extraction problem. In a table, the columns with string values can be mapped to RDF properties, by matching the cell values with patterns in the database. This might help to improve the recall measure when performing approaches like the one described in [14].

6. Document wrappers will also benefit from this database. Extracted values by a wrapper, via CSS or XPath queries over HTML pages can be validated against the corresponding patterns.
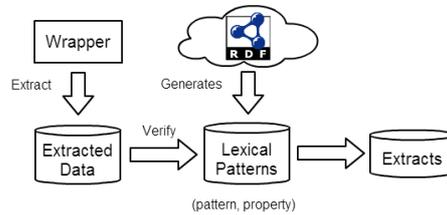


Fig. 2: Wrapper use case for the database.

A practical example is shown in Figure 2 where the lexical patterns database is generated from a Linked Data dataset, and used to validate extracted data by a wrapper. Consider as example, a wrapper that extracts data from the business card of a person embedded in his HTML page, that contains the following attributes and values:

```
E-mail: user1@domain.com
Given name: John
Surname: Snow
Web address: www.jsnow.com
```

we could use both the attributes (properties) name and its values to generate an RDF representation of this data. The first field, `E-mail` can be mapped to `dbp:email` that is further associated with the following patterns in our database:

```
ALPHANUMERIC PUNCTUATION ALL_LOWERCASE PUNCTUATION ALL_LOWERCASE
ALPHANUMERIC @ ALL_LOWERCASE . ALL_LOWERCASE
```

By using a conversion of the patterns to regular expressions (cf. [9]) we can check that this attribute value, `user1@domain.com`, match both regular expressions, and then conclude that it is valid since it satisfies the content patterns. Therefore, the extraction task is successfully achieved and we can represent the extracted data in RDF as:

```
_:l1 dbp:email "user1@domain.com"@en .
```

were `_:l1` is a blank node (in the subject position) that represents the business card itself. The same process can be followed for the other attributes. In practice, RDF vocabularies to model information extraction tasks, like this barely showed

here, are needed. This is still an open research area that could help adding some metadata regarding the variables involved in the extraction, or even to assign URIs to identify common IE modules.

## 6 Related Work

A significant amount of research activity have been made in the topic of pattern recognition (see [2,12] among others). Pattern-based extraction of information and the use of lexico-syntactic patterns are far from new and have been used in a variety of tasks [10,4,16]. In information extraction, patterns are widely used on the Web [6,1]. Within the range of uses for patterns, [6,13] reported a reasonable success in ontology creation and population. Recently, Linked Data has been exploited as a background knowledge base to support wrapper induction [8], HTML tables interpretation [14], among other tasks. However, as far as we know, this paper is the first to explore the learning of lexico-syntactic patterns from string data embedded in LD datasets for validation. Our approach was inspired by algorithms proposed for wrapper induction [11,8], and a previous work focused on RDF extraction from Wikipedia tables [14].

## 7 Conclusions and Future Work

In this paper, we presented a method to build a content patterns database generated exclusively from DBpedia knowledge base, but applicable to any Linked Data dataset. By exploiting the implicit grammar present in the content of RDF properties whose range is in the set of literals, we could generate content patterns for such properties. These syntactic patterns are rich sources to be used in Information Extraction tasks as shown in this paper. Furthermore, we present some possible use cases where the database can be exploited.

To the best of our knowledge, our work is the first in conducting an analysis over properties with textual values; and also the first in generate a content pattern database from Linked Data datasets.

As future work, we identified two major directions in which our work can be extended and/or improved: (1) The content patterns database can be used in the evaluation of the knowledge base used to build the former. In other words, we can perform a checking of the RDF triples examining cases where the object values do not comply with the most common generated patterns. This will guide us to a consistency analysis of the knowledge represented by the LD dataset, where this can be cleaned increasing its value. This task also will give insights on common errors incurred in automatic extractions from semi-structured data sources. (2) The consideration of other LD datasets besides DBpedia, such as Freebase could help to enrich in both size and quality dimensions the database, making it more valuable.

## References

1. Banko, M., Cafarella, M.J., Soderland, S., Broadhead, M., Etzioni, O.: Open Information Extraction from the Web. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence. IJCAI'07, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2007) 2670–2676
2. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
3. Brickley, D., Guha, R., McBride, B.: RDF Schema 1.1. `http://www.w3.org/TR/rdf-schema/` (February 2014)
4. Brin, S.: Extracting Patterns and Relations from the World Wide Web. In: Selected Papers from the International Workshop on The World Wide Web and Databases. WebDB '98, London, UK, UK, Springer-Verlag (1999) 172–183
5. Cyganiak, R.: SPARQL for Tables: Turn CSV into RDF using SPARQL syntax. `https://github.com/cygri/tarql` (August 2013)
6. Etzioni, O., Cafarella, M., Downey, D., Kok, S., Popescu, A.M., Shaked, T., Soderland, S., Weld, D.S., Yates, A.: Web-scale Information Extraction in Knowitall: (Preliminary Results). In: Proceedings of the 13th International Conference on World Wide Web. WWW '04, New York, NY, USA, ACM (2004) 100–110
7. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF Representation for Publication and Exchange (HDT). Web Semantics: Science, Services and Agents on the World Wide Web **19** (2013) 22–41
8. Gentile, A.L., Zhang, Z., Augenstein, I., Ciravegna, F.: Unsupervised Wrapper Induction Using Linked Data. In: Proceedings of the 7th International Conference on Knowledge Capture. K-CAP'13, New York, NY, USA, ACM (2013) 41–48
9. Goyvaerts, J., Levithan, S.: Regular Expressions Cookbook - Detailed Solutions in Eight Programming Languages, Second Edition. O'Reilly (2012)
10. Hearst, M.A.: Automatic Acquisition of Hyponyms from Large Text Corpora. In: Proceedings of the 14th Conference on Computational Linguistics. COLING '92, Stroudsburg, PA, USA, Association for Computational Linguistics (1992) 539–545
11. Lerman, K., Minton, S.N., Knoblock, C.A.: Wrapper maintenance: A machine learning approach. J. Artif. Int. Res. **18**(1) (February 2003) 149–181
12. Liu, B.: Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data. Data-Centric Systems and Applications. Springer (2011)
13. Maynard, D.: Using Lexico-Syntactic Ontology Design Patterns for Ontology Creation and Population. In: Proc. of the Workshop on Ontology Patterns. (2009)
14. Muñoz, E., Hogan, A., Mileo, A.: Using Linked Data to Mine RDF from Wikipedia's Tables. In: Proceedings of the 7th ACM International Conference on Web Search and Data Mining. WSDM'14, New York, NY, USA, ACM (2014) 533–542
15. Muñoz, S., Pérez, J., Gutierrez, C.: Simple and Efficient Minimal RDFS. Web Semantics: Science, Services and Agents on the World Wide Web **7**(3) (2009) 220–234 The Web of Data.
16. Soderland, S.: Learning Information Extraction Rules for Semi-Structured and Free Text. Mach. Learn. **34**(1-3) (February 1999) 233–272
17. Vatant, B., Vandenbussche, P.Y.: Linked Open Vocabularies (LOV). `http://lov.okfn.org/` (July 2014)