ATOLL – A framework for the automatic induction of ontology lexica

Sebastian Walter, Christina Unger, Philipp Cimiano
Semantic Computing Group, CITEC, Bielefeld University

Abstract

There is a range of large knowledge bases, such as Freebase and DBpedia, as well as linked data sets available on the web, but they typically lack lexical information stating how the properties and classes they comprise are realized lexically. Often only one label is attached, if at all, thus lacking rich linguistic information, e.g. about morphological forms, syntactic arguments or possible lexical variants and paraphrases. While ontology lexicon models like *lemon* allow for defining such linguistic information with respect to a given ontology, the cost involved in creating and maintaining such lexica is substantial, requiring a high manual effort. Towards lowering this effort we present ATOLL, a framework for the automatic induction of ontology lexica, based both on existing labels and dependency paths extracted from a text corpus. We instantiate ATOLL with respect to DBpedia as dataset and Wikipedia as corresponding corpus, and evaluate it by comparing the automatically generated lexicon with a manually constructed one. Our results clearly corroborate that our approach shows a high potential to be applied in a semi-automatic fashion in which a lexicon engineer can validate, reject or refine the automatically generated lexical entries, thus having a clear potential to contributing to the reduction the overall cost of creating ontology lexica.

Keywords: ontology lexicalization, knowledge bases, ontologies, natural language processing

1. Introduction

The amount of structured knowledge available on the web is increasing. The linked data cloud, consisting of a large amount of interlinked RDF (*Resource Description Framework*[1]) datasets, has been growing steadily in recent years, now comprising more than 30 billion RDF triples[2]. Knowledge bases such as

[1] http://www.w3.org/TR/rdf-primer/
[2] http://www4.wiwiss.fu-berlin.de/lodcloud/state/

Freebase[3] and DBpedia[4] are huge and become more and more popular for various applications. Structured data is by now also collected and exploited by search engines such as Google, e.g. in the form of knowledge graphs[5] that are used to enhance search results. As the amount of structured knowledge available keeps growing, intuitive and effective paradigms for accessing and querying this knowledge become more and more important. An appealing way of accessing this growing body of knowledge is through natural language. In fact, in recent years several researchers have developed question answering systems that provide access to the knowledge in the linked data cloud (e.g. [17], [27], [30], [5]). Further, there have been approaches to applying natural language generation techniques to RDF in order to verbalize knowledge contained in RDF datasets (e.g. [20], [26], [7]).

For all such systems, knowledge about how properties, classes and individuals are verbalized in natural language is required. One way to capture such knowledge are ontology lexica. The *lemon* model, for example, has been developed for exactly this purpose: creating a standard format for publishing lexical information about how ontology elements are verbalized in different languages as RDF data. However, the creation of lexica for large ontologies and knowledge bases such as the ones mentioned above involves a high manual effort. Towards reducing the costs involved in building such lexica, we propose a corpus-based approach for the induction of lexica for a given ontology which is capable of automatically inducing an ontology lexicon given a knowledge base or ontology and an appropriate text corpus. Our approach is supposed to be deployed in a semi-automatic fashion by proposing a set of lexical entries for each property and class, which are to be validated by a lexicon engineer, e.g. using a web interface such as *lemon source*[6].

As an example, consider the property `spouse` as defined in DBpedia. In order to be able to answer natural language questions such as 'Who is Obama married to?' or 'Who is the wife of Obama?', we need to know possible lexicalizations of this property, such as 'married to', 'wife of', and so on. Our approach is able to find such lexicalizations on the basis of a sufficiently large corpus. The approach relies on the fact that many existing knowledge bases are populated with instances, i.e. triples relating entities through properties, such as $\langle$ `resource:Barack_Obama`, `dbpedia:spouse`, `resource:Michelle_Obama` $\rangle$. On the basis of these instances, it searches for occurrences of both entities in sentences in the corpus. These sentences are then parsed using a dependency parser, and the resulting dependency graphs are used to extract lexicalization patterns that very likely express the property or class in question.

The approach is trained on the 20 classes and 60 properties available as part of the ontology lexicon induction task of the CLEF 2013 challenge on *Question*

[3] `http://www.freebase.com/`
[4] `http://dbpedia.org/`
[5] `http://www.google.com/insidesearch/features/search/knowledge.html`
[6] `http://lemon-model.net/source/`

Answering over Linked Data, and is evaluated on the recently manually created English DBpedia lexicon [28], which contains lexicalizations for 353 classes as well as the 232 most frequent properties. We present the results in terms of precision, recall and F-measure and discuss its usefulness for semi-automatic lexicon engineering.

The paper is structured as follows. In Section 2 we briefly introduce *lemon*. Then we present our approach to automatically inducing *lemon* lexica in Section 3, and give details of an evaluation of our approach with respect to 585 classes and properties from the DBpedia ontology in Section 4. Before concluding, we discuss some related work in Section 5 and our future plans in Section 6.

2. The lexicon-ontology interface

While ontologies formalize a shared conceptualization, ontology lexica [23] describe the lexical encoding of that conceptualization in a particular language. A number of models were proposed for describing the ontology-lexicon interface, among them *lemon*[7] (*Lexicon Model for Ontologies*) [18], which is currently subject to standardization efforts in a W3C community group[8]. *lemon* is a model for the declarative specification of multilingual, machine-readable lexica in RDF that capture morphological, syntactic and semantic aspects of lexical items relative to some ontology.

The core of the *lemon* model comprises lexical entries together with their forms, specified by a simple string or some phonological representation, and their meaning, given by reference to an ontology element. For example, an entry for the DBpedia class `Star` could look as follows:

```
:star a lemon:Word;
  lemon:canonicalForm [ lemon:writtenRep "star"@en ];
  lemon:sense         [ lemon:reference dbpedia:Star ].
```

It specifies the entry as a word with the English lemma 'star' and a meaning referring to the ontology class `Star`. In the same way we can specify entries for lexical variants, such as 'sun', which refer to the same class, as well as lexical entries in different languages.

By specifying the meaning as reference to an ontology element, *lemon* ensures a clean separation between the ontological and lexical layer.

Since *lemon* abstracts from specific linguistic theories and grammar formalisms, all linguistic categories such as parts of speech, syntactic frames and argument roles have to be defined in an external linguistic ontology. In this paper we assume this linguistic ontology to be LexInfo [8], but it can easily be adapted to any other. Using LexInfo, the above entry can be extended as follows, now also specifying the part of speech as well as singular and plural forms:

[7] `http://lemon-model.net`
[8] `http://www.w3.org/community/ontolex/`

```
:star a lemon:Word;
  lexinfo:partOfSpeech lexinfo:noun;
  lemon:canonicalForm [ lemon:writtenRep "star"@en;
                        lexinfo:number lexinfo:singular ];
  lemon:otherForm     [ lemon:writtenRep "stars"@en;
                        lexinfo:number lexinfo:plural ];
  lemon:sense         [ lemon:reference dbpedia:Star ].
```

In addition to the core classes and properties, *lemon* provides several modules for capturing further aspects of the lexicon-ontology interface. One of them is the *syntax and mapping* module that allows for specifying syntactic frames as well as their arguments and how those arguments map to the semantic arguments. As an example, consider the following verb entry 'discover' for the DBpedia property `discoverer`.

```
:discover a lemon:Word;
 lexinfo:partOfSpeech lexinfo:verb;
 lemon:canonicalForm [ lemon:writtenRep "discover"@en ];
 lemon:synBehavior   [ a lexinfo:TransitiveFrame;
                       lexinfo:subject      :arg1;
                       lexinfo:directObject :arg2 ];
 lemon:sense         [ lemon:reference dbpedia:discoverer;
                       lemon:subjOfProp     :arg2;
                       lemon:objOfProp      :arg1 ].
```

It specifies the part of speech, the lemma 'discover', a transitive verb frame as the entry's typical syntactic context with two arguments (a subject and a direct object) as well as a meaning referring to the property `discoverer` with two semantic arguments. Using the same URIs for syntactic and semantic arguments describes a specific argument mapping, in this case that the semantic subject of the property corresponds to the syntactic direct object, and the semantic object corresponds to the syntactic subject. This means that the triple ⟨ `resource:Uranus`, `dbpedia:discoverer`, `resource:William_Herschel` ⟩ can be verbalized as 'William Herschel discovered Uranus' (and not 'Uranus discovered William Herschel').

In order to support lexicon engineers in the construction of lexical entries and liberate them from the need of writing RDF, *lemon* comes with a design patterns library[9] [19]. Using these patterns, the above entries would have the following equivalent representations:

```
ClassNoun("star",dbpedia:Star) with plural "stars"
ClassNoun("sun", dbpedia:Star) with plural "suns"

StateVerb("discover",dbpedia:discoverer,
  propSubj = DirectObject,
  propObj  = Subject)
```

[9] `https://github.com/jmccrae/lemon.patterns`

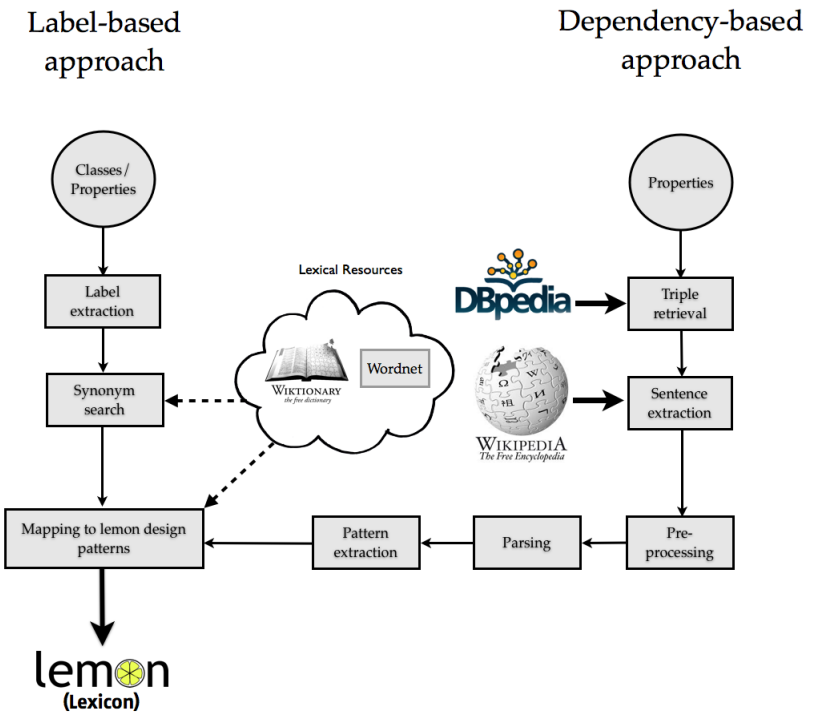


Figure 1: ATOLL system architecture

The lexical entries induced by our approach are specified using these patterns, as this makes them shorter, more concise and easier to read than RDF, and therefore facilitate a manual validation and correction step following the automatic induction of lexical entries.

3. Approach

The architecture of ATOLL is depicted in Figure 1. The input is an ontology together with an RDF knowledge base as well as an (ideally in-domain) text corpus; the output is a lexicon in *lemon* format, specifying lexicalizations both for ontology classes and properties. The approach relies on two basic processing chains: one *label-based* method that extracts lexicalizations from ontology labels and additional information, such as synonyms, from external lexical resources, and one *dependency-based* method that extracts lexicalization patterns for properties from the available text corpus. The main idea is to start from pairs of entities that are related by a given property, find occurrences of those entities in the text corpus, and generalize over the dependency paths that connect them.

We instantiate the approach using the DBpedia ontology and a corresponding English Wikipedia corpus. The DBpedia ontology covers a wide range of

domains and describes over two million individuals, i.e. it provides a large cross-domain dataset for which corresponding text corpora in a multitude of languages are available. It is thus an ideal basis to assess the performance and benefits of automatic ontology lexicalization. Moreover, with DBpedia being the central hub of the linked data cloud, a lexicon capturing verbalizations of the DBpedia ontology in several languages will prove useful beyond a particular project or application. However, our approach is general enough to apply also to other ontologies, knowledge bases and corpora.

In the following, we first present the procedure behind constructing lexical entries from dependency paths, and then describe the lexicalization of ontology elements on the basis of their labels.

3.1. Dependency-based approach

Figure 2 provides an overview of the dependency-based approach. The input is a property from a given ontology and a related text corpus; the output is a *lemon* lexicon. The approach comprises the following steps:

1. First, all pairs of entities connected through a given property are extracted from the knowledge base.
2. These pairs of entities are then used to find sentences in the text corpus that mention both entities, assuming that such sentences contain potential lexicalizations for the property in question.
3. Next, the retrieved sentences are preprocessed and parsed with a dependency parser, and the shortest path between the two given entities is extracted from the resulting dependency graph.
4. Depending on the path in question and the involved parts of speech, lexical entries are generated.

In the following, we describe these steps in more detail.

3.1.1. Triple retrieval and sentence extraction

Given a property, the first step consists in extracting all entities that are connected through the property from a given RDF knowledge base. For the DBpedia property `spouse`, for example, the following triples are returned (among many others):[10]

⟨`res:Barack_Obama, dbpedia:spouse, res:Michelle_Obama`⟩
⟨`res:Hilda_Gadea, dbpedia:spouse, res:Che_Guevara`⟩
⟨`res:Mel_Ferrer, dbpedia:spouse, res:Audrey_Hepburn`⟩

Next, for each triple ⟨`s,p,o`⟩ that was extracted for a property `p`, we retrieve all sentences from the text corpus in which the labels of both entities `s` and `o` occur, relying on an inverted Lucene index, which in our case contains around 25 million English sentences from Wikipedia, pre-tagged using the part-of-speech tagger included in the Natural Language Toolkit[11]. Currently, this lookup step

[10] Throughout the paper, we use the prefix `dbpedia` for `http://dbpedia.org/ontology/` and `resource` or `res` for `http://dbpedia.org/resource/`.

[11] `http://nltk.org/api/nltk.tag.html`

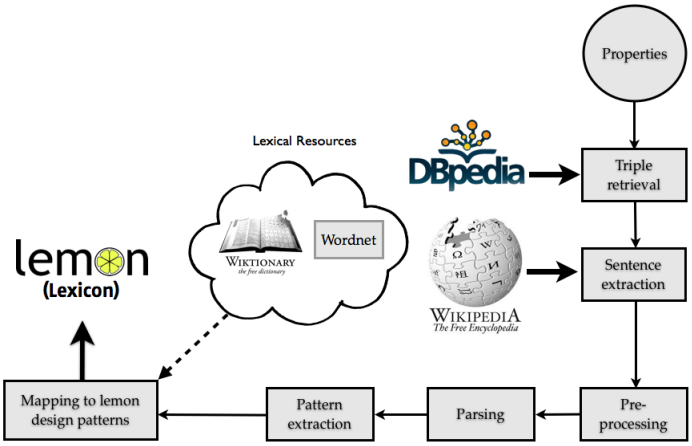


Figure 2: Overview of the dependency path approach

does not include any entity disambiguation.

In order to increase the number of retrieved entity pairs, and therefore the number of relevant sentences, we extract entity synonyms from the hyperlinks and corresponding labels provided by Wikipedia anchor texts. For the entity Barack Obama, for example, the anchor texts provide additional 45 written forms, including 'Barak Obama', 'Barack Hussein Obama', and '44th President of the United States'. For practical reasons, we use only the top 10 most frequent labels as synonyms for a given entity term, as otherwise the number of pairs and therefore lookups in the index become intractable. For the property `spouse`, for example, around 40,000 entity pairs are retrieved. Choosing the top 10 entity synonyms already increases the number of pairs to be looked up to 4 million. Table 1 lists the top 10 most frequent anchor texts for Barack Obama and Michelle Obama, together with their number of occurrences in Wikipedia.

3.1.2. Sentence preprocessing

After successfully retrieving sentences for a given triple, those sentences are preprocessed in order to improve parsing results. As an example, consider the following sentence for the entity pair Barack and Michelle Obama:

'Michelle Obama is the wife of Barack Obama, the current president.'

First, this sentence is tagged with part of speech information and stored in the index in the following form:

```
[ (Michelle,NNP), (Obama,NNP),
  (is,VBZ), (the,DT), (wife,NN), (of,TO),
  (Barack,NNP), (Obama,NNP),
  (the,DT), (current,JJ), (president,NN) ]
```

Next we apply a simple strategy for contracting entity names by concatenating all tokens tagged as `NNP` in a sequence. In our example, this leads to `(Michelle Obama,NNP)` and `(Barack Obama,NNP)`. As a further step, the resulting `NNP` terms are replaced with the entity labels that were used to retrieve the sentence, in this case 'Barack Obama' and 'Michelle Obama'. For the above sentence it does not make a difference, but it would, for example, normalize terms like 'First Lady Michelle Obama' or 'U.S. President Obama'. In order to minimize errors during parsing, also all whitespaces are removed, in our case reducing the proper nouns to `(BarackObama,NNP)` and `(MichelleObama,NNP)`.

Alternatively, we could include a named entity recognition tool, but have refrained from doing so in order to keep the approach very simple and not introduce additional sources of errors.

This sentence preprocessing procedure is slightly more complicated for datatype properties, as datatypes require a different kind of normalization. Consider, for example, the property `birthDate` with range `xsd:Date`. One of the received triples is ⟨`res:Barack_Obama,dbo:birthDate,1961-08-04`⟩. Since searching for sentences with an occurence of '1961-08-04' in the corpus will not give results, the date has to be changed to the usual date representations occurring in text, such as '08.04.1986', 'August 4 1961', '4 August 1961', and so on. Given these reformulations, one of the retrieved sentences is 'Barack Obama was born on August 4 1961', stored as:

```
[ (Barack,NNP), (Obama,NNP),
  (was,VBD), (born,VBN), (on,TO)
  (August,NNP), (4,CD), (1961,CD) ]
```

Again all tokens tagged with `NNP` are joined, as well as groups of tokens tagged with `NNP` and `CD`, often describing a date, yielding:

```
[ (Barack Obama,NNP),
  (was,VBD), (born,VBN), (on,TO),
  (August 4 1961,NNP) ]
```

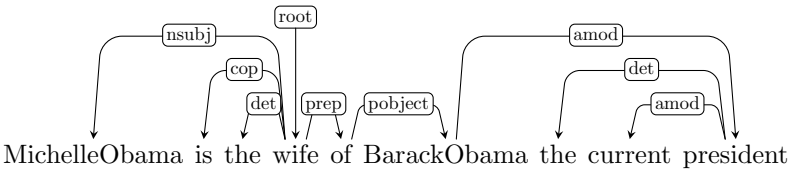
In the last step we again replace the entity names with the labels that were used for searching, and replace white spaces.

A list of the ten most frequent ranges for datatype properties in DBpedia 3.8 is given in Table 2. For each of these datatypes, a number of language-specific heuristics need to be implemented, as we have detailed above for the case of dates. Currently, ATOLL has built-in support for all those datatypes except for `xsd:double`, as this includes negative numbers. Attributes with negative numbers have turned out to be difficult to recognize, as the expression of negativity is very specific to the property in question. For instance, for altitude negativity is expressed as 'below see level', while for temperatures it is expressed as 'below zero'. We intend to cover such negative values in future work.

3.1.3. Dependency parsing

After successful preprocessing, the retrieved sentences are parsed using the MaltParser[12] [21] in an off-the-shelf fashion with the pre-trained English model. We have not yet tried to improve the results using the MaltOptimizer [3] or other dependency parsers such as the Stanford Parser, which—according to Stevenson [25]—delivers better results.

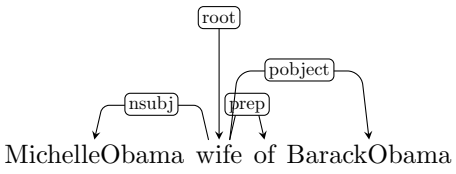
For the above example sentence 'Michelle Obama is the wife of Barack Obama, the current president' the following dependency parse is built:



In order to minimize further work, all parsed sentences are stored in a Lucene index, together with the corresponding entities, the corresponding URI of the property and the ID of the Wikipedia article from which the sentence was extracted.

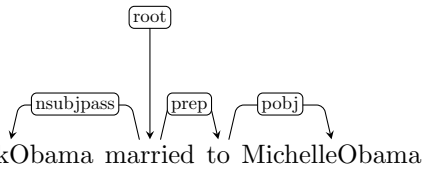
3.1.4. Pattern extraction

In order to extract lexicalization patterns, we start from one of the given entities and traverse the dependency graph in order to find the shortest path connecting it to the other entity. We focus on the shortest path, as this contains all words relevant for lexicalization while removing irrelevant ones such as modifiers. (We briefly discuss a resulting shortcoming of this in Section 4.4.) For the example 'Michelle Obama is the wife of Barack Obama, the current president', the direct and shortest path is the following one (which disregards the copula 'is', the determiner 'the', and the appositive 'the current president'):



This graph is captured as the pattern `[MichelleObama(subject), wife(root), of(preposition), BarackObama(object)]`, which represents one possible noun lexicalization for the property `spouse`. Another lexicalization pattern for `spouse` is retrieved from the depedency graphs for sentences like 'Barack Obama is married to Michelle Obama' with the following shortest path (which is again disregarding the copula):

[12] `http://www.maltparser.org/`



The resulting pattern is `[BarackObama (subject), married (root), to (preposition), MichelleObama (object)]`.

In order to determine the frequency of such patterns in the corpus, we replace the specific arguments by variables x and y, and thereby group all equivalent paths. This allows us to implement a simple frequeny-based cut-off, as for example done in [9] and [12], in order to reduce noise among the candidate lexicalizations by only generating lexical entries for patterns that occur at least twice.

3.1.5. Generation of lexical entries

In the following step each pattern is converted into a lexical entry, based on the part of speech of the root and the types of its arguments (direct object, prepositional object, etc.). The pattern [x `(subject), wife (root), of (preposition),` y `(object)]`, for example, would correspond to the following noun entry:

```
:wife a lemon:LexicalEntry ;
  lexinfo:partOfSpeech lexinfo:noun ;
  lemon:canonicalForm [ lemon:writtenRep "wife"@en ] ;
  lemon:synBehavior   [ rdf:type lexinfo:NounPPFrame ;
                        lexinfo:copulativeArg       :x_subj ;
                        lexinfo:prepositionalObject :y_pobj ] ;
  lemon:sense         [ lemon:reference dbpedia:spouse;
                        lemon:subjOfProp :x_subj ;
                        lemon:objOfProp  :y_pobj ] .

:y_pobj lemon:marker [ lemon:canonicalForm
                     [ lemon:writtenRep "of"@en ] ] .
```

This entry comprises a part of speech (noun), a canonical form (the head noun 'wife'), a sense referring to the property `spouse` in the ontology, and a syntactic behavior specifying that the noun occurs with two arguments, a copulative argument that corresponds to the subject of the property and a prepositional object that corresponds to the object of the property and is accompanied by a marker 'of'. From a standard lexical point of view, the syntactic behavior might look weird. Instead of viewing the specified arguments as elements that are locally selected by the noun, they should rather be seen as elements that occur in a prototypical syntactic context of the noun. They are explicitly named as it would otherwise be impossible to specify the mapping between syntactic and semantic arguments.

As it is easier for non-experts to read and validate or correct the less complex representation offered by the *lemon* design patterns, we rather generate the following `RelationalNoun` pattern, equivalent to the above RDF representation:

```
RelationalNoun("wife",dbpedia:spouse,
    propSubj = PossessiveAdjunct,
    propObj  = CopulativeArg),
```

Since lexical entries always specify the lemma of the lexical item, while patterns usually contain inflected forms, we need to carry out an additional step that retrieves the lemma and the base form of the root word occurring in the pattern. To this end, we exploit the SPARQL endpoint containing Wiktionary at `http://wiktionary.dbpedia.org`. For the noun 'wife', the result is 'wife', while for the participle 'married' the lemma is 'married', and the base form is 'marry'. In case the base form of the participle is found, we generate a verb entry:

```
StateVerb("marry",dbpedia:spouse),
    propSubj = Subject,
    propObj  = DirectObject)
```

If the base form cannot be retrieved, then a relational adjective entry for the participle form is constructed:

```
RelationalAdjective("married",dbpedia:spouse),
    propSubj = Subject,
    propObj  = PrepositionalObject("to"))
```

3.2. Label-based approach

The label-based approach to the induction of lexical entries differs from the corpus-based approach described above in that it relies on the ontology label and external lexical resources to find synonyms for that label.

In our approach we use WordNet for finding synonyms. In order to disambiguate the relevant WordNet synset, we implement a LESK-based disambiguation algorithm [15] inspired by [4]. To this end, we first create a list of all synsets found for a given ontology label. In case the label belongs to a class, we retrieve 100 resources belonging to this class and then for each resource collect a bag of all words that are contained in the corresponding Wikipedia article. These bags of words are compared to the lemma of each synset, using normalized Levenshtein distance, finally averaging the results for each synset and choosing the highest ranked one.

The label of the DBpedia class `Activity`, for example, is 'activity', for which we retrieve the synonym 'action' from WordNet. For both nouns lexical entries are generated:

```
ClassNoun("activity",dbpedia:Activity)
ClassNoun("action",dbpedia:Activity)
```

The same processing is done for labels of properties, yielding, for example, the following entries for the property `spouse`:

```
RelationalNoun("spouse",dbpedia:spouse,
    propSubj = PossessiveAdjunct,
    propObj  = CopulativeArg)

RelationalNoun("partner",dbpedia:spouse,
  propSubj = PossessiveAdjunct,
  propObj  = CopulativeArg))

RelationalNoun("better half",dbpedia:spouse,
  propSubj = PossessiveAdjunct,
  propObj  = CopulativeArg)
```

As we will show in the following section, the label-based approach complements the dependency-based approach quite nicely.

4. Evaluation

We developed ATOLL using the data from the ontology lexicalization task of the QALD-3 challenge as development set, and then evaluated the approach on a manually constructed *lemon* lexicon for DBpedia. Results are reported using precision and recall, as well as frame accuracy as defined below. In this section, we describe in more detail the methodology and measures of evaluation, followed by a presentation and discussion of the results, including a comparison of how well the dependency-based and label-based approaches work on different kinds of properties.

We do not attempt to compare these results to previous results reported in [29]. First of all, the system has been significantly re-designed, for example now computing the frequency of lexicalizations on the basis of the resulting lexical entries, as several dependency patterns can be mapped to the same entry. Further, the corpus we use now is much bigger than the one previously used, containing more than 66 million sentences from English Wikipedia articles. Finally, the previous system was evaluated with respect to a much smaller gold standard comprising 30 properties and 10 classes, while we now evaluate on 177 properties and 326 classes (all properties and classes contained in the manual lexicon minus its overlap with the develoment set), thus yielding more reliable results.

4.1. Methodology and dataset

As development dataset we use the dataset associated to the ontology lexicalization task of the QALD-3 challenge[13] at CLEF 2013. It consists of 20 DBpedia

[13] `http://www.sc.cit-ec.uni-bielefeld.de/qald`

classes and 60 DBpedia properties that were randomly selected from different frequency ranges, i.e. including properties with a large amount of instances as well as properties with very few instances.

For testing purposes, we use the manually created *lemon* lexicon for DBpedia [28][14], which comprises lexicalizations of 326 classes and the 232 properties, thus covering 98% of the classes and approximately 20% of the properties. The lexicon currently contains 1,217 entries (443 class lexicalizations and 774 property lexicalizations), which amounts to approximately 1.8 entries per ontology concept (1.3 per class and 2.4 per property). From this dataset we removed all classes and properties used for training, in order to avoid any overlap, leaving a test dataset that is approximately 14 times bigger than the training dataset.

4.2. Evaluation measures

For each property and class, we evaluate the automatically generated lexical entries by comparing them to the manually created lexical entries along two dimensions: i) lexical precision, lexical recall and lexical F-measure at the lemma level, and ii) frame accuracy. In the first dimension, we evaluate how many of the gold standard entries for a property are generated by our approach (recall), and how many of the automatically generated entries are among the gold standard entries (precision), where two entries count as the same lexicaliztation if their lemma, part of speech and sense coincide. Thus lexical precision P_{lex} and recall R_{lex} for a property p are defined as follows:

$$\begin{aligned} P_{lex}(p) &= \frac{|entries_{auto}(p) \cap entries_{gold}(p)|}{|entries_{auto}(p)|} \\ R_{lex}(p) &= \frac{|entries_{auto}(p) \cap entries_{gold}(p)|}{|entries_{gold}(p)|} \end{aligned}$$

Where $entries_{auto}(p)$ is the set of entries for the property p in the automatically constructed lexicon, while $entries_{gold}(p)$ is the set of entries for the property p in the manually constructed gold lexicon. The F-measure $F_{lex}(p)$ is then defined as the harmonic mean of $P_{lex}(p)$ and $R_{lex}(p)$, as usual.

The second dimension, frame accuracy, is necessary in order to evaluate whether the specified subcategorization frame and its arguments are correct, and whether these syntactic arguments have been mapped correctly to the semantic arguments (domain and range) of the property in question. The accuracy of an automatically generated lexical entry l_{auto} for a property p w.r.t. the corresponding gold standard entry l_{gold} is therefore defined as:

$$A_p(l_{auto}) = (frameEq(l_{auto}, l_{gold}) + \frac{|args(l_{auto}) \cap args(l_{gold})|}{|args(l_{gold})|} + \frac{\sum_{a \in args(l_{auto})} map(a)}{|args(l_{auto})|})/3$$

[14] `https://github.com/cunger/lemon.dbpedia`

Where $frameEq(l_1, l_2)$ is 1 if the subcategorization frame of l_1 is the same as the subcategorization frame of l_2, and 0 otherwise, where $args(l)$ returns the syntactic arguments of l's frame, and where $map(a)$ is defined as follows:

$$map(a) = \begin{cases} 1, & \textit{if } a \textit{ in } l_{auto} \textit{ has been mapped to the same semantic argument} \\ & \textit{of } p \textit{ as in } l_{\text{gold}} \\ 0, & \textit{otherwise} \end{cases}$$

When comparing the argument mapping of the automatically generated entry with that of the gold standard entry, we only consider the class of the argument, simply being *subject* or *object*. This abstracts from the specific type of subject (e.g. copulative subject) and object (e.g. indirect object, prepositional object, etc.) and therefore allows for an evaluation of the argument mappings independently of the correctness of the frame and frame arguments. The frame accuracy $A_{lex}(p)$ for a property p is then computed as the average mean of the accuracy values of each generated lexicalization. All measures are computed for each property and then averaged for all properties. In the section below, we report only the average values.

4.3. Results

In this section, we present the results for the dependency-based and the label-based approach, as well as a hybrid strategy which returns the union of the lexical entries delivered by the label-based and dependency-based approaches at a number k of top lexical entries. During training phase we determine k such that it yields a good balance between recall and precision.

Figure 3 depicts the training results of the dependency-based strategy for object properties in terms of precision, recall and F-measure with respect to the number k of top lexical entries that were considered. The maximal F-measure obtained is 0.24, resulting from a precision of 0.3 and recall of 0.2. For k greater than 10, recall increases further but precision drops significantly. We therefore choose 10 as optimal value for k, which will be used in all further experiments on the test dataset.

Table 3 shows the results of all three strategies (label-based, dependency-based and hybrid) over the classes, object and datatype properties of the test dataset in terms of lexical precision P_{lex}, recall R_{lex} and F-measure F_{lex} as well as frame accuracy A_{lex}, with the above mentioned choice of $k = 10$. Table 4 then gives a more detailed account of the results for the object properties depending on the number of entity pairs that are retrieved for those properties. The table is divided into properties with 0–1,000 entity pairs, properties with 1,000–10,000 entity pairs, properties with 10,000–100,000 entity pairs, and finally properties with more than 100,000 entity pairs. Recall continuously increases with the number of entity pairs and drops only for properties with more than 100,000 entity pairs (mainly because such properties are very few which also happen to be difficult to lexicalize).

One important observation in both tables is that the recall of the label- and dependency-based approach is additive, which means that both approaches in fact find different lexicalizations for the same properties. From Table 3 we can

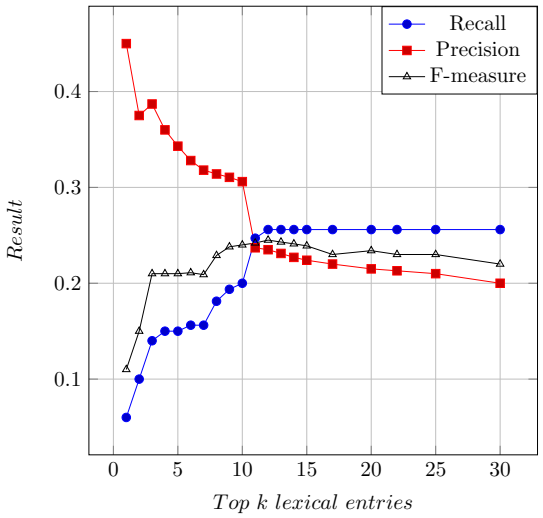


Figure 3: Recall, precision and F-measure for object properties in the training dataset

also see that the label-based approach gives better results for datatype properties, while the dependency-based approach works slightly better for object properties (although Table 4 shows that it fails for properties that are instantiated by less than 1,000 entity pairs). For these two reasons it makes sense to combine both approaches in a hybrid strategy. The overall results for the whole test dataset, containing all 326 classes and 177 properties, obtained by the hybrid approach are given in the last row of Table 3, showing an overall recall of 0.67 and an overall precision of 0.51.

While precision is rather poor for all approaches, except for the dependency-based approach over the datatype properties (where most patterns are filtered out before generating lexical entries), frame accuracy is generally very high, always being highest for the label-based approach.

It is also worthy to note that the threshold $k = 10$ was determined on a rather small training set and in fact turned out to not lead to optimal results on the test datatset. We recomputed the results for the dependency-based approach over object properties, setting k to 5 and 8, which resulted in an increase of precision from 0.09 to 0.12 for $k = 5$ and from 0.09 to 0.11 for $k = 8$, with only a small drop of recall from 0.35 to 0.27 for $k = 5$ and no drop in recall for $k = 8$.

With respect to performance, our implementation is rather efficient. It takes around two hours to extract patterns for all 177 properties, given that an index for the corpus was created (in our case requiring about 72 hours to tag and parse the whole of Wikipedia).

4.4. Discussion

Overall the results of the hybrid strategy over the whole dataset is decent but still far from being able to be used in a fully automatic setting. Roughly speaking, every second lexical entry that is generated is not appropriate (see precision of 0.51 in Table 3). In fact, we rather envision it as the basis of a semi-automatic scenario, in which lexical entries are generated automatically, but are then manually checked by a lexicon engineer and corrected if necessary. From this perspective, our approach has a clear potential to reduce the amount of manual work required to develop a high-quality lexicon.

In order to simulate such a semi-automatic procedure, we randomly selected five object properties, four datatype properties, and five classes from the test dataset, for which we inspected the top ten lexical entries generated by the dependency-based approach, both removing all incorrect lexicalizations and correcting those lexicalizations that were appropriate but contained some error in the generated lexical entry. Finally we also adapted these changes in the gold standard lexicon. We then recomputed precision, recall and F-measure for the examples; the results are listed in Table 5. The effect is quite strong: For object properties, F-measure increased from 0.26 to 0.39, and for datatype properties from 0.17 to 0.33.

These results mean that a lexicon engineer would on average have to examine ten lexical entries per property and discard a bit more than half of them, yielding about four lexical entries per property. After this manual correction step and adapting the changes to the gold standard, the lexicon reaches a precision of nearly 1, as all incorrect entries have been discarded, and a recall of about 0.8 (assuming that no new entries have been inserted).

A general limitation of our tool is that up to now it only knows four frame types (class nouns, relational nouns, relational adjectives and state verbs) but has no way to construct complex entries. Therefore more complex lexicalizations such as 'write music for' for the property `musicBy` cannot be captured yet.

Also, our assumption that the lexicalization of a property is equal to the direct dependency path between the entities it relates is too simple, as was also observed by Yao and Riedel et al. [31, 24]. Especially if the root is a verb, the path should include not only the entities but also other required arguments of the verb. For the property `almaMater`, for example, we find sentences of the form 'x received a degree from y', from which we extract the pattern `[(`x`,subject),(receive,root),(from,preposition),(`y`,object)]`, according to the direct path between x and y, while the correct pattern would rather be `[(`x`,subject),(receive,root),(degree,directObject),(from,preposition),(`y`,prepositionalObject)]`. In addition, it could be possible that one sentence contains several lexicalization patterns—a case that is not yet considered in our approach but is, for example, addressed in [31].

Another limitation of the approach is that not always all possible lexicalizations are contained in the corpus. For example, neither the lexicalization 'write music for' for the property `musicBy`, nor the lexicalization 'sublime at' for the property `sublimationPoint` are contained in the Wikipedia corpus and

therefore cannot be found by our approach. One future goal thus is to extend the information sources, finally moving to the whole web as corpus.

5. Related work

An approach to extracting lexicalization patterns from corpora that is similar in spirit to our approach is *Wanderlust* [2], which relies on a dependency parser to find grammatical patterns in a given corpus—Wikipedia in their case as in ours. These patterns are generic and non-lexical and can be used to extract any semantic relation. However, *Wanderlust* also differs from our approach in one major aspect. We start from a given property and use instance data to find all different lexical variants of expressing one and the same property, while *Wanderlust* maps each dependency path to a different property (modulo some postprocessing to detect subrelations). They are therefore not able to find different variants of expressing one and the same property, thus not allowing for semantic normalization across lexicalization patterns.

Another related tool is DIRT [16] (*Discovery of Inference Rules from Text*), also very similar to *Snowball* [1], which is based on an unsupervised method for finding inferences in text, thereby for example establishing that 'x is author of y is a paraphrase of 'x wrote y'. DIRT relies on a similarity-based approach to group dependency paths, where two paths count as similar if they show a high degree of overlap in the nouns that appear at the argument positions of the paths. Such a similarity-based grouping of dependency paths could also be integrated into our approach, in order to find further paraphrases. The main difference to our approach is that DIRT does not rely on an existing knowledge base of instantiated triples to bootstrap the acquisition of patterns from textual data, thus being completely unsupervised. Given the fact that nowadays there are large knowledge bases such as Freebase and DBpedia, there is no reason why an approach should not exploit the available instances of a property or class to bootstrap the acquisition process.

A system that does rely on existing triples from a knowledge base, in particular DBpedia, is BOA [13]. BOA applies a recursive procedure, starting with extracting triples from linked data, then extracting natural language patterns from sentences and inserting this patterns as RDF data back into the Linked Data Cloud. The main difference to our approach is that BOA relies on simple string-based generalization techniques to find lexicalization patterns. This makes it difficult, for example, to discard optional modifiers and thus can generate a high amount of noise, which has been corroborated by initial experiments in our lab on inducing patterns from the string context between two entities.

Espresso [22] employs a minimally supervised bootstrapping algorithm which, based on only a few seed instances of a relation, learns patterns that can be used to extract more instances. *Espresso* is thus comparable to our approach in the sense that both rely on a set of seed sentences to induce patterns. In our case, these are derived from a knowledge base, while in the case of *Espresso* they are manually annotated. Besides a constrast in the overall task (relation extraction in the case of *Espresso* and ontology lexicalization in our case), one difference

is that *Espresso* uses string-based patterns, while we rely on dependency paths, which constitutes a more principled approach to discarding modifiers and yielding more general patterns. A system that is similar to *Espresso* and uses dependency paths was proposed by Ittoo and Bouma [14]. A further difference is that *Espresso* leverages the web to find further occurrences of the seed instances. The corpus we use, Wikipedia, is bigger than the compared text corpora used in the evaluation by *Espresso*. But it would be bery interesting to extend our approach to work with web data in order to overcome data sparseness, e.g. as in [6], in case there is not enough instance data or there are not enough seed sentences available in a given corpus to bootstrap the pattern acquisition process.

What sets our approach appart from all above mentioned approaches is the combination of a dependeny-based and a label-based approach, benefiting from the complementary lexicalizations they find.

Furthermore, we improved the implementation of ATOLL by adding two simple constraints that were proposed by Fader et al. [11] and were shown to have a positive influence for pattern extraction. One constraint requires every multi-word pattern to begin with a verb, end with a preposition, and be a contiguous sequence of words in the sentence. The second constraint requires a two-word phrase to appear with at least a minimal number of distinct argument pairs in a large corpus. These constraints were extended in [10], additionally training a classifier to determine left and right bounds for the first argument of a pattern and the right bound of the second argument.

6. Future work

Currently our approach faces two main principled shortcomings, both concerning the lexicalization of more complex senses. First, ATOLL does not yet check whether patterns are appropriate lexicalizations only for a restricted domain or range of the target property. For example, the property `team`, which connects an athlete or manager with a sports team, could be verbalized as 'play for' in case the subject is a football, basketball or volleyball player, as 'race for' in case the subject is a cyclist or race driver, and as 'manage' if the subject is a sports manager. This could be captured by additionally checking the set of entity pairs that led to a certain pattern for a common subclass of the domain or range of the target property.

Second, ATOLL only finds lexicalizations for simple classes and properties, but not for more complex constructs such as property chains. For example, 'born in' is found as verbalization for the property `birthPlace`, connecting people to the city and sometimes also the country of their birth. This, however, misses the fact that in the dataset the country of birth is not always expressed by `birthPlace` directly, but often indirectly by the property chain `birthPlace` $\circ$ `country`. A generated lexicon should contain both senses for 'born in'. Similarly, a lexicon would benefit, e.g., from containing the verbalization 'grandchildren' for the propert chain `child` $\circ$ `child`. To be able to find patterns for such kinds of complex senses, we plan to extend ATOLL with other depen-

dency models, in addition to the shortest path model currently used, which were proven to be useful in [25].

Furthermore, we will explore ways to increase the number and quality of lexicalization patterns found. One such way is to investigate the extent to which our approach can benefit from the preprocessing of corpus sentences, e.g. by applying coreference resolution. Consider, for example, the following sentences: 'Barack Obama hosted a White House dinner. He and his personal secretary decided to mainly serve vegan food.' The pattern 'and his personal secretary' in the second sentence can only be extracted if the reference of the pronoun 'he' is resolved to the entity Barack Obama. One simple solution to start with, for example, would be to replace all pronouns by the label of the Wikipedia article from which the sentence was retrieved. So if the sentence 'He and his personal secretary decided to mainly serve vegan food' was retrieved from the Wikipedia article for Barack Obama, the pronoun 'he' would be replaced with 'Barack Obama'.

Most importantly, the next goal is to port ATOLL to other languages, starting with German and Spanish, later also moving to Non-Indo-European languages. Such porting mainly involves including a new language model for the dependency parser, importing language resources, such as Wiktionary, for the target language, and adapting the normalization procedure for datatypes. We plan to investigate how well our approach generalizes across different languages and to which extent it suffers from data sparsity for under-resourced languages.

7. Conclusion

We presented ATOLL, an approach for the automatic induction of ontology lexica, and instantiated it for DBpedia and a corresponding Wikipedia corpus. ATOLL starts from instance data and combines two strategies in order to find possible lexicalizations, one based on dependency paths extracted from a given text corpus, and one based on ontology labels and corresponding paraphrases that can be retrieved from external linguistic resources. Both strategies are independent of the particular RDF dataset and text corpus. We also presented a detailed evaluation of the dependency- and label-based strategies with respect to the different kinds of properties and classes, and sketched the use of ATOLL in a semi-automatic scenario, in which lexical entries are generated automatically, but are then manually checked and corrected if necessary.

Acknowledgment

This work was funded within the EU project PortDial (FP7-296170).

References

[1] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 85–94. ACM, 2000.

[2] Alan Akbik and Jürgen Broß. Wanderlust: Extracting semantic relations from natural language text using dependency grammar patterns. In *Proceedings of the Workshop on Semantic Search in Conjunction with the 18th Int. World Wide Web Conference*, 2009.

[3] Miguel Ballesteros and Joakim Nivre. Maltoptimizer: an optimization tool for maltparser. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, pages 58–62, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[4] Satanjeev Banerjee and Ted Pedersen. An adapted lesk algorithm for word sense disambiguation using wordnet. In *Computational linguistics and intelligent text processing*, pages 136–145. Springer, 2002.

[5] Abraham Bernstein, Esther Kaufmann, and Christian Kaiser. Querying the semantic web with ginseng: A guided input natural language search engine. In *15th Workshop on Information Technologies and Systems, Las Vegas, NV*, pages 112–126, 2005.

[6] Sebastian Blohm and Philipp Cimiano. Using the web to reduce data sparseness in pattern-based information extraction. In JoostN. Kok, Jacek Koronacki, Ramon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Knowledge Discovery in Databases: PKDD 2007*, volume 4702 of *Lecture Notes in Computer Science*, pages 18–29. Springer Berlin Heidelberg, 2007.

[7] N Bouayad-Agha, G Casamayor, and L Wanner. Natural language generation and semantic web technologies. *Semantic Web Journal*, 2012.

[8] Philipp Cimiano, Paul Buitelaar, John McCrae, and Michael Sintek. LexInfo: A declarative model for the lexicon-ontology interface. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(1):29–51, 2011.

[9] Hang Cui, Renxu Sun, Keya Li, Min-Yen Kan, and Tat-Seng Chua. Question answering passage retrieval using dependency relations. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 400–407. ACM, 2005.

[10] Oren Etzioni, Anthony Fader, Janara Christensen, Stephen Soderland, and Mausam Mausam. Open information extraction: The second generation. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume One*, pages 3–10. AAAI Press, 2011.

[11] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545. Association for Computational Linguistics, 2011.

[12] André Freitas, JoãoGabriel Oliveira, Seán O'Riain, Edward Curry, and JoãoCarlos Pereira da Silva. Querying linked data using semantic relatedness: A vocabulary independent approach. In Rafael Muñoz, Andrés Montoyo, and Elisabeth Métais, editors, *Natural Language Processing and Information Systems*, volume 6716 of *Lecture Notes in Computer Science*, pages 40–51. Springer Berlin Heidelberg, 2011.

[13] Daniel Gerber and Axel-Cyrille Ngonga Ngomo Ngomo. Bootstrapping the linked data web. In *Proc. of 1st Workshop on Web Scale Knowledge Extraction ISWC*, 2011.

[14] Ashwin Ittoo and Gosse Bouma. On learning subtypes of the part-whole relation: do not mix your seeds. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1328–1336. Association for Computational Linguistics, 2010.

[15] Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation*, pages 24–26. ACM, 1986.

[16] Dekang Lin and Patrick Pantel. DIRT - discovery of inference rules of text. In *Proceedings of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 323–328. ACM, 2001.

[17] Vanessa Lopez, Miriam Fernández, Enrico Motta, and Nico Stieler. Poweraqua: Supporting users in querying and exploring the semantic web. *Semantic Web*, 3(3):249–265, 2012.

[18] John McCrae, Dennis Spohr, and Philipp Cimiano. Linking lexical resources and ontologies on the semantic web with lemon. In Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Pan, editors, *The Semantic Web: Research and Applications*, volume 6643 of *Lecture Notes in Computer Science*, pages 245–259. Springer Berlin Heidelberg, 2011.

[19] John McCrae and Christina Unger. Design patterns for engineering the ontology-lexicon interface. In *P. Buitelaar and P. Cimiano (eds.): Towards the Multilingual Semantic Web*. Springer, to appear.

[20] Chris Mellish and Xiantang Sun. The semantic web as a¡ i¿ linguistic¡/i¿ resource: Opportunities for natural language generation. *Knowledge-Based Systems*, 19(5):298–303, 2006.

[21] Joakim Nivre, Johan Hall, and Jens Nilsson. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the fifth international conference on Language Resources and Evaluation (LREC2006), May 24-26, 2006, Genoa, Italy*, pages 2216–2219. European Language Resource Association, Paris, 2006.

[22] Patrick Pantel and Marco Pennacchiotti. Espresso: Leveraging generic patterns for automatically harvesting semantic relations. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 113–120. Association for Computational Linguistics, 2006.

[23] Laurent Prévot, Chu-Ren Huang, Nicoletta Calzolari, Aldo Gangemi, Alessandro Lenci, and Alessandro Oltramari. Ontology and the lexicon: a multi-disciplinary perspective. In *Ontology and the Lexicon: A Natural Language Processing Perspective*, pages 3–24. Cambridge University Press, 2010.

[24] Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M Marlin. Relation extraction with matrix factorization and universal schemas. In *Proceedings of NAACL-HLT*, pages 74–84, 2013.

[25] Mark Stevenson and Mark A. Greenwood. Dependency pattern models for information extraction. *Research on Language and Computation*, 7(1):13–39, 2009.

[26] Allan Third, Sandra Williams, and Richard Power. Owl to english: a tool for generating organised easily-navigated hypertexts from ontologies. In *Proceedings of of 10th International Semantic Web Conference (ISWC)*, pages 298—-303, 2011.

[27] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. Template-based question answering over rdf data. In *Proceedings of the 21st international conference on World Wide Web*, pages 639–648. ACM, 2012.

[28] Christina Unger, John McCrae, Sebastian Walter, Sara Winter, and Philipp Cimiano. A lemon lexicon for dbpedia. In *Proceedings of 1st International Workshop on NLP and DBpedia, October 21–25, Sydney, Australia*, volume 1064 of *NLP & DBpedia 2013*, Sydney, Australia, October 2013. CEUR Workshop Proceedings, `http://ceur-ws.org/Vol-1064`.

[29] Sebastian Walter, Christina Unger, and Philipp Cimiano. A corpus-based approach for the induction of ontology lexica. In Elisabeth Métais, Farid Meziane, Mohamad Saraee, Vijayan Sugumaran, and Sunil Vadera, editors, *Natural Language Processing and Information Systems*, volume 7934 of *Lecture Notes in Computer Science*, pages 102–113. Springer Berlin Heidelberg, 2013.

[30] Sebastian Walter, Christina Unger, Philipp Cimiano, and Daniel Bär. Evaluation of a layered approach to question answering over linked data. In Philippe Cudré-Mauroux, Jeff Heflin, Evren Sirin, Tania Tudorache, Jérôme Euzenat, Manfred Hauswirth, JosianeXavier Parreira, Jim Hendler,

Guus Schreiber, Abraham Bernstein, and Eva Blomqvist, editors, *The Semantic Web – ISWC 2012*, volume 7650 of *Lecture Notes in Computer Science*, pages 362–374. Springer Berlin Heidelberg, 2012.

[31] Limin Yao, Sebastian Riedel, and Andrew McCallum. Unsupervised relation discovery with sense disambiguation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 712–720. Association for Computational Linguistics, 2012.

	TF		TF
Barack Obama	16,809	Michelle Obama	866
Obama	957	Michelle Robinson	14
Barack H. Obama	63	First Lady Michelle Obama	9
Barak Obama	60	Michelle LaVaughn Robinson Obama	4
Barack Obama's	47	Michele Obama	4
Barrack Obama	41	Michelle Robinson Obama	3
Barack	31	Melvinia Shields	2
Barack_Obama	22	Mrs. Obama	1
Obama, Barack	13	Michelle obama	1
Sen. Barack Obama	10	Michelle_Obama	1

Table 1: The most frequent anchor texts for Barack Obama and Michelle Obama together with their term frequency (TF)

Range	Number of occurences
`xsd:string`	319
`xsd:double`	163
`xsd:nonNegativeInteger`	133
`xsd:date`	133
`xsd:gYear`	51
`xsd:integer`	37
`xsd:float`	18
`http://dbpedia.org/datatype/kilogram`	17
`http://dbpedia.org/datatype/kilometre`	15
`xsd:positiveInteger`	15

Table 2: Top ten ranges of datatype properties (using the prefix `xsd` for `http://www.w3.org/2001/XMLSchema#`).

	R_{lex}	P_{lex}	F_{lex}	A_{lex}
Classes				
Label-based approach	0.83	0.79	0.81	0.99
Object properties				
Label-based approach	0.15	**0.09**	0.11	**0.95**
Dependency-based approach	0.35	**0.09**	**0.14**	0.81
Hybrid approach	**0.50**	0.08	**0.14**	0.77
Datatype properties				
Label-based approach	0.35	0.19	**0.24**	**0.96**
Dependency-based approach	0.05	**0.60**	0.09	0.90
Hybrid approach	**0.39**	0.13	0.20	0.94
Whole dataset (classes and properties)				
Hybrid approach	0.67	0.51	0.58	0.93

Table 3: Results for the label-based approach over classes from the test dataset, and for all approaches over object and datatype properties from the test dataset

	R_{lex}	P_{lex}	F_{lex}	A_{lex}
Properties with 0–1,000 entity pairs				
Label-based approach	**0.45**	**0.29**	**0.35**	**0.70**
Dependency-based approach	0.00	0.00	0.00	0.00
Hybrid approach	**0.45**	0.18	0.25	**0.70**
Properties with 1,000–10,000 entity pairs				
Label-based approach	0.13	**0.08**	0.10	**0.95**
Dependency-based approach	0.36	**0.08**	**0.13**	0.81
Hybrid approach	**0.50**	0.06	0.11	0.77
Properties with 10,000–100,000 entity pairs				
Label-based approach	0.12	**0.07**	0.09	**0.96**
Dependency-based approach	0.41	**0.07**	**0.13**	0.78
Hybrid approach	**0.53**	**0.07**	**0.13**	0.78
Properties with more than 100,000 entity pairs				
Label-based approach	0.27	**0.18**	**0.22**	**0.95**
Dependency-based approach	0.16	0.03	0.05	0.90
Hybrid approach	**0.41**	0.06	0.11	0.88

Table 4: Results for all approaches over object properties from the test dataset, depending on the number of instances

	Fully automatic			Manually corrected			Adapted gold standard		
	R_{lex}	P_{lex}	F_{lex}	R_{lex}	P_{lex}	F_{lex}	R_{lex}	P_{lex}	F_{lex}
Classes									
`Brain` (1)	1.00	0.50	0.66	1.00	0.50	0.66	1.00	1.00	1.00
`Legislature` (0)	1.00	0.50	0.66	1.00	1.00	1.00	1.00	1.00	1.00
`Philosopher` (0)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
`Planet` (1)	1.00	0.50	0.66	1.00	0.50	0.66	1.00	1.00	1.00
`Politican` (0)	1.00	0.50	0.66	1.00	1.00	1.00	1.00	1.00	1.00
Average (0.4)	**1.00**	**0.60**	**0.72**	**1.00**	**0.80**	**0.86**	**1.00**	**1.00**	**1.00**
Object properties									
`almaMater` (4)	0.66	0.20	0.30	0.66	0.33	0.44	0.83	1.00	0.90
`birthPlace` (1)	0.33	0.10	0.15	0.33	0.50	0.39	0.50	0.50	0.50
`designer` (5)	1.00	0.33	0.50	1.00	0.38	0.54	1.00	1.00	1.00
`militaryUnit` (7)	1.00	0.10	0.18	1.00	0.13	0.22	1.00	1.00	1.00
`stateOfOrigin` (3)	0.50	0.11	0.18	0.50	0.33	0.38	0.80	1.00	0.88
Average (4)	**0.70**	**0.17**	**0.26**	**0.70**	**0.33**	**0.39**	**0.82**	**0.90**	**0.86**
Datatype properties									
`abbreviation` (1)	0.33	0.10	0.15	0.33	0.50	0.39	0.50	1.00	0.66
`birthDate` (6)	0.33	0.11	0.16	0.33	0.14	0.20	0.77	1.00	0.87
`militaryCommand` (7)	1.00	0.10	0.18	1.00	0.13	0.22	1.00	1.00	1.00
`title` (3)	1.00	0.10	0.18	1.00	0.33	0.49	1.00	1.0	1.00
Average (4.3)	**0.63**	**0.10**	**0.17**	**0.66**	**0.27**	**0.33**	**0.81**	**1.0**	**0.88**

Table 5: Sample results of a semi-automatic procedure, where numbers in brackets indicate how many of the inspected ten lexicalizations were kept and added to the gold standard