A corpus-based approach for the induction of ontology lexica

Sebastian Walter, Christina Unger, and Philipp Cimiano

Semantic Computing Group, CITEC, Bielefeld University

Abstract. While there are many large knowledge bases (e.g. Freebase, Yago, DBpedia) as well as linked data sets available on the web, they typically lack lexical information stating how the properties and classes are realized lexically. If at all, typically only one label is attached to these properties, thus lacking any deeper syntactic information, e.g. about syntactic arguments and how these map to the semantic arguments of the property as well as about possible lexical variants or paraphrases. While there are lexicon models such as *lemon* allowing to define a lexicon for a given ontology, the cost involved in creating and maintaining such lexica is substantial, requiring a high manual effort. Towards lowering this effort, in this paper we present a semi-automatic approach that exploits a corpus to find occurrences in which a given property is expressed, and generalizing over these occurrences by extracting dependency paths that can be used as a basis to create *lemon* lexicon entries. We evaluate the resulting automatically generated lexica with respect to DBpedia as dataset and Wikipedia as corresponding corpus, both in an automatic mode, by comparing to a manually created lexicon, and in a semi-automatic mode in which a lexicon engineer inspected the results of the corpus-based approach, adding them to the existing lexicon if appropriate.

Keywords: ontology lexicalization, corpus-based approach, lemon

1 Introduction

The structured knowledge available on the web is increasing. The Linked Data Cloud, consisting of a large amount of interlinked RDF datasets, has been growing steadily in recent years, now comprising more than 30 billion RDF triples[1]. Popular and huge knowledge bases exploited for various purposes are Freebase, DBpedia, and Yago.[2] Search engines such as Google are by now also collecting and exploiting structured data, e.g. in the form of knowledge graphs that are used to enhance search results.[3] As the amount of structured knowledge available keeps growing, intuitive and effective paradigms for accessing and querying

[1] `http://www4.wiwiss.fu-berlin.de/lodcloud/state/`

[2] `http://www.freebase.com/`, `http://dbpedia.org/`, `http://www.mpi-inf.mpg.de/yago-naga/yago/`

[3] `http://www.google.com/insidesearch/features/search/knowledge.html`

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 Open Data Cloud (e.g. [8], [13], [14], [2]). Further, there have been some approaches to applying natural language generation techniques to RDF in order to verbalize knowledge contained in RDF datasets (e.g. [10], [12], [4]). For all such systems, knowledge about how properties, classes and individuals are verbalized in natural language is required. The *lemon* model[4] [9] has been developed for the purpose of creating a standard format for publishing such lexica 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 (domain) 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*[5].

As an example, consider the property `dbpedia:spouse` as defined in DBpedia. In order to be able to answer natural language questions such as *Who is Barack Obama married to?* we need to know the different lexicalizations of this property, such as *to be married to*, *to be the 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. by triples relating entities through properties such as the property `dbpedia:spouse`. Our approach relies on such triples, e.g. ⟨`dbpedia:Barack_Obama`, `dbpedia:spouse`, `dbpedia:Michelle_Obama`⟩ to find occurrences in a corpus where both entities, the subject and the object, are mentioned in one sentence. On the basis of these occurrences, we use a dependency parser to parse the relevant context and generate a set of lexicalized patterns that very likely express the property or class in question.

The paper is structured as follows: in Section 2 we present the general approach, distinguishing the case of inducing lexical entries for properties and for classes. The evaluation of our approach with respect to 80 pseudo-randomly selected classes and properties is presented in Section 3. Before concluding, we discuss some related work in Section 4.

2 Approach

Our approach[6] is summarized in Figure 1. The input is an ontology and the output is a lexicon in *lemon* format for the input ontology. In addition, it relies on an RDF knowledge base as well as a (domain) corpus.

[4] For detailed information, see `http://lemon-model.net/`.
[5] `http://monnetproject.deri.ie/lemonsource/`
[6] Available at `https://github.com/swalter2/knowledgeLexicalisation`

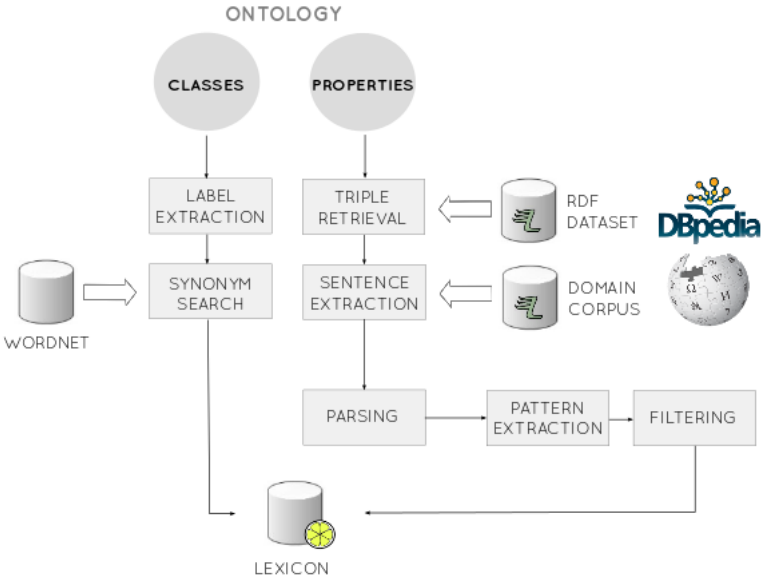


Fig. 1: System overview

The processing differs for properties and classes. In what follows, we describe the processing of properties, while the processing of classes, which does not rely on the corpus, is explained below in Section 2.5. For each property to be lexicalized, all triples from the knowledge base containing this property are retrieved. The labels of the subject and object entities of these triples are then used for searching the corpus for sentences in which both occur. Based on a dependency parse of these sentences, patterns are extracted that serve as basis for the construction of lexical entries. In the following, we describe each of the steps in more detail.

2.1 Triple retrieval

Given a property, the first step consists in extracting from the RDF knowledge base all triples containing that property. In the case of DBpedia, for the property `dbpedia:spouse`, for example, 44 197 triples are found, including the following[7]:

- (`resource:Barack_Obama`, *dbpedia:spouse*, `resource:Michelle_Obama`)
- (`resource:Alexandra_of_Denmark`, *dbpedia:spouse*, `resource:Edward_VII`)
- (`resource:Hilda_Gadea`, *dbpedia:spouse*, `resource:Che_Guevara`)
- (`resource:Mel_Ferrer`, *dbpedia:spouse*, `resource:Audrey_Hepburn`)

[7] Throughout the paper we use the prefixes `dbpedia` and `resource` for `http://dbpedia.org/ontology/` and `http://dbpedia.org/resource/`, respectively.

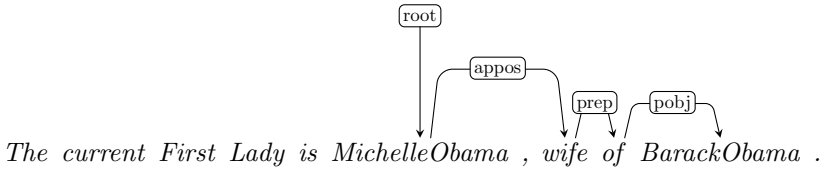
2.2 Sentence extraction and parsing

For each triple (s, p, o), that was extracted for a property p, we retrieve all sentences from the domain corpus in which the labels of both entities s and o occur. This step is performed relying on an inverted index. An example sentence extracted from Wikipedia for the subject/object pair *Barack Obama* and *Michelle Obama* is the following:

The current First Lady is Michelle Obama, wife of Barack Obama.

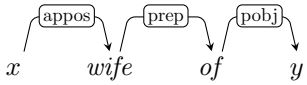
Each of the retrieved sentences is parsed with the pre-trained Malt dependency parser[8]. In order to avoid errors in parsing, the entity occurrences are replaced with a single word. For example, *Queen Silvia of Sweden* is replaced with *QueenSilviaofSweden*; this ensures that it is tagged as a named entity. Once a sentence has been parsed, the dependency parse is added to the index. This speeds up the process when the same sentence is retrieved again later.

From the dependency parses, we extract all paths that connect the entities in question. For the sentence above, for example, the following path connecting *Barack Obama* and *Michelle Obama* is found:



2.3 Pattern generation, postprocessing and filtering

On the basis of the discovered dependency paths, patterns are generated by abstracting over the specific entities occuring in the parse. The above mentioned path would for instance be generalized to:



In addition, the generalized patterns are postprocessed, e.g. by removing determiners such as *the*. To avoid unnecessary noise, only patterns with a length of at least three but not longer than six tokens are accepted. Also, if the entities x or y are related to another token by `nn`, i.e. are modifiers, the pattern is not considered. Additional processing, such as subsuming similar patterns under a single one, are planned but not yet implemented.

Finally, for each property we compute the relative frequency of the found patterns, i.e. the number of sentences that yielded a certain pattern in relation to the overall number of sentences for that property. We then consider only those patterns that occur at least twice and surpass a certain threshold θ, which is determined empirically in Section 3.3 below.

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

2.4 Generation of lexical entries

All patterns found by the above process, whose relative frequency is above a given threshold θ, are then transformed into a lexical entry in *lemon* format. For instance, the above mentioned pattern is stored as the following entry:

```
:wife a lemon:LexicalEntry ;
  lexinfo:partOfSpeech lexinfo:noun ;
  lemon:canonicalForm [ lemon:writtenRep "wife"@en ] ;
  lemon:synBehavior [ rdf:type lexinfo:NounPPFrame ;
                      lexinfo:copulativeArg       :x_appos ;
                      lexinfo:prepositionalObject :y_pobj ] ;
  lemon:sense [ lemon:reference
                <http://dbpedia.org/ontology/spouse>;
                lemon:subjOfProp :x_appos ;
                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*.[9] The specific subcategorization frame is determined by the kind of dependency relations that occur in the pattern. Currently, our approach covers nominal frames (e.g. *activity* and *wife of*), transitive verb frames (e.g *loves*), and adjectival frames (e.g. *Spanish*).

2.5 Lexicalization of classes

The lexicalization process for classes differs from that for properties in that the corpus is not used. Instead, for each class in the ontology, its label is extracted as lexicalization. In order to also find alternative lexicalizations, we consult WordNet to find synonyms. For example, for the class `http://dbpedia.org/ontology/Activity` with label *activity*, we find the additional synonym *action*, thus leading to the following two entries in the *lemon* lexicon[10]:

```
:activity a lemon:LexicalEntry ;
  lexinfo:partOfSpeech lexinfo:noun ;
```

[9] 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.

[10] As linguistic ontology we use ISOcat (`http://isocat.org`); in the examples, however, we will use the LexInfo vocabulary (`http://www.lexinfo.net/ontology/2.0/lexinfo.owl`) for better readability.

```
  lemon:canonicalForm [ lemon:writtenRep "activity"@en ] ;
  lemon:sense [ lemon:reference
                <http://dbpedia.org/ontology/Activity> ] .

:action a lemon:LexicalEntry ;
  lexinfo:partOfSpeech lexinfo:noun ;
  lemon:canonicalForm [ lemon:writtenRep "action"@en ] ;
  lemon:sense [ lemon:reference
                <http://dbpedia.org/ontology/Activity> ] .
```

These entries specify a part of speech (noun), together with a canonical form (the class label) and a sense referring to the class URI in the ontology.

3 Evaluation

In this section, we describe the methodology used in our evaluation as well as the evaluation measures, followed by a presentation and discussion of the results. Note that we evaluate our methodology in terms of how well it can support the creation of a lexicon. Of course the extracted patterns could also be used to find new instances of a relation within an information extraction paradigm. However, an evaluation of this potential use is out of the scope of the current paper.

3.1 Methodology and dataset

We evaluate our approach in two modes: fully *automatic* and *semi-automatic*. In the *automatic mode*, we evaluate the results of our corpus-based lexicon induction method by comparing the automatically generated lexicon with a manually constructed lexicon for DBpedia. The manually constructed lexicon was created by two persons not directly involved in the development and evaluation of the approach presented in this paper. In particular, these lexicon engineers did not have access to the results of the algorithm proposed here when creating their lexica. For the evaluation of our approach in the *semi-automatic mode*, the above mentioned lexicon engineers and one of the authors inspected the automatically generated lexica and added all appropriate lexical entries to their manually created lexicon in case it was appropriate and missing in the lexicon. In this evaluation mode we thus compare the automatically generated lexicon with a superset of the manually constructed lexica. By this, we do not penalize our approach for finding lexical entries that are correct but not contained in the manually constructed lexicon, thus representing a fair evaluation of our approach with respect to the targeted setting in which a lexicon engineer validates the automatically constructed lexical entries.

For the purposes of evaluation, we selected a training set for parameter tuning and a test set for evaluation, each consisting of 10 DBpedia classes and 30 DBpedia properties, in a largely pseudo-random fashion in the sense that we randomly selected properties from different frequency ranges, i.e. ranging from properties with very few instances to triples with many instances. We then filtered those that turned out to either have no instances—leaving in only one

empty property per set, `meltingPoint` and `sublimationPoint`, in order to be able to evaluate possible fallback strategies—or to not have an intuitive lexicalization, e.g. `espnId`. On average, the properties selected for training have 36 100 instances (ranging from 15 to 229 579), while the properties in the test set have 59 532 instances on average (ranging from 9 to 444 025). The training and test sets are also used in the ontology lexicalization task of the QALD-3 challenge[11] at CLEF 2013.

We use the training set to determine the threshold θ, and then evaluate the approach on the unseen properties in the test set.

3.2 Evaluation measures

For each property, 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, and ii) lexical 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 lexicalization 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:

$$P_{lex}(p) = \frac{|entries_{auto}(p) \cap entries_{gold}(p)|}{|entries_{auto}(p)|}$$

$$R_{lex}(p) = \frac{|entries_{auto}(p) \cap entries_{gold}(p)|}{|entries_{gold}(p)|}$$

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, lexical 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:

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

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

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

syntactic arguments of l's frame, and where

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

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 lexical 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 sections below, we will report only the average values.

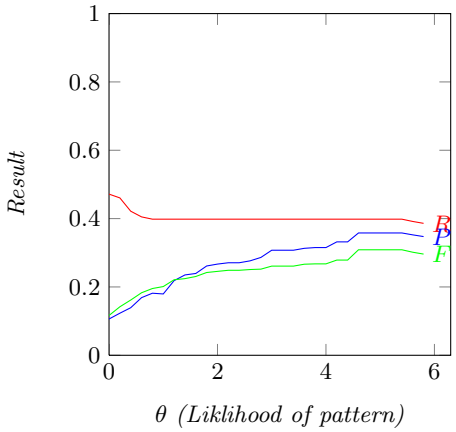
3.3 Results and discussion

Figure 2a shows results for the 30 training properties in automatic mode in terms of P_{lex}, R_{lex}, and F_{lex}, depending on the threshold θ. Accuracy is not plotted, as it is not influenced by θ. Neither are results for the classes, as they also do not vary with θ (recall is 0.73, precision is 0.55, and accuracy 0.9). The value θ is the likelihood that a specific pattern occurs given all the sentences expressing the property in question. On the basis of these results, we identify two θ values that are of interest: a low value around 0.2, which leads to high recall, and a high value around 5.0, which results in a drop in recall but an increase in precision. Having in mind a semi-automatic scenario, in which a lexion engineer validates and, if necessary, corrects the automatically generated lexical entries, we put more emphasis on recall, as it is easier and faster to filter out wrong entries than to discover and add missing one.

Figure 2b gives the results in terms of the average precision P_{lex}, recall R_{lex}, and F-measure F_{lex}, as well as average accuracy A_{lex} for the test set in both evaluation modes, for both relevant θ values, and on all 40 URIs. As with the training set, precision increases and recall decreases for higher θ.

In automatic mode, roughly half of the gold standard entries are generated, usually with a fair precision and accuracy, together with an additional amount of lexicalizations, ranging from 2 to 500, that are not in the gold standard. Of these additional lexicalizations, on average 1.4 entries are correct and were added to the gold standard lexicon. This improved precision and recall roughly by 0.2, accuracy even 0.3 and 0.4.

The property `programmingLanguage` is an example of a proeprty that performs quite bad in terms of precision. Here, six out of seven gold standard lexicalizations are found, leading to a recall of 0.85 and accuracy of 0.96, but also more than 500 wrong lexicalizations are created, yielding a precision of 0.01. The main reason is that the entity labels are not yet preprocessed and therefore take



(a) Results on train in automatic mode, with varying θ

	θ	R_{lex}	P_{lex}	F_{lex}	A_{lex}
Full	0.2	0.43	0.27	0.24	0.37
	5.0	0.42	0.4	0.34	0.37
Semi	0.2	0.64	0.48	0.41	0.76
	5.0	0.59	0.69	0.55	0.68

(b) Results on test in fully automatic and semi-automatic mode, with fixed θ

Fig. 2: Results on training (a) and test (b) dataset.

forms such as *C (programming language)*, which, in combination with the index lookup, leads to the extraction of sentences that might not be relevant, and also hinders the dependency path search between those entities. A similar problem is connected to datatype properties: Literals, such as floating point numbers and dates, have to be preprocessed in order to be found in corpus sentences. Also, the property `elevation`, for example, relates (among others) Barcelona with the number 12, which co-occur in quite some sentences that have nothing to do with the property, such as: *Originally from Barcelona, Spain, he was born March 12, 1971.* A more sophisticated way of filtering patterns found for datatype properties is thus of high importance.

A more general limitation of our tool is that it yet only creates three frame types, so more complex entries such as *to write music for* (lexicalizing the property `musicBy`) still cannot be created. Another problem of the approach is that not all lexicalizations in the gold standard lexicon do occur in the given corpus. For example, for *write music for* no sentence can be found; also no sentences with *sublime* are found that contain an entity pair related by the property `sublimationPoint`.

The average processing time amounts to around 15 seconds per property for the test dataset, assuming that the sentences extracted from the corpus have already been parsed.

4 Related work

In this section we briefly discuss related work in the area of extracting lexical patterns or paraphrases from corpora that verbalize a given relation in an ontology. An approach that is similar in spirit to our approach is *Wanderlust* [1] 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. Wanderlust differs from our approach in several aspects. First, our dependency paths are anchored in particular lexical entries. Second, we start from a given property and use instance data to find all different lexical variants of expressing one and the same property. Wanderlust, on the other hand, maps each dependency path to a different property (modulo some postprocessing to detect subrelations) and is in principle not able to find different variants of expressing one and the same property, so that semantic normalization is not achieved.

Another related tool is DIRT [7] (*Discovery of Inference Rules from Text*), which is an unsupervised method for finding inferences in text, so that *x is author of y* is a paraphrase of *x wrote y*. DIRT relies on a similarity-based approach to discover similar dependency paths, where two paths are similar if they show a high degree of overlap in the nouns that appear at the argument positions of the paths. We could esily extend our approach to also exploit similarity of the nouns occurring as arguments in patterns 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 very similar system, BOA [5], also relies on existing triples from a knowledge base, in particular DBpedia. 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. However, BOA rely on simple string-based generalization techniques to find actual patterns. This makes it difficult to discard optional modifiers and can generate a high amount of noise. This has been corroborated by initial experiments in our lab on inducing patterns from all the context between the two entities in question.

Espresso [11] employs a minimally supervised bootstrapping algorithm which, based on only a few seed instances of a relation, learns patterns to extract more instances. Espresso is thus comparable to our approach in the sense that both rely on a set of seed sentences to induces patterns. In our case, these are derived from a knowledge base, while in the case of Espresso they are manually annotated. A difference is that we rely on dependency paths connecting two entities, which yields a principled approach to discarding modifiers and yielding more general patterns. A system that is similar to Espresso and uses dependencies is the one proposed by Ittoo & Bouma [6]. In contrast to Espresso, we have not evaluated our approach on a relation extraction task. A further difference is that Espresso leverages the web to find further occurrences of the seed instances. The corpus we use, Wikipedia, is several order of magnitude bigger compared to the corpora used by Espresso, but nevertheless it would be interesting to extend our

approach to work with web data in order to overcome data sparseness (e.g. as in [3]). This is clearly an option to make use of in case not enough instances are available or not enough seed sentences can be found in the given corpus to bootstrap the pattern acquisition process.

5 Conclusion and future work

We presented an approach to the automatic induction of ontology lexica and instantiated it for DBpedia with a corresponding Wikipedia corpus. The approach itself is independent of the chosen domain and could be applied to any other dataset. The results will, however, depend on the specific modelling of the RDF data and the size and quality of the corpus. In particular, our approach faces two principled shortcomings that we want to address in future work. They concern domain and range restrictions as well as verbalizations of complex senses.

First, our approach does not yet check whether patterns are appropriate verbalizations only for a 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 *plays for* in case the subject is a football, basketball or volleyball player, as *races for* in case the subject is a cyclist or race driver, and as *manages* if the subject is a sports manager. This can 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, our approach only finds verbalizations for simple classes and properties, but not for more complex constructs such as property chains. For example, *born in* is found as verbalization for `dbpedia: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 directly by `dbpedia:birthPlace`, but often indirectly by the property chain `dbpedia:birthPlace` ∘ `dbpedia:country`. A generated lexicon should contain both senses for *born in*.

Additionally, future work will include an ongoing effort in increasing the number and quality of patterns found. One direction to explore is the extent to which our approach can benefit from the preprocessing of corpus sentences, e.g. by applying reference 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.

Overall, in this paper we have proposed a first step towards lowering the cost for creating lexica for a given ontology. Such lexica are crucial for any approach requiring access to information about how properties and classes are verbalized in a given language. While our evaluation has shown that our approach is promising, future work will provide a more extensive proof-of-concept, showing that the lexica can be exploited successfully for tasks such as question answering and natural language generation, also in a multilingual settings.

Acknowledgment This work has been funded by the European Union's Seventh Framework Programme (FP7-ICT-2011-SME-DCL) under grant agreement number 296170 (PortDial).

References

1. A. Akbik and J. 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.
2. A. Bernstein, E. Kaufmann, C. Kaiser, and C. Kiefer. Ginseng: A guided input natural language search engine. In *Proceedings of the 15th Workshop on Information Technologies and Systems*, pages 45–50, 2005.
3. Sebastian Blohm and Philipp Cimiano. Using the web to reduce data sparseness in pattern-based information. In *Proceedings of 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 18–29. Springer, 2007.
4. N. Bouayad-Agha, G. Casamayor, and L. Wanner. Natural language generation and semantic web technologies. *Semantic Web Journal*, in press.
5. D. Gerber and A.Ngomo. Bootstrapping the linked data web. In *Proceedings of the 10th International Semantic Web Conference (ISWC)*, 2011.
6. 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 (ACL)*, pages 1328–1336, 2010.
7. D. Ling and P. 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.
8. V. Lopez, M. Fernandez, E. Motta, and N. Stieler. Poweraqua: Supporting users in querying and exploring the semantic web. *Semantic Web Journal*, pages 249–265, 2012.
9. J. McCrae, D. Spohr, and P. Cimiano. Linking lexical resources and ontologies on the semantic web with lemon. In *Proceedings of the 8th Extended Semantic Web Conference (ESWC)*, pages 245–259. Springer, 2011.
10. C. Mellish and X. Sun. The semantic web as a linguistic resource: opportunities for natural language generation. In *Proceedings of 26th SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 298–303. Elsevier, 2006.
11. P. Pantel and M. Pennacchiotti. Espresso: Leveraging generic patterns for automatically harvesting semantic relations. In *Proceedings of the 21st International Conference on Computational Linguistics (COLING)*, pages 113–120. ACM, 2006.
12. A. Third, S. Williams, and R. 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.
13. C. Unger, L. Bühmann, J. Lehmann, A.-C. Ngonga-Ngomo, D. Gerber, and P. Cimiano. Sparql template-based question answering. In *Proceedings of the World Wide Web Conference (WWW)*, pages 639–648. ACM, 2012.
14. S. Walter, C. Unger, P. Cimiano, and D. Bär. Evaluation of a layered approach to question answering over linked data. In *Proceedings of the 11th International Semantic Web Conference (ISWC)*, pages 362–374. Springer, 2012.