Semantic Enrichment in OWL Knowledge Bases

Tim Schmiedl
Student: Computer Science (M.Sc.) University Freiburg
Sundgaualle 52 0107
73110 Freiburg
timschmiedl@neptun.uni-freiburg.de

ABSTRACT

The Semantic Web is still growing and the availability of large knowledge graphs increased over the past years. In spite of the growing number of knowlege bases there exist very few with a sophisticated schema. Often they only consist of a collection of facts with no consistant structure. Other knowlege bases contain only schema information without instances of the defined schemata.
But only the combination of both of these extremes, sophisticated schema and available instance data can enable powerful reasoning, easier checking for consistency and improved queryability.
This article show two methods for the semantic enrichment of large OWL knowlege bases. The first method focuses at finding and creating class expressions in an automatic or semiautomatic approach based on given knowlege in the graph. Whereas the second method enrichs knowlege bases with different types of OWL2 axioms.

General Terms

Theory

Keywords

Ontology engineering, Supervised machine learning, Knowledge Base Enrichment, OWL, Heuristics

1. INTRODUCTION

- semantic web: growing, bigger knowledge graphs

- Open data Initiative, Protoge ontologie etc -> hard to maintain, debug / find error inconsitencies

- lack sophisticated schema (only schema no instances, only facts)

- combination good schema + instance data –> powerful reasoning, consistency, improved query

- Example: Person birthplace + Benefits + missing info + semi-automated

2. ENRICHMENT OVERVIEW

The term enrichment in this context describes the extension of the (semantic) schema of a knowledge base. The process of knowledge base enrichment increases the semantic richness and the expressiveness of the knowledge base. The goal of the enrichment progress is to find axioms, witch can be added to the existing ontology. A special case is to find definition of classes and subclasses. This is closely related to the Inductive Logic Programming (ILP) as it is described later in this article. Ontology enrichment methods usually depend on machine learning or on applying heuristics to find additional axioms in the knowledge graph.[3]

As stated before, knowledge base enrichment usually work on existing data to improve the semantic schema. This supports the so called *grass-root* approach for creating ontologies. Here the whole ontology structure is not created upfront, but evolves over time and with every part of data that is added to the knowledge base.[3]

3. CLASS LEARNING

3.1 Motivation

The class learning approach is one method of enrichment of knowledge bases. It aims at finding new definition of classes to extend the semantic schema. For the motivation of the method consider the following example.

Example 1. For this example consider a knowledge base containing a class *President of the United States* with instance data like Abraham Lincoln, John F. Kennedy, Bill Clinton and Barack Obama. A class Learning algorithm may suggest that the President class is equivalent to the following two class expressions:

`Person and born in the USA`
`American citizen and born in the USA`[1]

These suggestions would then be presented to a knowledge engineer who than can decide if they are plausible and should be added to the knowledge graph. Should the engineer for instance choose the second statement he could check if there are instances of the class president, where the individual is

[1]The class 'American citizen' is here a subclass of 'Person', witch makes the second statement more specific.

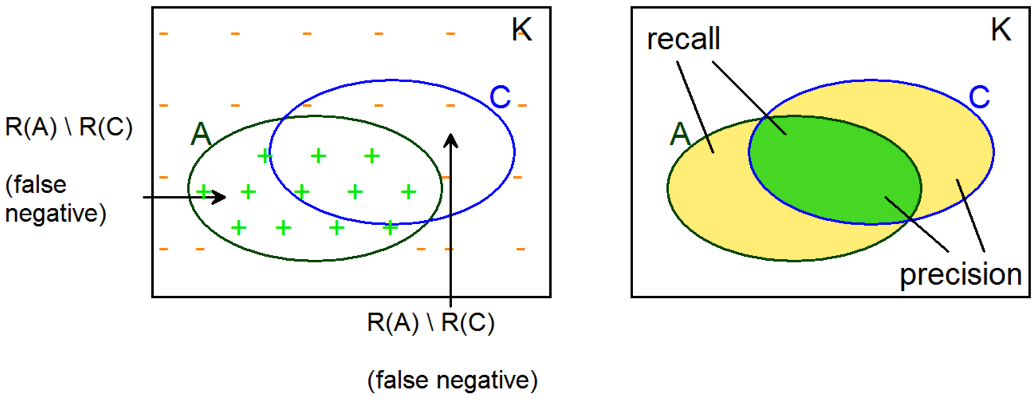


Figure 1: recall precision

not of type American citizen. This could indicate an error or missing information in the knowledge graph witch could be fixed by this semi-automated approach by the knowledge engineer.

3.2 Learning Problem

The problem of learning class definitions for given data depend on the so called inductive reasoning as opposed to inference or deductive reasoning. [6] Inductive Reasoning is also a key concept in Inductive Logic Programming.

Definition 1. We are searching for a formal description of the class A, which has existing instances in the examined ontology. A possible class expression C then contains axioms of the form $A \subseteq C$ or $A \equiv C$.

This means that the learned expression C is a description of the individuals of A. In our president example, the individuals are the presidents John F. Kennedy, Barack Obama etc. whereas C can be one of the suggested expression. In many cases there will be no exact solution for C, but rather an approximation. This can be the case, if the knowledge base contains false class assignments or missing information. In our example the birthplace of Thomas Jefferson might be missing in the ontology. However, if most of the other presidents have the correct birthplace the learning algorithm may still suggest the expressions. Again, missing information may be completed by the knowledge engineer.

In a complex knowledge base a class learning algorithm may find many new class definition and often different expression for the same class. Based on Occam's razor [**?**] simple solutions are preferred over complex ones, because they are more readable an thus easier for the knowledge engineer to evaluate. Simplicity is measured in an straight forward way: the length of an expression, which consists of role, concept and quantifiers. The algorithm is biased towards shorter expressions. [6]

3.3 Algorithm

One algorithm for solving the learning problem is called CELOE (Class Expression Learning for Ontology Engineering). It is described in [6]. An brief overview of CELOE

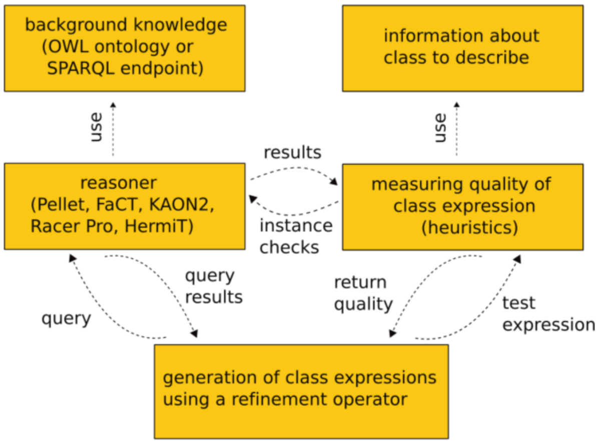


Figure 2: CELOE[6]

is given in Figure 2. The algorithm follows the "generate and test" approach which is the common concept in ILP. [10] In the learning process many class expression are created and tested against the background knowledge. Each of these class expressions is evaluated using different heuristics, which are described in detail in a separate section.

To find appropriate class expression to describe existing classes CELOE uses a so called *refinement operator*. The idea of these operator is based on the work in [7],[8] and [9]. Refinement operators are used to search in the space of expressions. It can be seen as a top-down algorithm as it is illustrated in Figure 3. As an example consider the following path ($\rightsquigarrow$ indicates a refinement step):

$T \rightsquigarrow Person \rightsquigarrow Person \sqcap takesPartIn.T \rightsquigarrow Person \sqcap takesPartIn.Meeting$

4. ENRICHMENT WITH OWL AXIOMS

OWL offers many different types of axioms for a method to enrich the knowledge base. Figure 4 shows the 3 steps in the enrichment workflow as described in [3]:

1. The first phase is about obtaining general information

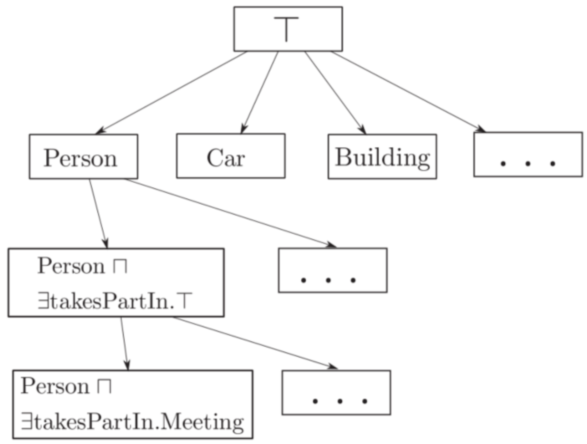


Figure 3: Tree[6]

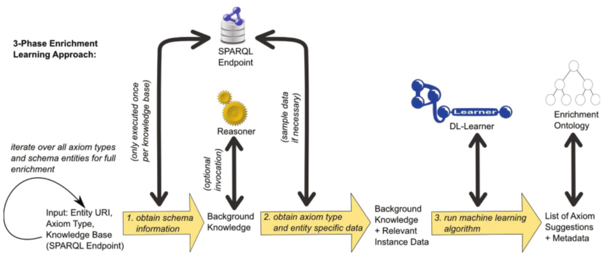


Figure 4: 3-Phase Enrichment Workflow[3]

about the knowledge base, in particular axioms, which define the class hierarchy are obtained. The schema queried via SPARQL, but is loaded only once.

2. In the second phase data is again obtained via SPARQL. These background checks allows the method to learn and test new axioms. Examples for axiom types are explained in the following text.

3. The score of axiom candidates is computed and the result is returned.

The algorithm for suggestion of new axioms is based on checking and counting RDF triples. In the following example we want to learn if the class A is an appropriate domain of a predicate p. For that we count the triples that match this statement to obtain a score.

Example 2. Lets look at the predicate `onto:attendsMeeting` and check if we can find a suitable candidate for a domain. Note that `onto:Manager` is a subclass of `onto:Employee`.

Listing 1: Triples written in turtle syntax

```
onto:Software_Engineer
        onto:attendsMeeting         onto:TeamMeeting;
        rdf:type                    onto:Employee.
onto:Software_Architekt
        onto:attendsMeeting         onto:TeamMeeting;
        rdf:type                    onto:Employee.
onto:Project_Manager
        onto:attendsMeeting         onto:ManagerMeeting;
        rdf:type                    onto:Manager.

onto:Manager    rdfs:subClassOf     onto:Employee.
```

Looking at this example we would obtain a score of 33.3 % (1 out of 3) for the class `onto:Manager` and 100 % (3 out of 3) for the class `onto:Employee`.
Obviously this extreme simple and straightforward method of calculating the score for the domain of a predicate p has some limitation. Mainly the method doesn't discriminate between a score calculated by having 100 out of 100 correct observations or only 3 out of 3. Different methods, for example the Wald method [?], overcome that problem. More involved heuristics are shown in the next section.

4.1 Obtaining Axioms via SPARQL queries

This section explains how SPARQL queries are used to extract information in step 2 of the enrichment workflow.

Subclass and Disjointness of classes

This query evaluates all individuals (`?ind`) and checks if they are instance of the user defined class definition in `$class`. In this query higher count indicates a better candidate for a superclass. Lower value indicates a disjointness.

```
SELECT ?type (COUNT(?ind ) AS ?count ) WHERE {
    ?ind a <$class >.
    ?ind a ?type .
} GROUP BY ?type
```

Subsumption and Disjointness of properties

A somewhat similar query can be used to learn subsumption and disjointness of predicates.

```
SELECT ?p (COUNT(? s ) AS ?count ) WHERE {
    ?s ?p ?o .
    ?s <$property> ?o .
} GROUP BY ?p
```

Domain and Range of properties

A query for the domain of a property counts the occurrences subjects of type `?type` having the property `$property`.

```
SELECT ?type COUNT(DISTINCT ?ind ) WHERE {
    ?ind <$property> ?o .
    ?ind a ?type .
} GROUP BY ?type
```

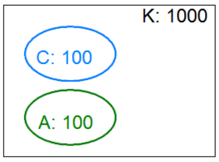
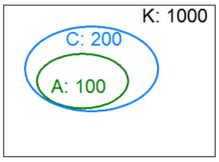
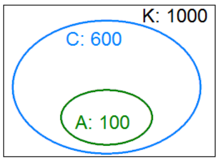
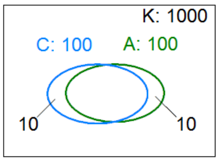
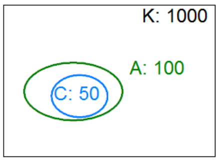
The query for the range of `$property` is analog. It can be also distinguished between object and data properties. Here only the queries for object properties are listed.

```
SELECT ?type (COUNT(DISTINCT ?ind ) AS ?cnt ) WHERE {
    ?s <$property> ? ind .
    ?ind a ?type .
} GROUP BY ?type
```

Inverse of Properties

To check if a property is inverse we check the `$property` with subject and object and in swapped position. As always we count how often the expression holds.

Table 1: Heuristics

Illustration	accuracy & recall	pred. acc.	F-Measure	A-Measure	Jaccard
	accuracy 0% recall 0%	80 %	0 %	0 %	0 %
	accuracy 50% recall 100%	90 %	66.7 %	75 %	50 %
	accuracy 16.7% recall 100%	50 %	28.6 %	58.3 %	16.7 %
	accuracy 90% recall 90%	98 %	90 %	90 %	81.8 %
	accuracy 50% recall 100%	95 %	66.7 %	75 %	50 %

```
SELECT ?p (COUNT(*) AS ?count ) WHERE {
    ?s <$property> ?o .
    ?o ?p ?s .
} GROUP BY ?p
```

5. HEURISTICS

5.1 Finding the right heuristic

A heuristic measures how well a given class expression fits the learning problem. [6] To test an algorithm we must have positive and negative examples. As we want to describe class A with C we can consider every instance of A as a positive and everything else as negative examples. The predictive accuracy can be described as:

$$predacc(C) = 1 - \frac{|R(A) \setminus R(C)| + |R(C) \setminus R(A)|}{n} \qquad n = |Ind(K)|$$

Here, Ind(K) stands for the set of individuals in the knowledge base. $R(A) \setminus R(C)$ are the false negatives whereas $R(C) \setminus R(A)$ are the false positives.
As you can see in Figure 2 for the term *precision* we consider the intersection of A and C ($R(A) \cap R(C)$) and the false positive. In other words, how many individuals are rightful considered with our class expression C.
For the other term *recall* we consider again the intersection of A and C and the false negative. This score tells us how many individuals of A we can describe with our class expression C.

Table 4.1 compares the score of four different heuristics. They are:

- **pred. acc.** The formula for the predictive accuracy was shown above.
- **F-Measure** The F-Measure is defined as harmonic mean of precision and recall. It can be weighted by a factor β, [6] choose 3 for β for learning super classes, wich gives recall a higher weight than precision.
  $F\text{-}Measure = \frac{\beta+1}{\frac{\beta}{rec} + \frac{1}{acc}}$
- **A-Measure** For the A-Measure we choose the arithmetic mean of precision and recall.
  $A\text{-}Measure = \frac{prec \times acc}{2}$
- **Jaccard** The Jaccard index is well known to compare two sets. It is defined as the size of the intersection divided by the size of the union of the sets.
  $Jaccard(A, B) = \frac{|A \cap C|}{|A \cup C|}$

5.2 Efficient heuristic computation

For big knowledge base the performance and efficiency of the algorithm is crucial. To compute and test class expression retrieval operations are often required. To minimize the cost and improve performance [6] provide three optimisations:

- **Reduction of instance checks** The obvious choice is to reduce the number of instance checks required to test expressions. Assuming that we want to learn an equivalence axiom for class A with the super class A'. Instead of checking every individual of the knowledge graph we can restrict our retrieval operation to instances of A'.

- **Approximate and closed world reasoning** To deal with the very high number of instance checks against the knowledge base CELOE uses a special reasoner.[6] It uses an approximate and incomplete reasoning for fast instance checks (FIC). The reasoner depends on the first run on a standard OWL reasoner to check instances and property relationships. The result of this first reasoning is stored in memory for fast (but approximated) instance checks. The reasoner follows a closed world assumption.

- **Stochastic coverage computation** This optimisation again reduces the necessary instance checks. Looking at the different heuristics, we can see that $|R(A)|$ needs to be computed only once whereas e.g. the expensive expression $R(A) \cap R(C)$ is computed for every different C. To improve performance we can try to approximate the result by testing randomly drawn objects an checking if we are sufficiently confident that our estimation is within a certain bound.

6. EVALUATION

To evaluate the methods of class learning, the authors of [6] tested their algorithm on a variety of real world ontologies of different sizes and domains. The goal of the evaluation consisted of 3 parts: determine the influence of reasoning and heuristics on suggestions, test the performance and efficiency on large real world ontologies and to test the accuracy of approximations described in section 5.2.

To perform the evaluation, the authors in [6] wrote a dedicated plugin for the Protégé editor. The plugin first looks for classes with enough instances. For each of these classes they ran the CELOE algorithm to generate suggestions for definitions, here they tested two different reasoner and the previously described five heuristics. The results of these tests were categorized in three main categories: (1) the suggestion improves the ontology (improvement), (2) the suggestion does not improve the ontology and therefor should not be added to the ontology (not), (3) adding the suggestion results in a modelling error (error). A small part of the results is shown in Table 6, the full evaluation results are shown in [6]. The second evaluation results were based on the performance of the approximation reasoner as described in section 5.2. The tests showed that an approximation lead to significant preformance improvement, this is especially the case for larger ontologies. The time required to test a class expression showed smaller variations and a performance gain of serveral orders of magnitudes has been achieved. The approximation has schon to be very accurate and had hardly any infuence on the learning algorithm.

Table 2: Evaluation

Reasoner/heuristic	Improv.	Not acc.	missed Improv.
Pellet/F-Measure	16.70	64.66	14.95
Pellet/A-Measure	16.70	64.66	14.95
Pellet/pred.acc.	16.59	64.93	15.22
Pellet FIC/F-Measure	36.60	52.62	1.90
Pellet FIC/A-Measure	36.19	52.84	1.63
Pellet FIC/pred.acc.	32.99	52.58	4.35

Prelimatary Evaluation has also been done in [3] for OWL axiom enrichment. The authors choosed DBpedia to test their algorithms. Table 6 shows a small subset of the results. In [3] the evaluation is discussed in great detail. For example the three newly discovered symmetic object properties are: `dbo:neighboringMunicipaly`,`dbo:sisterCollege` and `dbo:currentPartner`.

Table 3: Evaluation combined of Object and data property

axiom type	recall	new axioms	precision
SubClassOf	180/185	155	75 %
EquivalentClasses	0/0	1812	50 %
DisjointClasses	0/0	2449	100 %
PropertyDomain	833/942	1298	54 %
PropertyRange	291/1032	500	46 %
SymmetricProperty	0/0	3	100 %

7. RELATED WORK

Related work can be divided into two categories: the first part covers supervised machine learning with OWL, the second part is focused on (semi-)automated ontology engineering methods.
Early work of supervised learning in description logic was published in [4, 5], which uses the so called *least common subsumer* to solve the learning problem. Later work invented the concept of the refinement operator to solve the problem in a top-down approach.[2] The refinement operator was later adapted for description logic [7, 8, 9] and is used as described in CELOE.
The starting point of (semi-)automatic ontology engineering was set by [11], a formal concept analysis was descibed in [1]. Another interesting approach is presented in [12] which proposes to improve knowlege bases through relation exploration. It is implemented in the RELExO framework.

8. CONCLUSIONS

- more

- more

- more

9. REFERENCES

[1] F. Baader, B. Ganter, B. Sertkaya, and U. Sattler. Completing description logic knowledge bases using

formal concept analysis. In *In Proc. of IJCAI 2007*, pages 230–235. AAAI Press, 2007.

[2] L. Badea and S.-H. Nienhuys-Cheng. A refinement operator for description logics. In *Inductive logic programming*, pages 40–59. Springer, 2000.

[3] L. Bühmann and J. Lehmann. Universal owl axiom enrichment for large knowledge bases. *EKAW*, pages 57–71, 2012.

[4] W. W. Cohen, A. Borgida, and H. Hirsh. Computing least common subsumers in description logics. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pages 754–760. AAAI Press, 1992.

[5] W. W. Cohen and H. Hirsh. Learning the classic description logic: Theoretical and experimental results. *KR*, 94:121–133, 1994.

[6] J. Lehmann, S. Auer, L. Bühmann, and S. Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, pages 71–81, 2011.

[7] J. Lehmann and P. Hitzler. Foundations of refinement operators for description logics. *LNCS*, 4894, 2007.

[8] J. Lehmann and P. Hitzler. A refinement operator based learning algorithm for the alc description logic. *LNCS*, 4894, 2007.

[9] J. Lehmann and P. Hitzler. Concept learning in description logics using refinement operators. *Machine Learning Journal*, 78:203–250, 2010.

[10] S.-H. Nienhuys-Cheng and R. d. Wolf. *Foundations of Inductive Logic Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[11] S. Rudolph. Exploring relational structures via fle. In *Conceptual Structures at Work: 12th International Conference on Conceptual Structures. Volume 3127 of LNCS*. Springer, 2004.

[12] J. Völker and S. Rudolph. Fostering web intelligence by semi-automatic owl ontology refinement. IEEE, 2008.