Unification in $\mathcal{EL}$ for Competency Question Generation

Yuri Malheiros[1] and Fred Freitas[2]

[1] Universidade Federal da Paraíba (UFPB), Rio Tinto - PB, Brazil
yuri@dcx.ufpb.br
[2] Universidade Federal de Pernambuco (UFPE), Recife - PE, Brazil
fred@cin.ufpe.br

Abstract. Competency Questions (CQs) are widely used in ontology development to represent the ontology requirements. Engineers can check if a CQ is satisfied manually or with software assistance. However, when a CQ is not satisfied, they need to analyze the axioms to discover what is missing. This activity may be hard and time-consuming, because of the size of the ontology and the complexity to inspect the axioms using inferences. In this paper, we present a method that uses unification in $\mathcal{EL}$ to generate new CQs based on an unsatisfied CQ. They are used to questioning an engineer, therefore she can provide answers to add the missing knowledge so as to satisfy the initial CQ to the ontology. We did two experiments using the SNOMED CT ontology. Our approach generated questions to add the missing knowledge in 69.09% cases.

Keywords: ontology engineering; competency questions; unification

1 Introduction

For years, ontologies were developed through ad hoc efforts. There was no patterns or methodologies to guide engineers during the process to build an ontology. Thus, each team followed its own rules and set of activities [10] [12].

However, since 1990, consistent methodologies and tools have been proposed to support ontology development. These methodologies address the tasks of creating and maintaining an ontology; thus, they specify an ontology life-cycle, define how to describe the ontology scope and requirements (this latter consisting of the competency questions (CQs) [11]), how to create the ontology specification, how to conduct its evolution, etc. Some well-known methodologies to ontology development are: Methontology [8], On-To-Knowledge [19], Ontology 101 [16], and NeOn [5]. And, some popular tools are: Protégé [9], OntoStudio[3], NeOn Toolkit [5], OntoEdit [20] and WebODE [1].

Competency questions (CQs) are widely used in ontology development to represent the ontology requirements. They are a set of questions that an ontology must answer using the knowledge represented by its axioms. For example, in an

[3] http://www.semafora-systems.com/en/products/ontostudio/

ontology about wine, we may have the CQs [16]: 'Is bordeaux a red or white wine?" or "Does cabernet sauvignon go well with seafood?".

Engineers can check if a CQ is satisfied manually or with software assistance [15]. When a CQ is not satisfied, the engineers usually need to add axioms to satisfy this requirement. This work may be hard and time-consuming, particularly in large ontologies. Furthermore, to create an ontology to model a domain is not a simple task. A team building this kind of artifact needs to have knowledge about logic, ontologies, the right tools, languages, and they need to know about the domain being specified. In this way, tools and methods to aid engineers and domain experts, could make the process of creating an ontology easier and faster.

In this paper, we present a method to help engineers to add axioms to an ontology. Given an unsatisfied CQ, our approach uses unification in DL $\mathcal{EL}$ with acyclic TBoxes [4] to generate a set new competency questions. An engineer should answer one or more generated questions to add new knowledge that is transformed to axioms in the ontology. The goal of the generated questions is to guide an engineer to add axioms to satisfy the initial unsatisfied CQ, thus helping he in the task of building an ontology.

For example, given an ontology with two axioms: $Herbivore \equiv Animal \sqcap \exists eat.Vegetable$ and $Cow \equiv Vertebrate \sqcap \exists eat.Grass$, an engineer could check this CQ: "Is cow a herbivore?" with the expected answer "true". This question can be interpreted as the axiom $Cow \sqsubseteq Herbivore$. With these axioms, a DL reasoner cannot conclude that the CQ is satisfied, thus our approach could suggest some questions, for instance, "Is vertebrate an animal?" and "Is grass a vegetable?". If an engineer answers both questions with a "yes", the axioms $Vertebrate \sqsubseteq Animal$ and $Grass \sqsubseteq Vegetable$ will be added to the ontology. Then, the initial CQ is checked again, and now it is satisfied.

The remainder of this paper is organized as follows: Section 2 provides the background about ontology engineering, competency questions, description logics ontologies, and unification; Section 3 presents our method to generate CQs; Section 4 shows the results of an experiment to evaluate our approach; In Section 5 we discuss the results; Section 6 presents the related work; and, Section 7 concludes the paper and shows some ideas for future works.

2 Background

This section presents concepts that serve as foundation of this work. In the following four sections we explain about ontology engineering, competency questions, descriptions logics ontologies, and unification.

2.1 Ontology Engineering

According to Gómez-Perez and colleagues, ontology engineering refers to the activities related to the process, life-cycle, methods, methodologies, tools, and languages to support the ontology development [10]. Devedzic defines that ontology engineering covers the set of activities done during the conceptualization, design, implementation, and deployment of ontologies [7].

In some ways, the methodologies to develop ontologies are analogous to the ones for software engineering. They provide guidance to developers and are divided in phases, for example, specification, execution, and evaluation. Besides, the process is usually iterative, and the ontology can evolve during its lifetime in a very similar way of a software, in the sense that it requires maintenance, versioning, etc.

2.2 Competency Questions

Competency questions [11] are a set of questions that an ontology must be capable to answer using its axioms. The questions can be used to specify the problems an ontology or a set of ontologies must solve. Thus, they work as requirements specification of one or more ontologies. With a set of CQs at hands, it is possible to know whether an ontology was created correctly, in other words, if it contains all the necessary and sufficient axioms that correctly answer the CQs.

The following list shows some CQs used in an ontology for public employment services [5]:

- What is the job seeker nationality?
- What is the job seeker desired job?
- What is the required work experience for the job offer?
- Is the offered salary given in Euros?

2.3 Description Logics Ontologies

Description Logics (DLs) are a family of knowledge representation formalisms that have been gaining growing interest in the last two decades, particularly after OWL (Web Ontology Language) [17] was approved as the W3C standard for representing the most expressive layer of the Semantic Web.

A DL ontology is a set of axioms a_i defined over the triple (N_C, N_R, N_O) [3], where N_C is the set of concept names or atomic concepts (unary predicate symbols), N_R is the set of role or property names (binary predicate symbols); N_O the set of individual names (constants), instances of N_C and N_R. N_{CO} is the set of classes' instances and N_{RO} the set or roles' instances, with $N_{CO} \cup N_{RO} = N_O$.

There are two axiom types allowed in DL: (i) Assertional axioms, which are concept assertions $C(a)$, or role assertions $r(a, b)$, where $C \in N_C, r \in N_R, a, b \in N_O$ and (ii) Terminological axioms, composed of any finite set of GCIs (general concept inclusion) in one of the forms $C \sqsubseteq D$ or $C \equiv D$, the latter meaning $C \sqsubseteq D$ and $D \sqsubseteq C$, C and D being concepts. An ontology or knowledge base (KB) is referred to as a pair $(\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is the terminological box (or TBox) which stores terminological axioms, and $\mathcal{A}$ is the assertional box (ABox) which stores assertional axioms. $\mathcal{T}$ may contain cycles, in case at least in an axiom of the form $C \sqsubseteq D$, D can be expanded to an expression that contains C. DL semantics is defined through interpretations. An interpretation $\mathcal{I}$ is a non-empty

domain Δ^I and an interpretation function $.^{\mathcal{I}}$, then $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$. Further, $.^{\mathcal{I}}$ maps concept names to subsets of $\Delta^{\mathcal{I}}$ and role names to binary relations over $\Delta^{\mathcal{I}}$ [3].

Distinct DL languages can be defined according the operators they support. In this work, we support $\mathcal{EL}$ ontologies with acyclic TBoxes. This DL language is less expressive than others well know DL languages, for instance, $\mathcal{ALC}$, but it is used in very large ontologies, e.g., SNOMED CT[4]. Also, because its simplicity, the inference of subsumptions is polynomial, in other words, it can be fast enough for real life applications. The Table 1 shows the operators supported by the DL $\mathcal{EL}$ and their semantics.

Table 1. DL $\mathcal{EL}$ operators syntax and semantics

Operators	Syntax	Semantics
Concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Role	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Top	$\top$	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Existential restriction	$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b : (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$
Concept definition	$A \equiv C$	$A^{\mathcal{I}} = C^{\mathcal{I}}$

2.4 Unification

A DL $\mathcal{EL}$ unification [4] problem is defined as a finite set: $\Gamma = \{C_1 \equiv^? D_1, ..., C_n \equiv^? D_n\}$, where $C_1, D_1, ..., C_n, D_n$ are concepts. A solution or unifier of Γ is called σ, that is, a substitution that solves all equations $\sigma(C_i) \equiv \sigma(D_i)$, for $i = 1, ..., n$. To define unification, we divide the set of concept names N_C in two: N_v (concept variables) and N_c (concept constants). The concepts in the former set may be replaced by substitutions, while the latter must not.

For example, given an ontology with two definitions of male sports car enthusiast: $X \equiv Human \sqcap Male \sqcap \exists loves.SportsCar$ and $Y \equiv Man \sqcap \exists loves.(Car \sqcap Fast)$. Although both concepts express the same idea, we have different concepts. Thus, this is a typical case to use unification. Defining $N_v = \{Man, SportsCar\}$ and $\Gamma = \{X \equiv^? Y\}$, the solution are the substitutions $Man \mapsto Human \sqcap Male$ and $SportsCar \mapsto Car \sqcap Fast$.

For acyclic TBoxes, a concept only occurs once as left-hand side, and there is no cyclic dependencies between concept definitions. In this case, the subsumption complexity is polynomial and the unification is NP-complete [4].

3 Generating Competency Questions

Our method generates CQs to guide an engineer to add axioms to satisfy a previously asked unsatisfied CQ. It is necessary that a CQ can be represented

[4] http://www.ihtsdo.org/snomed-ct/

as an axiom, thus we can define the problem as a TBox abduction problem. Different approaches may be used to transform CQ into axioms, for instance the transformations showed in [15].

Given a DL ontology O, a CQ α and $O \cup \{\alpha\}$ consistent. The solution for a TBox abduction problem is a finite set Σ, such that $O \cup \Sigma \models \alpha$ and $O \cup \Sigma \not\models \bot$. In our approach, to find Σ we solve a unification problem. All substitutions found are transformed into equivalence axioms. For instance, $X \mapsto Y$ is transformed into $X \equiv Y$. Thus, Σ is a set of these axioms.

We can convert the axioms of Σ into natural language questions. Thus, an engineer can answer these questions. If she answers positively a generated CQ, then this knowledge can be added to the ontology. In this way, an engineer has an easier and natural way to add axioms to an ontology. Besides, she is guided in this process, because the questions generated are based on the unsatisfied CQ.

There are four steps to generate CQs solving a unification problem. First, we need to define the equation of the unification problem. Second, the variables are defined to specify which concepts may receive a definition. After this, the unification is processed to find the set of substitutions and their corresponding axioms. In the end, the axioms are converted to questions in natural language. The Figure 1 shows how the steps interact with each other.

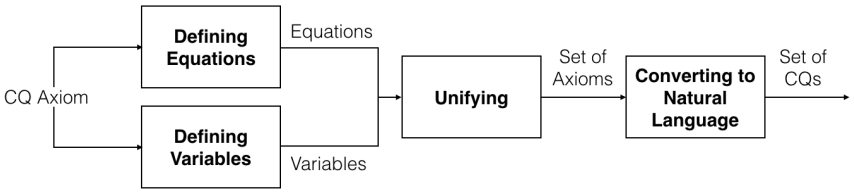

Fig. 1. Process to generate competency questions

3.1 Defining Equations

A unification problem is a set of equations $\Gamma = \{C_1 \equiv^? D_1, ..., C_n \equiv^? D_n\}$. To generate questions based on an unsatisfied CQ, the problem is a set with one element representing the unsatisfied CQ. Given α the unsatisfied CQ axiom, then $\Gamma = \{\alpha\}$. It is important to notice, that a CQ axiom may be a subsumption axiom instead of an equivalence axiom. In this case, we transform $C \sqsubseteq D$ into $C \equiv C \sqcap D$.

3.2 Defining Variables

The variables are the concepts that may receive a definition as the solution of the unification problem. We use the following method to define the variable concepts based on the approach of the UEL library [2].

First, the equation is stored. Next, every axiom that defines the concepts in the equation is stored. Then, all axioms that define the concepts of the concepts stored previously are saved. This process continues recursively until there is no axiom defining the concepts stored. For example, given an equation $X \equiv Y$, and a set of axioms: $X \equiv A$; $Y \equiv B$; $B \equiv C$; $D \equiv E$. First, $X \equiv Y$ is stored. Then, the axioms that define concepts in the equations are stored, in this case, $X \equiv A$ and $Y \equiv B$. Now, recursively, the axioms defining concepts in the previously stored axioms are stored. Thus, $B \equiv C$ is saved. After this, the process ends.

With this set of axioms in hands, the next step is to choose the variable concepts. For this, all concepts in the stored axioms that have no equivalence definition are added to the set of incompletely defined concepts ψ_{N_v}. In the example, we stored $\{X \equiv Y, X \equiv A, Y \equiv B, B \equiv C\}$, then the concepts in the axioms are $\{X, Y, A, B, C\}$. However, among these concepts, only $\{A, C\}$ does not have equivalence definitions, hence $\psi_{N_v} = \{A, C\}$.

Furthermore, in addition to the individual concepts, we also store the two by two combination of the concepts in ψ_{N_v}, thus generating the set of incompletely defined concepts pairs $C\psi_{N_v}$.

3.3 Unifying

In this step we use the equation and variables defined in the previous steps. For each element in ψ_{N_v} and each pair in $C\psi_{N_v}$, we run the unification algorithm of the UEL library [2]. Thus, following the example of the previous section, we try to solve the unification problem three times, the first time using A as variable, the second using C, and the last time using A and C. The result of this step is a set of equivalence axioms that represents the substitutions.

3.4 Converting to Natural Language

The last step transforms the equivalence axioms of the previous step in natural language competency questions.

All axioms have the form $X \equiv C$, such that C is a complex concept. The algorithm to convert to natural language starts adding C to a queue and set a flag with an empty value. Next, the algorithm gets the first element of the queue, and tests if it is a concept, a role, an existential restriction or a conjunction. Based on the element type and the flag value, the algorithm builds the question step by step. If the element is an existential restriction or a conjunction, it is broken in smaller parts that are added to the queue. However, if the item is a concept or a role, parts of the question are generated. The process ends when there is no element in the queue.

The algorithm processes the names of concepts and roles to transform them into a more natural form. The process adds a space before each uppercase letter, except the first, and convert everything to lowercase in the end. For instance, a concept “ProcedureOnSkeletalSystem” is transformed into “procedure on skeletal system”.

The Figure 2 shows all steps of the algorithm to convert axioms into natural language questions. It is important to notice, that when the algorithm reaches a return (the items with quote marks) this text is concatenated with the previously generated texts.

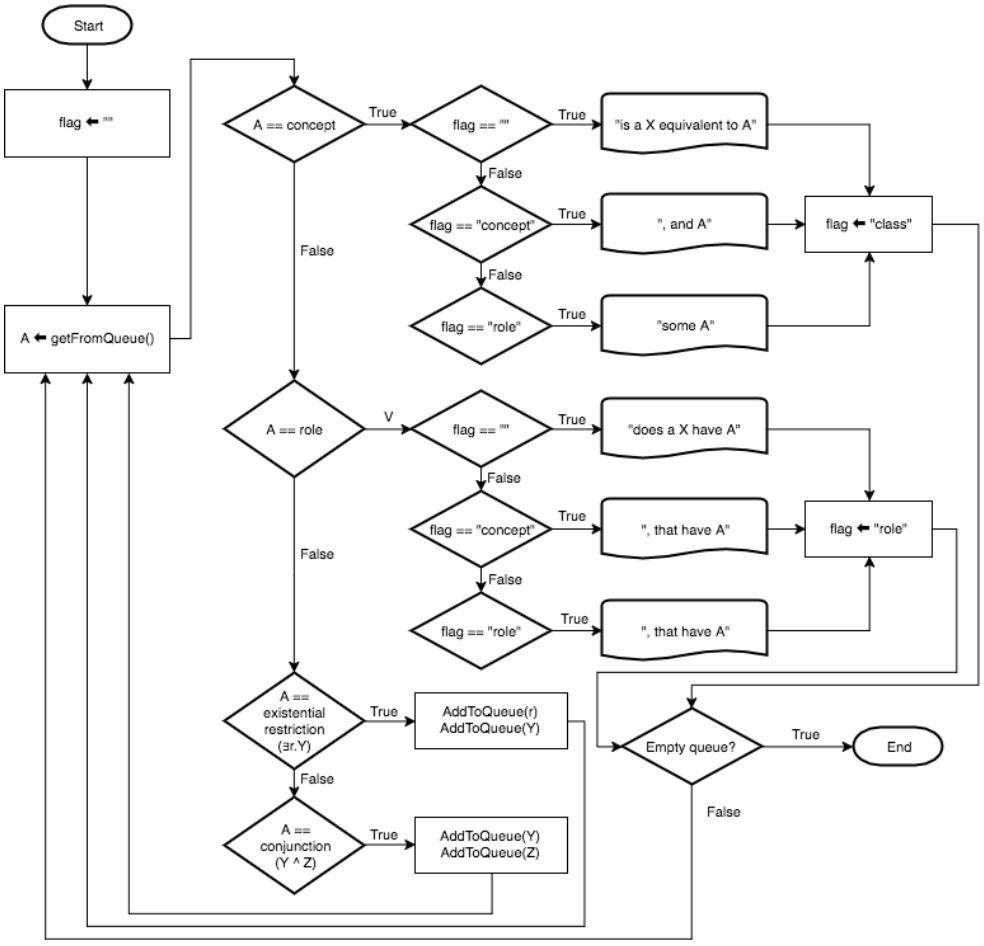

Fig. 2. Natural language conversion algorithm flowchart

3.5 Example

In this section, we present the whole process of generate CQs based on an unsatisfied CQ. Consider an ontology with these axioms:

– $FastEngine \sqsubseteq CarEngine$

- $RaceEngine \sqsubseteq CarEngine$
- $FastCar \sqsubseteq Car \sqcap \exists engine.FastEngine$
- $RaceCar \sqsubseteq Car \sqcap \exists engine.RaceEngine$

Given a CQ "is race car a fast car", or, as an axiom, $RaceCar \sqsubseteq FastCar$. First, we define the unification problem equation, that is $\Gamma = \{RaceCar \sqsubseteq^? FastCar\}$. The second step defines the variables. The algorithm stores the axioms according to the process described in Section 3.2. Next, it chooses only the concepts in the axioms that have no equivalence definitions . In this case, $\psi_{M_v} = \{Car, FastEngine, RaceEngine\}$. Also, it saves the combination pairs of the concepts in ψ_{M_v}, creating the set: $C\psi_{M_v} = \{(Car, FastEngine), (Car, RaceEngine), (FastEngine, RaceEngine)\}$.

In the third step, the unification problem is solved for each element in ψ_{M_v} and each pair in $C\psi_{M_v}$. Using Car as variable the result is $Car \equiv \exists engine.FastEngine$, and using $RaceEngine$ as variable the result is $RaceEngine \equiv FastEngine$. For the other cases, it does not find any unifier.

To conclude the process, both axioms found in the unification step are transformed into natural language questions. The first axiom is transformed into the question "does a car have engine some fast engine?", and the second into the question "is a race engine equivalent to fast engine?".

4 Results

We did two experiments to evaluate our method. The goal of the first was measuring the percentage of cases that the algorithm finds a unifier, so that the system can generate CQs. In the second experiment, we evaluate if the generated competency questions were similar to axioms created by engineers.

The SNOMED CT ontology was used in the experiments. It is the largest ontology about clinical concepts with more than 350,000 concepts and 1.38 millions relations. Many concepts in the ontology use a special role called "roleGroup" to group some existential restrictions. However, this practice can harm the semantics of the axiom, and hinder the translation to natural language. Thus, we removed all roleGroups before the experiments, but we kept the content inside the role. For instance, $X \equiv \exists roleGroup.(\exists r.Y \sqcap s.Z)$ was changed to $X \equiv \exists r.Y \sqcap \exists s.Z$.

4.1 Creating Questions

To test the capability to generate competency questions, we created three modules extracted from the SNOMED CT ontology using the OWL-ME tool [6]. Then, we did not have a too large ontology to test. The tool creates locality-based modules using a set of concepts as input. Thus, for each module, we used an input of 20 randomly chosen concepts. The first module contained 3832 axioms and 775 concepts, the second 4492 axioms and 913 concepts, and the third 4738 axioms e 958 concepts.

Next, we chose 10 random axioms from each module. Further, we made 10 copies of each module, and remove, from each copy, one of the 10 random axioms chosen previously. Thus, we had 30 modules, each with a missing axiom. To ensure that the knowledge represented by the axioms was removed, we tested if the axioms could be inferred. In all cases, they could not.

The next step was creating manually CQs that needed the axioms removed to be satisfied. Thus, we expected that our approach would return questions to guide us to add the axioms to satisfy the unsatisfied CQs.

Table 2. Results for the experiment to create questions experiment

Module	Generated	Did not generate	Total
1	38 (60,32%)	25 (39,68%)	63
2	36 (66,67%)	18 (33,33%)	54
3	53 (80,30%)	13 (19,69%)	66

The Table 2 shows the results of the tests. For the 10 modules created from the first module, we had 63 CQs, and our approach generated questions to help satisfy 38 CQs. In other words, in 38 cases one or more unifiers were found, and in the remainder none were found. For the 10 modules created from the second module, we had 54 CQs, and our approach generated questions for 36 CQs. For the last 10 modules created from the third module, we had 66 CQs, and our approach generated questions for 53 CQs.

4.2 Axioms' Similarity

In this experiment we tested if the axioms generated by the unification are similar to the axioms coded by engineers. To this end, we used the same modules of the first test. We compared if the axioms of the unification problem solution are similar to the axioms removed in each module.

To compare axioms, we used two approaches. The first, for each axiom we created a set of concepts according to them. For example, for an axiom $X \equiv A \sqsubseteq \exists r.B$, the set is $\{X, A, r, B\}$. In the second approach, we also created a set for each axiom, however, we did not dissociate the role from its related concept. For example, given the same axiom $X \equiv A \sqsubseteq \exists r.B$, the set in the second approach is $\{X, A, r.B\}$. Thus, the second approach is more rigorous than the first, because the role and its concept must appear together.

The sets were compared using the following metrics: precision, recall and f-measure. Given $S_s = \{A, B, X, Y\}$, a set generated from a suggested axiom by our approach, and $S_c = \{A, B, C\}$, a set generated from an axiom coded by an engineer. The precision is $2/4$, the recall is $2/3$ and the f-measure is calculated using the equation $F = 2 \times \frac{Precision \cdot Recall}{Precision + Recall}$.

Table 3 shows the results of this experiment. The metric name followed by the number 1 means that it was calculated using the first approach to generate

the sets. The metric name followed by the number 2 means that it used the second approach to generate the sets.

Table 3. Results of the axioms similarity experiment

Module	Precision (1)	Recall (1)	F-measure (1)	Precision (2)	Recall (2)	F-measure (2)
1	56,88%	30,31%	35,09%	35,26%	12,15%	15,81%
2	50,42%	35,09%	36,80%	27,41%	15,48%	16,50%
3	46,69%	33,33%	35,09%	21,86%	11,32%	12,59%

5 Discussion

Our approach can generate different types of CQs. Some are simple, for instance "does a disease of musculoskeletal system have finding site some body system structure?". And, some are complex, for instance, "is a malignant neoplasm of genitourinary organ equivalent to a clinical finding, that have associated morphology some mass, that have finding site some anatomical or acquired body structure, that have finding site some anatomical structure?".

Smaller questions are easier to understand than longer questions, because the latter represent large axioms with many concepts, roles and relationships among them. The unification ensures that the axioms represented by the questions will satisfy an unsatisfied CQ; however an engineer must analyze the content of the questions, to be sure that the knowledge make sense. The approach is not concerned with the ontological engagement.

The algorithm generated CQs to add the missing axioms in 69.09% of the cases, this happened because these are the cases that the unification could be solved. The cases that no CQ was generated were the ones that it was not found any unifier for the equation using the variables defined by our approach.

In the results of the second experiment, precision in average is higher than recall. Our approach usually suggests axioms that have roles and concepts of the original axiom, but, it misses some roles and concepts too. For instance, the ontology have this axiom:

$$
\begin{aligned}
&CongenitalFemaleUrogenitalAnomaly \equiv \\
&Disease \sqcap GenitourinaryCongenitalAnomalies \sqcap \\
&\exists associatedMorphology.CongenitalAnomaly \sqcap \\
&\exists associatedMorphology.DevelopmentalAbnormality \sqcap \\
&\exists findingSite.FemaleGenitourinarySystemStructure \sqcap \\
&\exists occurrence.Congenital
\end{aligned}
$$

and our approach suggests a question represented by this axiom:

$$CongenitalFemaleUrogenitalAnomaly \equiv \exists occurrence.Congenital$$

In this example precision is 100% in both cases to generate the sets, however recall is 22.22% in the first case and 16.67% in the second.

Sometimes, the suggested axiom has a correct role, but the concept is different, or vice-versa. Thus, the values of the metrics in the first case is higher than the second case. In other cases, despite the concepts are different, they subsume one another. For instance, in one of the tests, the original axiom has $findingSite.FaceStructure$ and the suggestion has $FindingSite.HeadStructure$.

6 Related Work

The idea of generating questions when a CQ is not satisfied is inspired in the work of Uschold and Gruninger [21]. They published one of the first methodologies to develop ontologies with four main steps: purpose and scope, building the ontology, evaluation, and documentation, in which they use CQs to define the requirements. They argued that when a CQ is not satisfied, it could be satisfied through other questions. In our approach we implemented this idea to help engineers to add axioms to an ontology.

The method query-the-user [18] [14] introduced a symmetric relationship between users and a logic computer program. In other words both users and the program can make questions and provide answers. In this method, a user provides information during a logic program execution whenever the program asks he. Thus, the query-the-user has many similarities with our method to generate CQs.

7 Conclusions and Future Work

In this paper we presented a method to suggest competency questions to guide engineers to satisfy a previous unsatisfied CQ. The method guide engineers to add axioms to an ontology through the answers to the suggested questions.

Two experiments were performed: the first tested the capability of the approach to suggest CQ. In 69.06% of the cases the method generated questions that represented the necessary axioms to satisfy an CQ. The second experiment evaluated if the axioms suggested were similar to the axioms coded by engineers. We tried two ways to measure the similarity through precision, recall and f-measure. On average, the first way had f-measure 35.66%, and the second way had f-measure 14,97%.

For future work, there is opportunity to test abduction in DL $\mathcal{EL}$ [13] to generate the axioms. Moreover, we need an approach to convert the axioms into more readable question, mainly because the large questions created based on complex axioms. Finally, the whole approach could be extended to support more expressive DL languages.

References

1. Arpírez, J.C., Corcho, O., Fernández-López, M., Gómez-Pérez, A.: Webode: a scalable workbench for ontological engineering. In: Proceedings of the 1st international conference on Knowledge capture. pp. 6–13. K-CAP '01, ACM, New York, NY, USA (2001), http://doi.acm.org/10.1145/500737.500743
2. Baader, F., Borgwardt, S., Mendez, J.A., Morawska, B.: Uel: Unification solver for $\mathcal{EL}$. In: Proc. of the 25th Int. Workshop on Description Logics (DL'12). vol. 846, pp. 26–36. Citeseer (2012)
3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The description logic handbook: theory, implementation, and applications. Cambridge University Press, New York, NY, USA (2003)
4. Baader, F., Morawska, B.: Unification in the description logic $\mathcal{EL}$. In: Rewriting techniques and applications. pp. 350–364. Springer (2009)
5. del Carmen Suárez-Figueroa, M., de Cea, G.A., Buil, C., Dellschaft, K., Fernández-López, M., García, A., Gómez-Pérez, A., Herrero, G., Montiel-Ponsoda, E., Sabou, M., Villazon-Terrazas, B., Yufei, Z.: Neon methodology for building contextualized ontology networks (Feb 2008)
6. Del Vescovo, C., Parsia, B., Sattler, U., Schneider, T.: The modular structure of an ontology: an empirical study. Description Logics 573 (2010)
7. Devedzić, V.: Understanding ontological engineering. Commun. ACM 45(4), 136–144 (Apr 2002), http://doi.acm.org/10.1145/505248.506002
8. Fernandez-Lopez, M., Gomez-Perez, A., Juristo, N.: Methontology: from ontological art towards ontological engineering. In: Proceedings of the AAAI97 Spring Symposium. pp. 33–40. Stanford, USA (March 1997)
9. Gennari, J.H., Musen, M.A., Fergerson, R.W., Grosso, W.E., Crubézy, M., Eriksson, H., Noy, N.F., Tu, S.W.: The evolution of protégé: an environment for knowledge-based systems development. International Journal of Human-computer studies 58(1), 89–123 (2003)
10. Gómez-Pérez, A., Fernández-López, M., Corcho, O.: Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web. Advanced Information and Knowledge Processing, Springer (2004), http://books.google.com.br/books?id=UjS0N1W7GSEC
11. Grüninger, M., Fox, M.S.: Methodology for the design and evaluation of ontologies. In: IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing (1995)
12. Guarino, N., Welty, C.: Evaluating ontological decisions with ontoclean. Communications of the ACM 45(2), 61–65 (2002)
13. Halland, K., Britz, K.: Abox abduction in alc using a dl tableau. In: Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference. pp. 51–58. ACM (2012)
14. Hammond, P., Sergot, M.: Apes: augmented prolog for expert systems. Logic Based Systems Ltd 40 (1984)
15. Malheiros, Y., Freitas, F.: A method to develop description logic ontologies iteratively with automatic requirement traceability. In: Informal Proceedings of the 27th International Workshop on Description Logics, Vienna, Austria, July 17-20, 2014. pp. 646–658 (2014)
16. Noy, N.F., McGuinness, D.L.: Ontology development 101: A guide to creating your first ontology 32(1), 1–25 (2001)
17. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: OWL web ontology language semantics and abstract syntax. W3C recommendation, W3C (February 2004), http://

www.w3.org/TR/2004/REC-owl-semantics-20040210/, published online on February 10th, 2004 at http://www.w3.org/TR/2004/REC-owl-semantics-20040210/
18. Sergot, M.: A query-the-user facility for logic programming. Wiley-Interscience (1987)
19. Staab, S., Studer, R., Schnurr, H.P., Sure, Y.: Knowledge processes and ontologies. IEEE Intelligent Systems 16(1), 26–34 (Jan 2001), http://dx.doi.org/10.1109/5254.912382
20. Sure, Y., Erdmann, M., Angele, J., Staab, S., Studer, R., Wenke, D.: Ontoedit: Collaborative ontology development for the semantic web. In: Proceedings of the First International Semantic Web Conference on The Semantic Web. pp. 221–235. ISWC'02, Springer-Verlag, London, UK, UK (2002), http://dl.acm.org/citation.cfm?id=646996.711413
21. Uschold, M.: Building ontologies: Towards a unified methodology. In: In 16th Annual Conf. of the British Computer Society Specialist Group on Expert Systems. pp. 16–18 (1996)