A Tool for Building Topic-specific Ontologies Using a Knowledge Graph

Katinka Böhm[1] and Magdalena Ortiz[1]

Faculty of Informatics, TU Wien [⋆]
katinka.boehm@gmail.com | ortiz@kr.tuwien.ac.at

Abstract. Nowadays we have very large knowledge graphs that organize common-sense knowledge about the world. They have been successfully deployed in many areas, but the quality and relevance of the contained knowledge varies greatly, and diverse knowledge domains are unpredictably intertwined. This paper summarizes our preliminary efforts at exploiting knowledge graphs to build digestible and reliable Description Logic ontologies for specific common-sense topics. We describe a simple yet effective approach to building small- to middle-sized topic ontologies with moderate efforts. It is implemented in a proof-of-concept tool with a command-line interface for quickly creating axioms by selecting from suggestions extracted from the ConceptNet knowledge graph. To illustrate the kind of ontologies that can be built, we describe a few example ontologies, which contain up to a few hundred concepts and roles, and were built in a couple of hours.

Keywords: topic ontologies · ontology construction · knowledge-graph

1 Introduction

Decades-long research efforts have lead to readily available knowledge graphs containing vast amounts of organized common-sense world knowledge, such as Freebase [2][3], YAGO [17], OpenCyc [16] and DBPedia [12]. They are often the result of integrating different knowledge extraction and data mining techniques, and have been successfully applied in a range of areas. However, little can be assumed about the quality of the knowledge they contain [8][7]. Possibly irrelevant and imprecise facts coexist with relevant knowledge, and different knowledge domains are interrelated in unpredictable ways. Moreover, these knowledge graphs are not ontologies in the strict sense, with a logic-based semantics, and therefore, they cannot be readily exploited for tasks that require non-trivial reasoning. On the other end of the spectrum, we also have many repositories of high-quality ontologies, but these are mostly for highly specialized domains, like life-sciences and health care [11,5], and tailored more for experts than for laymen. Some of them have been manually curated and extended over years, and are far too large to allow a non-expert to get an understanding of the domain they describe.

[⋆] This work was supported by the Austrian Science Fund (FWF) via projects P30360 and P30873."

When it comes to understandable and reliable ontologies of manageable size, describing non-technical every-day topics, there is a big gap. In fact, the only such ontologies that we are aware of are hand-crafted toy examples, like the Manchester Pizza Ontology[1] and the DAML Wine Ontology[2].

In this paper we claim that the vast work invested into knowledge graphs can be leveraged to build, with moderate efforts, small- to middle-sized topic-specific Description Logic (DL) ontologies of every-day domains. We believe such ontologies can be extremely valuable for illustrative and didactical purposes, and in research for quickly building ontologies for testing purposes. They could be further improved with existing ontology-editing tools, and used as an starting point for engineering larger high-quality ontologies for more specialized uses, reducing the time and effort needed to develop them from scratch. We describe a simple yet effective algorithm that allows users to interactively build topic-specific ontologies with moderate efforts, using suggestions retrieved from a knowledge graph. We implemented the algorithm in a tool that queries the ConceptNet2 knowledge graph, and then suggests concepts and axioms to the user, who can create the ontology using a simple command-line interface where he chooses from the current suggestions, and creates axioms using both suggested and possibly self-defined concepts and roles. Our simple proof-of-concept prototype allows to build small- to middle-sized ontologies on everyday topics in a moderate time. The created axioms range from plain concept inclusions, to inclusions with complex $\mathcal{ALCIO}$ concepts. A key challenge is to identify suitable relations in ConceptNet that are likely to encode ontological knowledge relevant to the user, and to establish adequate orderings and thresholds for processing the suggestions in a convenient way that does not overwhelm the user and yields a good trade-off between the time invested and the quality of the resulting ontology. To illustrate the usefulness of our prototype, we describe some illustrative ontologies with a few hundred axioms, created with our tool in a couple of hours.

Related Work. *Ontology engineering* is a vast research field, with goals as varied as developing user-friendly ontology authoring tools, and partly automatizing the ontology design process using machine learning and data mining. Works aimed at *learning ontologies* include DL-FOIL [6] and OCEL [15], which use *inductive logic programming* to learn definitions of concepts from existing ontologies and instance data. A newly published approach employs learning via queries to an oracle [13]. Somewhat related to our work are frameworks for ontology construction like Text2Onto [4] and OntoGen [9], where ontologies are generated semi-automatically with the help of user feedback and interference to improve the results. However, these approaches exploit text corpora and rely on established text processing methods. The creation of knowledge graphs already incorporates this kind of text and data mining, as well as other means of knowledge extraction, so we want to take advantage of such preprocessed and semi-structured knowledge. We are not aware of other attempts aiming at building manageable ontologies for varying topics from existing knowledge graphs.

[1] https://protegewiki.stanford.edu/wiki/Protege4Pizzas10Minutes

[2] https://www.w3.org/TR/2003/PR-owl-guide-20031215/wine

2 Ontologies and Knowledge Graphs

We consider the DL $\mathcal{ALCIO}$. Here we recall its syntax only, and refer to [1] for details. Let N_C, N_R and N_I be countably infinite, pairwise disjoint alphabets of concept names, role names, and individuals. *Roles* are expressions r and r^- with $r \in N_R$, while concepts C obey the grammar $C \to \top \mid \bot \mid A \mid \{a\} \mid \neg C \mid C \sqcap C \mid C \sqcup C \mid \exists r.C \mid \forall r.C$, where $A \in N_C$, $a \in N_I$, and $r \in N_R$. An *ontology* or TBox $\mathcal{T}$ is a set of *general concept inclusions (GCIs)* $C \sqsubseteq D$ where C and D are concepts. We use $N_C(\mathcal{T})$ to denote the concept names occurring in $\mathcal{T}$, and define *taxonomies* as restricted ontologies containing only atomic $GCIs$.

Definition 1 (ACIs, taxonomies). *An* atomic concept inclusion *(ACI) is a GCI of the form $A \sqsubseteq B$ with $A, B \in N_C \setminus \{\top\}$, and a* taxonomy *is a set of ACIs. The taxonomy* $\mathrm{Tax}(\mathcal{T})$ *of an ontology $\mathcal{T}$ is the set of ACIs in $\mathcal{T}$. A taxonomy* Tax *can be represented as a directed graph $\mathcal{G}_{\mathrm{Tax}}$ whose nodes are the concept names in* Tax*, with an edge from A to B whenever $A \sqsubseteq B \in \mathrm{Tax}$.*

We note that, for a taxonomy $\mathcal{T}$, we have $\mathcal{T} \models A \sqsubseteq B$ iff $A \sqsubseteq^*_{\mathcal{T}} B$, where $\sqsubseteq^*_{\mathcal{T}}$ is the reflexive transitive closure of $\sqsubseteq_{\mathcal{T}} = \{(A, B) \mid A \sqsubseteq B \in \mathcal{T}\}$.

We use a generic notion of knowledge graphs formalized as follows:

Definition 2 (Knowledge Graph). *We assume a fixed finite set R of* relation types*. A* knowledge graph *is a triple $K = (G, \ell, \omega)$ where*
- *$G = (N, E)$ is a directed graph whose nodes N are called* entities*,*
- *ℓ is a* labeling *function that assigns to each entity n a string $\ell(n)$, which we call the* label *of n, and to each edge $e \in E$ a relation type $\ell(e) \in R$, and*
- *ω is a* weight *function that assigns a real number to each edge e.*

If there is an edge $e = (n, n') \in E$ with $\ell(e) = r$, we write $e = (n, r, n') \in K$.

ConceptNet. As a knowledge graph, we chose the freely-available ConceptNet 5.5 [18],which originated from the *Open Mind Common Sense Project* [10] of the MIT Media Lab. It contains approximately 28 million edges in over 304 languages and a core set of 37 relation types. A big part of the data was gathered through crowd-sourcing and word games, and also integrates dictionary knowledge from DBPedia, Wiktionary and WordNet. Each node has an ID and a natural language label, which may have articles or filler words. The literature does not give a precise definition of the edge weights, but they are described as an estimation of the trustability of statements, obtained by assessing individual sources, and adding the values when an edge originates from more than one source.

Topic Ontologies. We define a *topic ontology* as a DL ontology with a distinguished *topic concept*. We connect topic ontologies to a knowledge graph K via a mapping ϕ from concept names in $\mathcal{T}$ to entities in K.

Definition 3 (Topic Ontology, KG-mapping). *A* topic ontology *is a pair $\mathcal{O} = (\mathcal{T}, tc)$ of an $\mathcal{ALCIO}$ ontology $\mathcal{T}$ and a concept name tc occurring in $\mathcal{T}$, which we call the* topic concept *of $\mathcal{O}$. Given a knowledge graph K, a K-mapping is a bijective mapping from a subset of $N_C(\mathcal{T})$ to a set of entities of K. We denote $\mathcal{X} = (\mathcal{T}, tc, \phi)$ a topic ontology with a K-mapping ϕ.*

Extracting Suggestions from the Knowledge Graph. In what follows, we assume a fixed knowledge graph K, and use it to build $\mathcal{X} = (\mathcal{T}, tc, \phi)$.

We use ϕ to pose queries to K. Each query returns a set $Req_t(n, r)$ for an entity n and a relation type r in K. Through $t \in \{s, e, b\}$ we differentiate between three types of such sets: *end* (e), *start* (s) and *both* (b). Req_e contains all pairs $(n', \omega(e))$, where there exists an outgoing edge from n to n' that is labeled with r in K. Similarly, Req_s returns all node-weight pairs for any incoming edges from n' to n labeled with r. In the case of Req_b, the union of both Req_e and Req_s is returned. The answers are used to generate *suggestions* for concept names that the user can rely on and use to iteratively add axioms to $\mathcal{T}$.

To combine query results over edges with different relation types and improve the generated suggestions, we define a *filtered request* $\widetilde{Req}(n, \mathcal{R}e\ell)$ for an entity n, which takes the union over multiple (r, t) of a relation r from k and a request type t, and filters them according to a threshold function Θ:

$$\widetilde{Req}(n, \mathcal{R}e\ell) = \bigcup_{(r,t)\in\mathcal{R}e\ell} \{(n', \omega(e)) \in Req_t(n, r) \mid \omega(e) \geq \max_{(r,t)\in\mathcal{R}e\ell} \Theta(Req_t(n, r))\}$$

For example, if we refer to the node n through its label $\ell(n) = $ '*animal*' and request a set of start entities Req_s('animal', IsA) $= \{$('a rabbit', 2.828), ('A beaver', 2.0), ('a centipede', 1.0) , ('a dolphin', 4.0), ('A primate', 2.0), $\ldots\}$, then assuming a threshold function $\Theta(Req_s$('animal', $IsA)) = 2.0$, the filtered request set looks as follows: $\widetilde{Req}$('animal', $\{(IsA, s)\}) = \{$('a rabbit', 2.828), ('A beaver', 2.0), ('a dolphin', 4.0), ('A primate', 2.0), $\ldots\}$

Taking the results from a filtered request, we apply a naming function *onto* that transforms the labels of entities in K into a set $\mathcal{S}(n, \mathcal{R}e\ell)$ of concept names. Formally, we let

$$\mathcal{S}(n, \mathcal{R}e\ell) = \{onto(n') \mid (n', \omega(e)) \in \widetilde{Req}(n, \mathcal{R}e\ell)\}$$

which results in $\mathcal{S}$('animal', $\{(IsA, s)\}) = \{$ Beaver, Dolphin, Primate, Rabbit, $\ldots\}$.

For our implementation we manually selected sets $\mathcal{R}e\ell$ of ConceptNet relation types to produce suggestions, based on different themes that we considered of interest for general knowledge domains. An overview can be found in Table 1, and the specific use of these relations is explained in Section 3.

As a threshold function Θ we chose the following $\hat{\Theta}$:

$$\hat{\Theta}(Req_t(n, r)) = \begin{cases} 2.0 & if\ |\{(n', \omega(e)) \in Req_t(n, r) \mid \omega(e) \geq 2.0\}| \ \geq 10 \\ 1.1 & if\ |\{(n', \omega(e)) \in Req_t(n, r) \mid \omega(e) \geq 1.1\}| \ \geq 5 \text{ and} \\ & \quad\ |\{(n', \omega(e)) \in Req_t(n, r) \mid \omega(e) \geq 2.0\}| \ < 10 \\ 1.0 & else \end{cases}$$

This function favors results with higher confidence than the baseline of 1, trying to keep at least five entities if possible, and it keeps only the suggestions with very high confidence (≥ 2.0) if there are sufficiently many of them (≥ 10).

For *onto*, we chose a function that applies CamelCase to the labels, after removing auxiliary words such as articles and conjunctions. As an example, $\ell(n) =$ "A romance novel" from K is converted to $onto(n) =$ RomanceNovel.

3 Building a Topic Ontology

In this section we describe our interactive algorithm to build $\mathcal{X} = (\mathcal{T}, tc, \phi)$. The user inputs a topic concept tc associated to some $\phi(tc)$ in K, and $tc \sqsubseteq \top$ is initially the only axiom in $\mathcal{T}$. Then $\mathcal{T}$ is expanded with axioms given by the user in three steps: (1) a core taxonomy is created, (2) axioms with possibly complex concepts are created, and (3) $\mathcal{T}$ is refined by adding and removing inclusions.

Table 1. Relations from ConceptNet and request types used

ACIs (step 1)	$\mathcal{R}e\ell_{\sqsubseteq} = \{(\mathit{IsA}, s)\}, \mathcal{R}e\ell_{\sqsupseteq} = \{(\mathit{IsA}, e)\}, \mathcal{R}e\ell_{\cong} = \{(\mathit{RelatedTo}, b)\}$
GCIs (step 2)	$\mathcal{R}e\ell_{part} = \{(\mathit{HasA}, e), (\mathit{PartOf}, s)\}$, $\mathcal{R}e\ell_{part^-} = \{(\mathit{HasA}, s), (\mathit{PartOf}, e)\}$
	$\mathcal{R}e\ell_{loc} = \{(\mathit{AtLocation}, s)\}$, $\mathcal{R}e\ell_{loc^-} = \{(\mathit{AtLocation}, e)\}$
	$\mathcal{R}e\ell_{cap} = \{(\mathit{CapableOf}, e), (\mathit{ReceivesAction}, e)\}$
	$\mathcal{R}e\ell_{cap^-} = \{(\mathit{CapableOf}, s), (\mathit{ReceivesAction}, s)\}$
	$\mathcal{R}e\ell_{sp} = \{(\mathit{HasProperty}, b), (\mathit{RelatedTo}, b)\}$

Step 1: Building the Central Taxonomy

We first build a set $\mathcal{CT}$ of ACIs which we call the *central taxonomy* of $\mathcal{X}$. This $\mathcal{CT}$ is such that (i) $tc \in N_C(\mathcal{CT})$, (ii) $\mathcal{G}_{\mathcal{CT}}$ is connected, and (iii) $\phi(A)$ is defined for every $A \in N_C(\mathcal{CT})$. To generate suggestions for $\mathcal{CT}$, we use the ConceptNet relations *IsA* and *RelatedTo*. More specifically, we use $\mathcal{R}e\ell_{\sqsubseteq}, \mathcal{R}e\ell_{\sqsupseteq}$ and $\mathcal{R}e\ell_{\cong}$ in Table 1. We build $\mathcal{CT}$ as follows:

(a) The suggestions in $\mathcal{S}(\phi(tc), \mathcal{R}e\ell_{\sqsubseteq})$ are prompted to the user. For each selected $A = onto(n) \in \mathcal{S}(\phi(tc), \mathcal{R}e\ell_{\sqsubseteq})$, an axiom $A \sqsubseteq tc$ is added to $\mathcal{T}$, and ϕ is extended with $\phi(A) = n$.
(b) Analogously, suggestions in $\mathcal{S}(\phi(tc), \mathcal{R}e\ell_{\sqsupseteq})$ are prompted, axioms $tc \sqsubseteq A$ are added for the selected $A = onto(n)$, and ϕ is extended with $\phi(A) = n$.
(c) Next, suggestions in $\mathcal{S}(\phi(tc), \mathcal{R}e\ell_{\cong})$ are prompted and the user makes two selections: those to treat as in in (a), and those to treat as in (b).
(d) For each $A \in N_C(\mathcal{CT})$ introduced in items (a) to (c), items (a) and (b) are repeated, adding axioms $A \sqsubseteq B$ and $B \sqsubseteq A$, and extending ϕ accordingly.
(e) Lastly, the algorithm iterates over $N_C(\mathcal{CT})$ and collects for each A, a set S_A of suggestions for possibly missing inclusions. S_A contains each B such that $A \sqsubseteq^*_{\mathcal{T}} B$ and $A \sqsubseteq^*_{\mathcal{T}} B'$ for some B', but neither $B \sqsubseteq^*_{\mathcal{T}} B'$ nor $B' \sqsubseteq^*_{\mathcal{T}} B$. For each pair $(B, B') \in S_A \times S_A$ selected by the user, $B \sqsubseteq B'$ is added to $\mathcal{CT}$.

Fig. 1 illustrates items (a) to (c) for the topic concept Animal.

Step 2: Adding complex GCIs

GCIs with possibly complex concepts are added next in four analogous substeps. The suggestions presented in each substep are based on a certain theme. The first three focus on popular relations in ConceptNet likely to be relevant in several

Fig. 1. Building a central taxonomy for the topic concept Animal.

Example Input:

```
Possible subclasses of Animal
Beaver[0], Ferret[1], Primate[2], Rabbit[3], Rodent[4]
Which concepts do you want to delete? 4 2
Possible superclasses of Animal
LivingCreature[0], Organism[1], Preditor[2]
Which concepts do you want to delete? not 0
Those might be sub- or super-classes too:
Bear[0], Beast[1], Fox[2], LivingBeing[3], Skunk[4], Squirrel[5]
Choose subclasses of Animal: 0 2 4 5
```

Result: $\mathcal{CT} = \{\mathsf{Beaver} \sqsubseteq \mathsf{Animal}, \mathsf{Ferret} \sqsubseteq \mathsf{Animal}, \mathsf{Rabbit} \sqsubseteq \mathsf{Animal}, \mathsf{Animal} \sqsubseteq \mathsf{LivingCreature}, \mathsf{Bear} \sqsubseteq \mathsf{Animal}, \mathsf{Fox} \sqsubseteq \mathsf{Animal}, \mathsf{Skunk} \sqsubseteq \mathsf{Animal}, \mathsf{Squirrel} \sqsubseteq \mathsf{Animal}\}$

domains: part-of relations, spatial locations, and capabilities (see Table 1). The last substep uses suggestions from the generic ConceptNet relations *HasProperty* and *RelatedTo* and creates axioms with topic-specific role names.

The procedure is then similar for each substep. An algorithm iterates over the concept names in $N_C(\mathcal{CT})$ using a queue $Q_{\mathcal{CT}} = (A_1, \ldots, A_n)$ where $i < j$ whenever $A_j \sqsubseteq^*_{\mathcal{T}} A_i$, to fix one concept name A from $N_C(\mathcal{CT})$. Respective suggestions are proposed for A and the user can select and create associated axioms.

Step C1: has-part and part-of. The user is asked whether he wants to create HasPart and PartOf relations. If the answer is positive, then for each $A \in N_C(\mathcal{CT})$ the two sets $\mathcal{S}(\phi(A), \mathcal{R}e\ell_{part})$ and $\mathcal{S}(\phi(A), \mathcal{R}e\ell_{part^-})$ of suggestions are retrieved. For each $A \in N_C(\mathcal{CT})$, we define the following combined set of suggestions

$$\mathcal{S}_A := \bigcup_{\phi(B)=n' \ \ \mathrm{Tax} \models B \sqsubseteq A} \mathcal{S}(n', \mathcal{R}e\ell_{part}). \tag{1}$$

Using this combined set may result in a large number of suggestions for very general concepts, but later redundant suggestions for more specific concepts (which inherit already created GCIs) are removed.

The system asks if the user wants suggestions s of the form A has part s. If the answer is yes, then the algorithm iterates over the queue $Q_{\mathcal{CT}}$, skipping those A for which $\mathcal{S}_A$ is empty, and in each iteration proceeds as follows:

1. It asks if the user wants suggestions of parts of A. If the answer is no, then it moves to the next concept in the queue.
2. If the answer is yes, then the user is prompted with the suggestions in $\mathcal{S}_A$. He can select from the list, or type fresh concept names, and the axiom
$$A \sqsubseteq \exists \mathsf{HasPart}.(B^1_1 \sqcup \cdots \sqcup B^1_{n_1})$$
is added, where $(B^1_1, \ldots, B^1_n)$ is the list of the selected and typed concept names. This step can be repeated to create as many analogous axioms as desired, with different disjunctions $(B^i_1 \sqcup \cdots \sqcup B^i_{n_i})$.
3. The user is given the option of adding an axiom $A \sqsubseteq \forall \mathsf{HasPart}.A\mathsf{Part}$ with $A\mathsf{Part}$ a fresh concept name.

4. The user is offered the option of jumping to step *C4* (domain-specific roles) with the current suggestions. If he does, he will then come back to this step.

After the whole queue has been processed, the user is asked if he wants suggestions s of the form s has part A. If the answer is yes, then the role inclusion PartOf^- is defined as the inverse of $\mathsf{HasPart}$, and the process is repeated, using the role PartOf and the following combined set of suggestions for each $A \in N_c(\mathcal{CT})$:

$$\mathcal{S}'_{\mathcal{A}} := \bigcup_{\phi(B)=n',\ \mathrm{Tax}\models B \sqsubseteq A} \mathcal{S}(n', \mathcal{R}e\ell_{part^-}). \tag{2}$$

Example 1 in Fig. 2 shows the creation of an axiom with $\mathsf{HasPart}$.

Fig. 2. Examples of input and resulting axioms during Step 2.

Example 1
```
1 MotorVehicle.HasPart.Engine Motor
```
$\mathsf{MotorVehicle} \sqsubseteq \exists\ \mathsf{HasPart}.(\mathsf{Engine} \sqcup \mathsf{Motor})$

Example 2
```
1 relation name = HasIngredient Contains
2 Cake.HasIngredient.Sugar
3 Cake.Contains.SugarSyrup ArtificialSweetener
```
$\mathsf{Cake} \sqsubseteq \exists\ \mathsf{HasIngredient}.\mathsf{Sugar} \sqcup \exists\ \mathsf{Contains}.(\mathsf{SugarSyrup} \sqcup \mathsf{ArtificialSweetener})$

Example 3
```
1 relation name = InStorage InTemporalStorage
2 LibraryBook.InStorage.Shelf: Bookcase()
3 Enter subclasses for Shelf:Bookshelf
4 Enter instances for Bookcase:bc1 bc2 bc3 bc4 bc5
5 LibraryBook.InTemporalStorage.Table:
6 Enter subclasses for Table:PresentationTable RollingTable
```
$\{\mathsf{bc1},\mathsf{bc2},\mathsf{bc3},\mathsf{bc4},\mathsf{bc5}\} \sqsubseteq \mathsf{Bookcase}$,
$\mathsf{LibraryBook} \sqsubseteq \exists\ \mathsf{InStorage}.(\mathsf{Shelf} \sqcup \mathsf{Bookcase}) \sqcup \exists\ \mathsf{InTemporalStorage}.(\mathsf{Table})$,
$\mathsf{Bookshelf} \sqsubseteq \mathsf{Shelf}$, $\mathsf{PresentationTable} \sqsubseteq \mathsf{Table}$, $\mathsf{RollingTable} \sqsubseteq \mathsf{Table}$

Example 4
```
1 relation name: HasTopping
2 Pizza.HasTopping.PizzaTopping:
3 Enter subclasses for PizzaTopping: Pepperoni Olive Onion
4 Create an existential axiom for Pizza? [y/n] y
5 Pizza.HasTopping.PizzaTopping
```
$\mathsf{Pizza} \sqsubseteq \exists\ \mathsf{HasTopping}.\mathsf{PizzaTopping}$,
$\mathsf{Pepperoni} \sqsubseteq \mathsf{PizzaTopping}$, $\mathsf{Olive} \sqsubseteq \mathsf{PizzaTopping}$, $\mathsf{Onion} \sqsubseteq \mathsf{PizzaTopping}$,
$\mathsf{Pepperoni} \sqcap \mathsf{Olive} \sqsubseteq \bot$, $\mathsf{Olive} \sqcap \mathsf{Onion} \sqsubseteq \bot$, $\mathsf{Onion} \sqcap \mathsf{Pepperoni} \sqsubseteq \bot$,
$\exists\ \mathsf{HasTopping}^-.\top \sqsubseteq \mathsf{PizzaTopping}$

Step C2 / Step C3: locations / capabilities. In these two identical steps, the user is asked whether he wants to get location/capability related suggestions. If yes, the algorithm iterates over $Q_{\mathcal{CT}}$, and accordingly proposes the suggestions in $\mathcal{S}(\phi(A), \mathcal{R}e\ell_{loc})$, $\mathcal{S}(\phi(A), \mathcal{R}e\ell_{loc^-})$, $\mathcal{S}(\phi(A), \mathcal{R}e\ell_{cap})$, or $\mathcal{S}(\phi(A), \mathcal{R}e\ell_{cap^-})$. The addition of axioms is then as described below in step *C4* (topic-specific roles).

Step C4: topic-specific roles. The queue $Q_{\mathcal{CT}}$ is iterated and for each A the user is optionally prompted with suggestions $\mathcal{S}(\phi(A), \mathcal{R}e\ell_{sp})$. Then role and concept names can be inserted and used to create GCIs as follows:

1. The user is asked if he wants to add a new axiom containing A. If not, the algorithm moves to the next item in the queue.

2. If yes, the user is prompted to enter a set of relation names $(R^i_1, \dots, R^i_m)$, which may include IsA. Each such set creates one disjunctive axiom that contains exactly the role names in it. Once entered, role names are saved and can be chosen through the auto-complete function for future prompts; a special role can be used to retain suggestions for later.
3. Following, m three-part prompts are issued, one for each R^i_j, $1 \leq j \leq m$, in which the user can select concept names or individuals with auto-complete, or type fresh names. The list for selection contains the suggestions $\mathcal{S}(\phi(A), \mathcal{R}e\ell_{sp})$, concept names that were retained, and all individuals in $N_I(\mathcal{T})$ (see *dynamic extensions* to add individuals). If a fresh string is entered, it is treated as a new concept name and added to $N_C(\mathcal{T})$. If an individual a is selected it is treated as a nominal $\{a\}$. Let $(B^i_{j_1}, \dots, B^i_{j_n})$ be the entered concept names or nominals for R^i_j. Example 2 in Fig. 2 illustrates how an input $A.R^i_j.B^i_{j_1} \dots B^i_{j_n}$ (line 3) is converted into $\exists R^i_j.(B^i_{j_1} \sqcup \cdots \sqcup B^i_{j_n})$. If $R^i_j = \mathsf{IsA}$, then the input is converted into $\bigsqcup_k B^i_k$, without an existential. After all m prompts are complete, they are combined into the final axiom

$$\beta_i = A \sqsubseteq \textstyle\bigsqcup_k B^i_k \sqcup \bigsqcup_j \exists R^i_j.(B^i_{j_1} \sqcup \cdots \sqcup B^i_{j_n})$$

Steps 1-3 can be repeated as often as wanted for each A to create multiple β_i. Note that if $k = 1$ and $j = 0$, then β_i has the form $A \sqsubseteq B$.

When entering $(B^i_{j_1}, \dots, B^i_{j_n})$ the user can add one of two suffixes to any B_{j_k} to receive further prompts. We call this process *dynamic extension.*

Appending a colon (:) allows for the creation of a set of subclasses $(A_1, \dots, A_m)$ together with respective axioms $A_i \sqsubseteq B_{j_k}, 1 \leq i \leq m$. Similarly with a set of brackets (()) individuals $(a_1 \cdots a_m)$ are created together with axioms $\{a_i\} \sqsubseteq B_{j_k}$. This is especially useful when suggestions contain concepts that ought to be added to the ontology, but not in the scope of A. More general concept names can then be chosen in β_i and subclasses or individuals assigned through dynamic extensions. If $k = 0$, $j = 1$, and $n = 1$ in β_i, a *range restriction* (3) for the chosen R^i_j is also created. If a colon is used, then additionally a set of *disjointness axioms* (4) is created for all A^i_j and A^i_k that were added as subclasses.

$$\exists R^{i-}_j.\top \sqsubseteq B^i_{j_k} \quad (3) \qquad\qquad A^i_j \sqcap A^i_k \sqsubseteq \bot \quad (4)$$

We can create axioms $B \sqsubseteq \exists R^i_j.A$ by writing R^i_j with suffix '$-$' and having $j = 1$. The successive prompt changes to specify the left- instead of the right-hand-side of the axiom; for this case dynamic extensions are not yet supported.

Examples 3 and 4 in Fig. 2 show the addition of an axiom β_i with $j = 2$ using dynamic extensions, and an axiom with range restriction and disjointness.

Step 3: Clean-Up

The last step offers the option to further modify the taxonomy of $\mathcal{T}$. In particular, the user can create inclusions between concept names added in Step 2, and introduce new concept names into the taxonomy. Initially the concept $A = \top$ is selected and $\mathcal{P}_\top = \{A \mid \text{ there is no } B \text{ s.t. } A \sqsubseteq B \in \mathcal{T}\}$. At each further step, the set $\mathcal{P}_A = \{B \mid B \sqsubseteq A \in \mathcal{T}\}$ is shown, and the user can choose between:

1. Get a new set $\mathcal{P}_B$ for some selected $B \in \mathcal{P}_A$ with $\mathcal{P}_B \neq \emptyset$, or
2. Input two sets $\{A_1, \cdots, A_m\}$ and $\{B_1, \cdots, B_n\}$ by either selecting from $\mathcal{P}_A$ or entering new concept names. Then $n \times m$ axioms $A_i \sqsubseteq B_j$ for $1 \leq i \leq m$ and $1 \leq j \leq n$ are added to $\mathcal{T}$, and additionally
   – if B_j is new, then $B_j \sqsubseteq A$ is added and $A_i \sqsubseteq A$ removed,
   – if $B_j \in N_C(\mathcal{T})$ and $B_j \sqsubseteq A \notin \mathcal{T}$ then the user can choose to add it.

   Note that the removed axioms are redundant, as they are implied by $\mathcal{T}$.

All substeps of the construction process can be started separately and intermediate results are continuously saved.

4 Example Ontologies

We implemented our approach in *CN2TopicOnto* using Python3, and the ConceptNet 5.6 REST API. To create and load OWL 2.0 files, we use the ontology-oriented programming module owlready2 [14].

To illustrate the usefulness of *CN2TopicOnto* we provide examples of topic ontologies built with it. Table 2 shows some parameters of selected topic ontologies. The last column gives a rough estimate of the total user time taken to build each ontology from scratch. This includes suggestion selection and axiom input.

We also show selected parts for two created ontologies, one with $tc = \mathsf{Animal}$ and the other with $tc = \mathsf{Vehicle}$. Fig. 3 and 5 show subgraphs of their $\mathcal{CT}$, where an `is-a`-arc between two concepts A and B translates to $A \sqsubseteq B$. Small arrowheads indicate that there exist further arcs that were omitted for space reasons. Fig. 4 and 6 provide a selection of axioms from the respective $\mathcal{T}$, where the concept name at the top is the relevant $A \in Q_{\mathcal{CT}}$.

Topic concept	$\lvert N_C(\mathcal{T})\rvert$	$\lvert N_R(\mathcal{T})\rvert$	$\lvert N_I(\mathcal{T})\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\mathsf{Dom}(\phi)\rvert$	complex GCIs	Time
Animal	670	67	50	1249	347	33.4 %	6h
Vehicle	300	34	7	483	128	37.7 %	5h
NaturalDisaster	89	16	4	150	35	44.7 %	1.5h
Fruit	309	35	0	564	125	44 %	5h
FastFood	109	20	0	200	22	77.5 %	2.5h
University	153	15	5	233	46	22.3 %	3h

Table 2. Statistics for some constructed ontologies. $Dom(\phi)$ denotes the domain of ϕ.

5 Discussion and Future Work

We have presented an algorithm and a prototype tool *CN2TopicOnto* for constructing ontologies with the help of existing knowledge graphs. Knowledge elicitation and ontology authoring require an immense amount of time and expertise to produce good results, but with our approach we want to simplify the process and exploit the vast amount of existing knowledge repositories to our advantage.

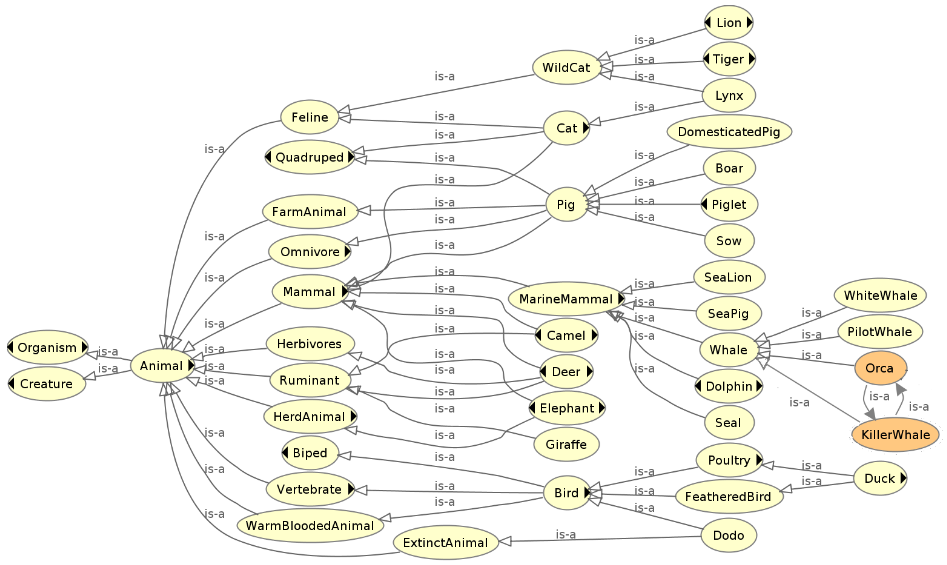

Fig. 3. A selected part of $\mathcal{CT}$ of the Animal taxonomy.

Animal
$\text{Animal} \sqsubseteq \text{Creature},$
$\text{Animal} \sqsubseteq \forall\ \text{HasPart.AnimalPart},$
$\text{Animal} \sqsubseteq \exists\ \text{HasPart.Cell},$
$\text{Animal} \sqsubseteq \exists\ \text{PartOf.Ecosystem},$
$\text{Animal} \sqsubseteq \exists\ \text{PartOf.Nature},$
$\text{Animal} \sqsubseteq \exists\ \text{HasSpecies.AnimalSpecies},$
$\text{Animal} \sqsubseteq \exists\ \text{LiveIn.}(\text{Home} \sqcup \text{Wilderness} \sqcup \text{Zoo})$

Herbivore
$\text{Herbivore} \sqsubseteq \text{Animal},$
$\text{Deer} \sqsubseteq \text{Herbivore},$
$\text{Herbivore} \sqsubseteq \exists\ \text{HasPreferredFood.Plant},$
$\text{Meadow} \sqsubseteq \exists\ \text{NaturalHabitatOf.Herbivore}$

Feather
$\text{Down} \sqsubseteq \text{Feather}, \text{GooseQuill} \sqsubseteq \text{Feather},$
$\text{DuckDown} \sqsubseteq \text{Down}, \text{GooseDown} \sqsubseteq \text{Down},$
$\text{SwanDown} \sqsubseteq \text{Down},$
$\text{Bird} \sqsubseteq \exists\ \text{Possess.Feather}$

Pig
$\text{Pig} \sqsubseteq \text{Animal}, \text{Pig} \sqsubseteq \text{FarmAnimal},$
$\text{Pig} \sqsubseteq \text{Mammal}, \text{Pig} \sqsubseteq \text{Omnivore},$
$\text{Pig} \sqsubseteq \text{Quadruped},$
$\text{Sow} \sqsubseteq \text{Pig}, \text{Piglet} \sqsubseteq \text{Pig}, \text{Boar} \sqsubseteq \text{Pig},$
$\text{DomesticatedPig} \sqsubseteq \text{Pig},$
$\text{Pig} \sqsubseteq \exists\ \text{HasBodyPart.Snout},$
$\text{Pig} \sqsubseteq \exists\ \text{HasCharAnatomicalFeature.WiggleTail},$
$\text{Pig} \sqsubseteq \exists\ \text{HasTypicalColor.}(\text{Brown} \sqcup \text{Pink}),$
$\text{Pig} \sqsubseteq \exists\ \text{LiveIn.Mud},$
$\text{Pig} \sqsubseteq \exists\ \text{HasBehaviourChar.}\{\text{smart}\},$
$\text{Pig} \sqsubseteq \exists\ \text{HasSenseOfSight.}\{\text{good}\},$
$\text{Pig} \sqsubseteq \exists\ \text{MakeAnimalSound.}\{\text{oink}\},$
$\text{Pigsty} \sqsubseteq \exists\ \text{UsedToHold.Pig},$
$\text{Pork} \sqsubseteq \exists\ \text{MeatOf.Pig}$

Dodo
$\text{Dodo} \sqsubseteq \text{Bird}, \text{Dodo} \sqsubseteq \text{ExtinctAnimal}$
$\text{Dodo} \sqsubseteq \exists\ \text{FoundInGeographLocation.Mauritius}$

Fig. 4. Selected axioms from the Animal ontology.

From an ontology engineering perspective, the approach combines top-down and bottom-up design. The topic concept and central taxonomy initialize a top-down process, where concepts are added from general to more specific. Through dynamic extensions it is possible to add new subsuming concepts for subsets of suggestions in a bottom-up fashion.

The ontologies created with our tool, despite being rather simple, can be used as a basis to be developed further into high-quality ontologies that can used in

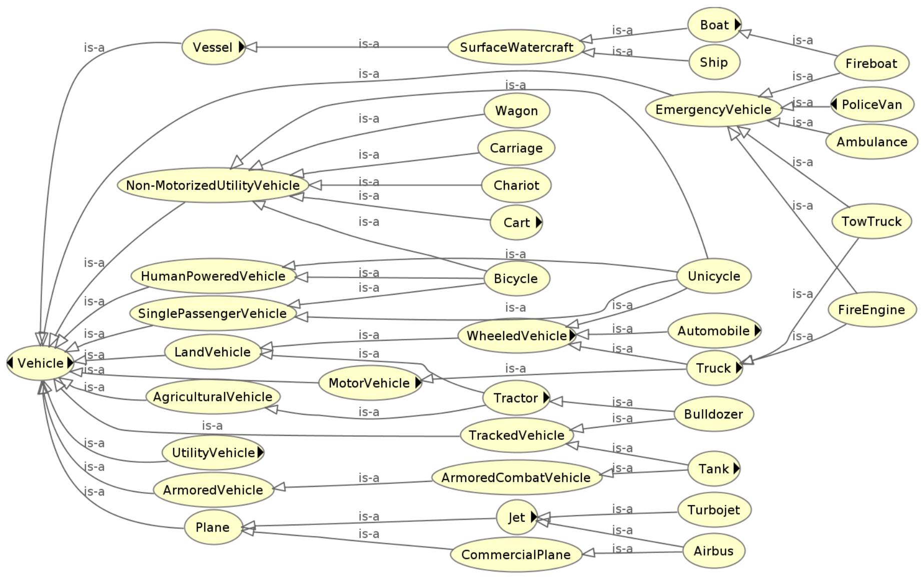

Fig. 5. A selected part of $\mathcal{CT}$ the Vehicle taxonomy.

FireEngine
FireEngine $\sqsubseteq$ EmergencyVehicle,
FireEngine $\sqsubseteq$ Truck,
FireEngine $\sqsubseteq \exists$ EquippedWith.Hose,
FireEngine $\sqsubseteq \exists$ EquippedWith.WaterEngine,
FireEngine $\sqsubseteq \exists$ HasColor.Red,
FireEngine $\sqsubseteq \exists$ WorkVehicleOf.Firefighter,
FireStation $\sqsubseteq \exists$ UsedForPark.FireEngine

SurfaceWatercraft
Boat $\sqsubseteq$ SurfaceWatercraft,
Ship $\sqsubseteq$ SurfaceWatercraft,
SurfaceWatercraft $\sqsubseteq$ Vessel,
SurfaceWatercraft $\sqsubseteq$ Vehicle,
SurfaceWatercraft $\sqsubseteq \exists$ HasPart.(Propellor $\sqcup$ Rudder $\sqcup$ Sail)

Plane
Plane $\sqsubseteq$ Vehicle,
CommercialPlane $\sqsubseteq$ Plane, Jet $\sqsubseteq$ Plane,
Plane $\sqsubseteq \exists$ HasPart.Wing,
Plane $\sqsubseteq \exists$ HasPart.Cabin,
Plane $\sqsubseteq \exists$ EquippedWith.AirplaneSeat,
Plane $\sqsubseteq \exists$ EquippedWith.Lavatory,
Plane $\sqsubseteq \exists$ EquippedWith.OverheadBin,
Plane $\sqsubseteq \exists$ EquippedWith.EmergencyOxygenMask,
Plane $\sqsubseteq \exists$ HasModeOfTransportation.Air,
Plane $\sqsubseteq \exists$ UsedToTransport.(Cargo $\sqcup$ Luggage $\sqcup$ Person),
Plane $\sqsubseteq \exists$ ArriveAtTime.(Late $\sqcup$ Punctual),
Plane $\sqsubseteq \exists$ WorkEnvironmentOf.FlightAttendant,
Plane $\sqsubseteq \exists$ WorkEnvironmentOf.AirHostess,
Plane $\sqsubseteq \exists$ WorkEnvironmentOf.Pilot,
Runway $\sqsubseteq \exists$ LandingAreaFor.Plane,
Taxiway $\sqsubseteq \exists$ DrivingAreaFor.Plane

Fig. 6. Selected axioms from the Vehicle ontology.

research, and in applications that call for non-trivial reasoning services using off-the-shelf DL algorithms.

Further empirical evaluation is needed to asses the quality of the created ontologies and the effectiveness of our tool. We want to expand the supported constructors to include negation and full conjunction. Moving to more expressive DL such as $\mathcal{SHOIQ}$ would be useful to express common-sense statements like Biped $\sqsubseteq =$ 2HasFoot.Foot or FourWheeledVehicle $\sqsubseteq =$ 4HasWheel.Wheel.

References

1. 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 (2003)
2. Bollacker, K.D., Cook, R.P., Tufts, P.: Freebase: A shared database of structured general human knowledge. In: AAAI. pp. 1962–1963. AAAI Press (2007)
3. Chah, N.: Freebase-triples: A methodology for processing the freebase data dumps. CoRR **abs/1712.08707** (2017)
4. Cimiano, P., Völker, J.: Text2onto. In: NLDB. Lecture Notes in Computer Science, vol. 3513, pp. 227–238. Springer (2005)
5. Donnelly, K.: Snomed-ct: The advanced terminology and coding system for ehealth. Studies in health technology and informatics **121**, 279 (2006)
6. Fanizzi, N., d'Amato, C., Esposito, F.: DL-FOIL concept learning in description logics. In: ILP. Lecture Notes in Computer Science, vol. 5194, pp. 107–121. Springer (2008)
7. Färber, M., Bartscherer, F., Menne, C., Rettinger, A.: Linked data quality of dbpedia, freebase, opencyc, wikidata, and YAGO. Semantic Web **9**(1), 77–129 (2018)
8. Färber, M., Ell, B., Menne, C., Rettinger, A.: A comparative survey of dbpedia, freebase, opencyc, wikidata, and yago. Semantic Web Journal **1**, 1–5 (2015), `http://www.semantic-web-journal.net/system/files/swj1141.pdf`
9. Fortuna, B., Grobelnik, M., Mladenic, D.: Ontogen: Semi-automatic ontology editor. In: HCI (9). Lecture Notes in Computer Science, vol. 4558, pp. 309–318. Springer (2007)
10. Havasi, C., Speer, R., Arnold, K.C., Lieberman, H., Alonso, J.B., Moeller, J.: Open mind common sense: Crowd-sourcing for common sense. In: Collaboratively-Built Knowledge Sources and AI. AAAI Workshops, vol. WS-10-02. AAAI (2010)
11. Hoehndorf, R., Schofield, P.N., Gkoutos, G.V.: The role of ontologies in biological and biomedical research: a functional perspective. Briefings in Bioinformatics **16**(6), 1069–1080 (2015)
12. Ismayilov, A., Kontokostas, D., Auer, S., Lehmann, J., Hellmann, S.: Wikidata through the eyes of dbpedia. Semantic Web **9**(4), 493–503 (2018)
13. Konev, B., Lutz, C., Ozaki, A., Wolter, F.: Exact learning of lightweight description logic ontologies. Journal of Machine Learning Research **18**, 201:1–201:63 (2017)
14. Lamy, J.: Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies. Artificial Intelligence in Medicine **80**, 11–28 (2017)
15. Lehmann, J., Auer, S., Bühmann, L., Tramp, S.: Class expression learning for ontology engineering. J. Web Sem. **9**(1), 71–81 (2011)
16. Matuszek, C., Cabral, J., Witbrock, M.J., DeOliveira, J.: An introduction to the syntax and content of cyc. In: AAAI Spring Symposium: Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering. pp. 44–49. AAAI (2006)
17. Rebele, T., Suchanek, F.M., Hoffart, J., Biega, J., Kuzey, E., Weikum, G.: YAGO: A multilingual knowledge base from wikipedia, wordnet, and geonames. In: International Semantic Web Conference (2). Lecture Notes in Computer Science, vol. 9982, pp. 177–185 (2016)
18. Speer, R., Chin, J., Havasi, C.: Conceptnet 5.5: An open multilingual graph of general knowledge. In: AAAI. pp. 4444–4451. AAAI Press (2017)