FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Semi-Automatic Engineering of Topic Ontologies from a Common-Sense Knowledge Graph

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Katinka Böhm, BSc
Matrikelnummer 0826017

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof.Dr.techn. Maria Magdalena Ortiz de la Fuente, MSc

Wien, 27. August 2018

Katinka Böhm

Maria Magdalena Ortiz de la Fuente

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Semi-Automatic Engineering of Topic Ontologies from a Common-Sense Knowledge Graph

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Computational Intelligence

by

Katinka Böhm, BSc
Registration Number 0826017

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof.Dr.techn. Maria Magdalena Ortiz de la Fuente, MSc

Vienna, 27th August, 2018

Katinka Böhm

Maria Magdalena Ortiz de la Fuente

Erklärung zur Verfassung der Arbeit

Katinka Böhm, BSc
Biberhaufenweg 100/39

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 27. August 2018

Katinka Böhm

Acknowledgements

I would first like to thank the people at the WTM research group of the University of Hamburg for giving me the inspiration and idea for this thesis.

Most of my thanks goes to my thesis advisor Magdalena Ortiz of the Faculty of Informatics at the Vienna University of Technology, who gave me the possibility and support to transform a first idea into this academic work. She guided my research and always took the time out of her busy schedule to listen to my questions and offer advice whenever I needed it.

Finally, I must express my very profound gratitude to my parents and my whole family, for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

"This work was conducted using the Protégé resource, which is supported by grant GM10331601 from the National Institute of General Medical Sciences of the United States National Institutes of Health."

Kurzfassung

Durch die Nachfrage im Bereich Künstliche Intelligenz und Semantic Web Technologien haben sich Semantische Netzwerke, Wissensgraphen und Ontologien, die die Organisation und Modellierung von Wissen ermöglichen, stark verbreitet. Neben dem bekannten Google Knowledge Graphen existieren weitere Wissensgraphen wie ConceptNet, Freebase, YAGO, OpenCyc und DBPedia, die durch Informationsextraktion aus unterschiedlichen Internetquellen riesige Mengen an strukturiertem Allgemeinwissen frei zur Verfügung stellen. Jedoch machen fehlerhafte Inhalte und eine Vielfalt an Wissen die automatische Weiterverarbeitung zu einer Herausforderung. Nachdem Wissensgraphen, im Gegensatz zu formalen Ontologien, nicht auf wohldefinierter Syntax und Semantik basieren, können sie auch nicht ohne weiteres für Aufgaben eingesetzt werden, die korrektes und nicht-triviales logisches Schließen verlangen. Wir vertreten die These, dass existierende Wissensgraphen genutzt werden können, um kleine bis mittelgroße themenspezifische Ontologien aufzubauen, die Fakten zu alltäglichen Themen beinhalten. In dieser Arbeit stellen wir eine einfache aber effektive Methode vor, um relevantes Wissen aus einem Wissensgraphen zu extrahieren und beschreiben ein interaktives, schrittweises Verfahren, das es ermöglicht vorschlagsunterstützt innerhalb kurzer Zeit eine Themen-Ontologie in Beschreibungslogik (Description Logic) zu erstellen. Wir haben das Verfahren in dem kleinen Kommandozeilentool *CN2TopicOnto* implementiert, das Anfragen an ConceptNet schickt und basierend auf den Ergebnissen, Konzepte vorschlägt. Die Vorschläge können vom Anwender genutzt werden um Axiome zu formulieren, die von einfachen Inklusionen zwischen Konzepten bis zu Axiomen der erweiterten Bescheibungslogik reichen. Um den Nutzen unseres halbautomatischen Tools zu veranschaulichen, präsentieren wir einige unserer mit *CN2TopicOnto* erstellten Ontologien zu den ausgewählten Themen *Tiere, Früchte, Fahrzeuge* und *Naturkatastrophen*.

Abstract

With the advancements in artifical intelligence and semantic web technologies, structures to organize and model data, such as semantic networks, knowledge graphs, and ontologies, have become widespread. A well-knwon example is the Google Knowledge Graph, but there are many other projects, such as ConceptNet, Freebase, YAGO, OpenCyc and DBPedia, that are openly available, and, as the result of knowledge extraction and data mining techniques, contain vast amounts of organized common-sense and general world knowledge on different topics. However, the quality of the knowledge they contain varies, and irrelevant and possibly flawed contents coexist with meaningful knowledge, making them harder to grasp. They also lack a formal logic-based semantics, as enjoyed by ontologies, and therefore they cannot be readily exploited for tasks that require correct and non-trivial reasoning.

We claim that the vast work invested into knowledge graphs can be leveraged to build, small- to middle-sized, topic-specific ontologies of every-day domains. We describe a simple, yet effective method that extracts thematically relevant knowledge from a knowledge-graph and, in a stepwise procedure, provides suggestions that help to interactively build a Description Logic (DL) topic ontology with moderate efforts. We present an implementation of our method in the semi-automatic *CN2TopicOnto* tool that systematically queries ConceptNet and suggests suitable concept names to the user, who can use them to create axioms in a simple command-line interface. The supported axioms range from plain inclusions between classes, to complex ontological axioms in expressive DLs. With our proof-of-concept prototype it is possible to build ontologies on different everyday topics in reasonable time. To demonstrate its usefulness we describe some illustrative ontologies on the topics *animals, fruits, vehicles* and *natural disasters* that were created with *CN2TopicOnto*.

Contents

CHAPTER 1

Introduction

With the increasing development and expansion of the Semantic Web, and the rapid growth of domain-specific data, knowledge-representation and reasoning systems, like ontologies, semantic networks, knowledge graphs and structured databases, have become irreplaceable tools to organize and model data. As a result, there has been an increasing effort to formally define requirements and standards for such types of structured knowledge. *Ontologies* and *knowledge graphs* are two prominent ways to organize knowledge that have different purposes and objectives.

In the most general sense, an ontology is a descriptive system that is used to formally model a complex domain. It is a sharable domain conceptualization that offers interchangeability of information through well-defined formalisms which allow for automated inference- and reasoning support. Together with a reasoning engine they form a complete system that incorporates artificial intelligence algorithms to solve reasoning problems. Ontologies are extensively used for scientific classifications, especially in biology and medicine [JG11] where large standardized and structured vocabularies are needed. They help scientists manipulate and organize their experimental data, and build the backbone for many industry applications.

The current standard for ontology representation is the Web Ontology Language (OWL) which is widely used to serialize and exchange ontologies on the Web. OWL is a family of standardized knowledge representation languages that are built on W3Cs XML standard Recourse Description Framework (RDF). There exist three different levels of expressiveness; OWL Lite, OWL DL, and OWL Full. The semantics of the first two are based on *Description Logics*, a family of logics that are decidable fragments of first-order logic. The newest version, OWL2, extends the previous specifications to provide different syntaxes and semantics. The OWL2 Direct Semantics are compatible with the model theoretic semantics of the $\mathcal{SROIQ}$ Description Logic. There exist three different profiles: OWL2 EL, OWL2 QL, and OWL2 RL, all syntactic restrictions of OWL DL with the purpose of trading expressive power for computational benefits [Gro12].

Another popular way to store organized knowledge are knowledge graphs. A knowledge graph is a graph-based domain representation for knowledge management. After the introduction of its Knowledge Graph in 2012, Google has established the term and it has since been picked up and used generically for different architectures and applications. There is no agreed upon definition, as knowledge graphs are often used synonymously to semantic networks. Frequently knowledge graphs themselves are described as ontologies and vice versa.

Two defining characteristics for knowledge graphs were listed by Paulheim [Pau17] as follows: a knowledge graph (1) describes real world entities and their interrelations, organized in a graph, and (2) covers various topical domains.

Another definition in Färber et al.[FBMR18] distinctly links knowledge graphs to RDF triples of the form (s, p, o) that join *subjects (s)* to *objects (o)* with the use of *predicates (p)*.

Most systems referred to as knowledge graphs, (semi-)automatically extract and integrate information from multiple external sources through data mining techniques, and apply some form of reasoning to further enrich the initial knowledge [EW16]. As a result, most of them contain a lot of *encyclopedic knowledge*, such as knowledge about well-known people, dates and geographical locations.

Examples of big knowledge graphs include DBPedia, Freebase, OpenCyc, Wikidata, YAGO [FEMR15, FBMR18] and ConceptNet [SH12], which plays a prominent role in this thesis. *ConceptNet* can be considered a knowledge graph according to the two definitions we gave above: It has its foundation in MITs Open Mind Common Sense crowdsourcing project [HSA+10] and therefore stores knowledge, expressed by humans, on various topics; and encodes its statements in approximately 28 million annotated triples using JSON-LD (a JSON format extension of the RDF data model).

Mere ontologies often differ from knowledge graphs in the amount of terminological, or schematic knowledge they contain. While the former aim to provide axioms on a structural level, the latter contain a higher magnitude of instance-level statements [Pau17]. This finding is compatible with the fact that most knowledge graphs focus on explicit *general knowledge* that does not allow for gradual interpretation, such as *Albert Einstein was born in 1879* or *London is a city*. Exemptions are ConceptNet and OpenCyc, which store *common-sense* facts, such as the knowledge that *fire is hot* or *a car is parked in a parking lot*. Common-sense knowledge tends to be susceptible to sentiment and is oftentimes conditional or dynamic, which increases the complexity and defeasibility of existing models. Implicit knowledge is frequently completely overlooked or not properly integrated, which results in incorrect reasoning that becomes significantly harder to backtrack and correct the larger models become.

As knowledge graphs cover a broader scope and prioritize quantity of data over well-defined semantics, often possibly irrelevant and imprecise facts coexist with relevant knowledge and knowledge domains are interrelated in unpredictable ways. As a result they cannot be readily exploited for tasks that require non-trivial reasoning. On the other

end of the spectrum, many high-quality ontologies exist that focus on highly specialized domains, like life-sciences and health care [HSG15], such as the well-known SNOMED CT [Don06]. Tailored for experts and manually curated and extended over years, they are far too large and complex to allow a non-expert to get an understanding of the domain they describe. Small, manageable and freely-available ontologies, describing non-technical every-day topics, are hard to come by. Hand-crafted toy examples, like the Manchester Pizza Ontology[1] and the DAML Wine Ontology[2] exist, but they only cover two out of many possible every-day topical domains.

Overall the distinctions between semantic networks, knowledge graphs and ontologies are not very clear. For this thesis we will stick to a strict, formal Description Logic-based definition of an ontology, as introduced in the next section, and a very broad definition of a knowledge graph as a set of possibly annotated triples. Each triple forms a statement and all triples together induce an abstract graph representation. We presuppose the existence of an evaluative weight as a measurement of correctness respectively certainty of each statement. Many knowledge graphs compute such weights from distributional similarities in text. Content-wise, we target common-sense and broad general knowledge over encyclopedic knowledge, but want to emphasize that the distinction relating thereto is not clear-cut either.

We make the following contributions in this thesis:

- We present a method to interactively build Description Logic topic ontologies with moderate efforts, by leveraging knowledge from knowledge graphs. The constructed topic ontologies are comprehensive sets of axioms on a chosen topic of interest, and should be able to answer the following competency questions with the help of an off-the-shelf reasoner:
  - What (common-sense) properties does a specific concept have?
  - Which properties are shared by different concepts? Which concepts share a common property?
  - Are there characteristics that possibly define a concept based on its elements or subsumed concepts?

  For this we claim that the vast amount of structured data that exists in knowledge graphs can be leveraged to make suggestions which help to quickly construct a small- to middle-sized topic ontology. We claim that such ontologies can be valuable for illustrative and didactic purposes, as well as in research, for example, for testing proof-of-concept prototypes.
- To prove our claim we implemented an algorithm in Python as a simple command-line tool that we call *CN2TopicOnto*, which extracts knowledge from ConceptNet to suggest concept names, and allows the user to interactively define axioms using both suggested and self-defined concepts and roles. Different knowledge domains can be quickly

[1] https://protegewiki.stanford.edu/wiki/Protege4Pizzas10Minutes
[2] https://www.w3.org/TR/2003/PR-owl-guide-20031215/wine

explored by simply changing the defined topic at the start of the construction process. Axioms are added in consecutive steps and range from plain concept inclusions to inclusions with complex $\mathcal{ALCIO}$ concepts. For the task, we identify suitable relations in ConceptNet that are likely to encode ontological knowledge relevant to the user, and establish adequate orderings and thresholds to process suggestions in a convenient way that does not overwhelm the user. Resulting ontologies have the benefit of being adaptable and arbitrarily extendable, both with our tool and with existing ontology-editing tools. This makes them susceptible to further improvement and refinement so that they can be used as a starting point for engineering larger high-quality ontologies. This reduces the effort needed to develop them from scratch.

- In Chapter 6 we discuss some related work and research in the field of *ontology engineering*. Using our method, we give a view on the process of knowledge elicitation and ontology construction from the perspective of ontology engineering. For this we examine good concept hierarchy design and briefly present established design methods such as ontology normalization, and the adequate modeling of modifying concepts, which are concepts that describe adjectives or adverbs.
- We illustrate the usefulness of our simple prototype by describing some illustrative ontologies on topics such as *vehicles*, *fruits* and *animals*. We compare multiple sample ontologies by providing statistics on axiom count, vocabulary size, and approximate time spent engineering them with our tool.
- We discuss some difficulties regarding the use of ConceptNet for knowledge extraction for our tool and describe methods to alleviate arising issues. We briefly analyze the type of ontologies that our algorithm produces, and their boundaries for common-sense knowledge, from a Description Logic perspective, and discuss possible uses as well as further improvements and extensions that could be implemented in the future.

CHAPTER 2

Preliminaries

Description Logics (DL), as a fragment of classical first order logic, are a family of languages for Knowledge Representation and Reasoning [BN03, BHLS17]. As such, they form the formal foundation of OWL.
This section provides some preliminaries and gives an overview of the required DL formalisms. At first we introduce the main components of a DL ontology. Then we formally define the syntax and semantics of the basic DL $\mathcal{ALC}$, followed by the relevant constructors that are needed to extend $\mathcal{ALC}$ to $\mathcal{ALCIO}$.

2.1 DL Ontologies

An *ontology* is defined by a set of formulas, which follow a well-defined syntax and semantics. The two building blocks of a DL ontology are *Concepts* (often called *Classes*) and *Roles* (also called *Relations* or *Properties*). Both are used to build *axioms*, the statements that are true within the ontology. The domain elements of an ontology are called *individuals* (sometimes *instances*). Individuals are subject to the restrictions defined by the axioms. Ontology axioms are split into two types, which are stored as the *terminological component (TBox)* and the *assertional component (ABox)*. In what follows we introduce a DL vocabulary and formally define how it is used to build a DL ontology.

Definition 2.1.1 ((DL) Vocabulary)**.**
The basic vocabulary *contains countably infinite, pairwise disjoint sets*

(a) N_C*, of concept names,*
(b) N_R*, of role names,*
(c) N_I*, of individual names.*

With the vocabulary we define *atomic* and *basic concepts* as the most primitive building blocks of an ontology.

Definition 2.1.2 (Atomic and Basic Concepts)**.**
We call a concept name $A \in N_C$, atomic concept*. An atomic concept is a unary predicate that defines a subset of domain elements that share a common property. A* basic concept *B is an atomic concept or a nominal $\{a\}$ where $a \in N_I$. Basic concepts are the main building blocks of an ontology.*

Examples: Student, Fruit, AutomatedVehicle, {london}, {the_times_newspaper}

Definition 2.1.3 (Roles)**.**
A role name *$R \in N_R$ is a binary predicate that defines the relation between a pair of elements of the concepts.*

Examples: PartOf, HasAnatomicalPart, Contains, HasColor, WorkEnvironmentOf

Concept and role *constructors*, such as $\sqcap, \sqcup, \neg$ and $\cap, \cup, .^-$ are then used to build more complex *concepts* and *roles*.

Examples: Animal $\sqcup$ Biped, RomanceNovel $\sqcup$ HistoryNovel, $\exists$HasSpecies.MammalSpecies, Car $\sqcap \forall$ HasColor.{red} and HasPart$^-$, EnrolledIn $\cup$ Attends, $\neg$HasFeature

Throughout the thesis we will use A to denote atomic concepts and B to denote basic concepts. We will use C, D to denote general concepts that possibly contain constructors.

Definition 2.1.4 (TBox)**.**
The TBox defines concepts and roles and constraints the relationships between them. There are two main kinds of terminological axioms:

- *General concept inclusions (GCIs): $C \sqsubseteq D$*
- *Definitions: $A \equiv C$, where A has to be an atomic concept*

The TBox $\mathcal{T}$ is then defined as a set of terminological axioms. Each axiom contains a lefthandside (LHS) *and a* righthandside (RHS) *respectively.*

We differentiate between two types of GCIs, I. atomic concept inclusions (ACIs) *$A \sqsubseteq A'$, where A and A' are atomic concepts, and II.* complex *GCIs that contain constructors and role quantification* $(\forall, \exists)$*.*

Intuitively, a TBox contains schema information that introduces the vocabulary and the rules of the world.

Examples: Elephant $\sqsubseteq \exists$ HasSpecies.MammalSpecies,
Spoon $\sqcup$ Knife $\sqcup$ Fork $\sqsubseteq$ Cutlery,
Watercraft $\equiv$ Vehicle $\sqcap \exists$ UsedOn.Watersurface,
GreenApple $\equiv$ Apple $\sqcap \forall$ HasColor.Green

Definition 2.1.5 (ABox)**.**
The ABox $\mathcal{A}$ consists of concept and role membership assertions *that assign a single individual $a \in N_I$ to an atomic concept A, denoted $A(a)$, or semantically link a pair of individuals (a, b) through a role predicate R, denoted $R(a, b)$.*

Intuitively, the ABox contains factual information and can be seen as a partial description of the world.

Examples: Car(toyotacorolla), HistoryNovel(warandpeace),
HasPart(apple, appleseed), OfferForSale(ikea, pax)

We can now define a DL ontology. Note that in DL terminology, the terms *knowledge base (KB)* and ontology are often used interchangeably.

Definition 2.1.6 ((DL) Ontology)**.**
A (DL) ontology or KB $\mathcal{O}$, is a pair $(\mathcal{T}, \mathcal{A})$, where the TBox $\mathcal{T}$ is a finite set of GCIs and the ABox $\mathcal{A}$ is a finite set of (concept and role) membership assertions.

For simplicity whenever we use the term *ontology*, we refer to a DL ontology.

We will denote all names that are effectively part of an ontology $\mathcal{O}$ the *effective vocabulary* and refer to the respective sets through $N_C(\mathcal{O})$, $N_R(\mathcal{O})$ and $N_I(\mathcal{O})$.

2.2 Syntax and Semantics of $\mathcal{ALCIO}$

DLs are a hierarchy of decidable logics with increasing expressive power and computational complexity. They range from *Lightweight DLs*, with very restricted expressiveness, to *expressive DLs*, that can model more complex domains but have higher computational complexity. The basic DL language is called $\mathcal{ALC}$. We shortly summarize the $\mathcal{ALC}$ syntax and define respective semantics, before introducting the extentions needed for $\mathcal{ALCIO}$.

2.2.1 Syntax and Semantics of $\mathcal{ALC}$

Let N_C, N_R and N_I be countably infinite, pairwise disjoint alphabets of concept names, role names, and individuals.

Concepts C obey the grammar $C \to \top \mid \bot \mid A \mid \{a\} \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C$, where $A \in N_C$ is an atomic concept, D is a concept, $a \in N_I$, and $R \in N_R$.

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of

- a non-empty set $\Delta^{\mathcal{I}}$ called *domain*
- an *interpretation function* $\cdot^{\mathcal{I}}$

The interpretation function $\cdot^{\mathcal{I}}$ maps every concept C to $C^{\mathcal{I}}$, where $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, every role R to $R^{\mathcal{I}}$, where $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and every individual a to $a^{\mathcal{I}}$, where $a \in \Delta^{\mathcal{I}}$.

The interpretation function is then extended to all concepts through

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}}, \\
\bot^{\mathcal{I}} &= \emptyset, \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}, \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \sqcap D^{\mathcal{I}}, \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \sqcup D^{\mathcal{I}}, \\
(\forall R.C)^{\mathcal{I}} &= \{a_1 \mid \forall a_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}}(a_1, a_2) \rightarrow a_2 \in C^{\mathcal{I}}\}, \\
(\exists R.C)^{\mathcal{I}} &= \{a_1 \mid \exists a_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}}(a_1, a_2) \wedge a_2 \in C^{\mathcal{I}}\},
\end{aligned}$$

We define *satisfiability* in an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$.

Definition 2.2.1 (Satisfiability)**.**
Assume an interpretation $\mathcal{I}$. Then $\mathcal{I}$ satisfies a GCI $C \sqsubseteq D$ *if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and $\mathcal{I}$* satisfies an assertion $C(a)$, *respectively* $R(a_1, a_2)$ *if $a \in C^{\mathcal{I}}$, respectively $({a_1}^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.*

As a consequence, $\mathcal{I}$ satisfies a TBox $\mathcal{T}$ *if it satisfies every GCI in $\mathcal{T}$, and $\mathcal{I}$* satisfies a ABox $\mathcal{A}$ *if it satisfies every assertion in $\mathcal{A}$.*

An interpretation $\mathcal{I}$ is called a *model* of an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$, if it satisfies $\mathcal{A}$ and $\mathcal{T}$. With the notion of *satisfiability*, we now define *entailment*.

Definition 2.2.2 (Entailment)**.**
Assume an ontology $\mathcal{O}$ and concepts C, D. Then $\mathcal{O}$ entails *the axiom $C \sqsubseteq D$, written $\mathcal{O} \models C \sqsubseteq D$, if and only if the axiom $C \sqsubseteq D$ holds in $\mathcal{O}$.*

An axiom $C \sqsubseteq D$ holds in *$\mathcal{O}$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ in every* model *of $\mathcal{O}$.*

2.2.2 $\mathcal{ALCIO}$, an Example of an Expressive DL

For $\mathcal{ALCIO}$, $\mathcal{ALC}$ is extended with the concept constructor for *nominals* $\mathcal{O}$ and the role constructor for *inverses* $\mathcal{I}$.

Nominals $\mathcal{O}$

With nominals $\mathcal{O}$ it holds that, if $a_1, \ldots, a_n$ are individuals, then $\{a_1, \ldots, a_n\}$ is a concept with the interpretation $\{a_1, \ldots, a_n\}^{\mathcal{I}} = \{{a_1}^{\mathcal{I}}, \ldots, {a_n}^{\mathcal{I}}\}$.

Not that basic concepts $\{a\}$ are nominals over one individual $a \in N_I$.

With nominals it is possible to describe concepts as sets of individuals.

Examples: {asia, africa, north_america, south_america, antarctica, europe, australia} ≡ Continent, {red, green, blue, yellow} ⊑ Color

Inverses $\mathcal{I}$

With inverses $\mathcal{I}$ it holds that if R is a role, then R^- is a role, where $(R^-)^{\mathcal{I}} = \{(a_2, a_1) \mid (a_1, a_2) \in R^{\mathcal{I}}\}$

Note that the following equivalence holds: $C \sqsubseteq \exists R^-.D \equiv \forall R.C \sqsubseteq D$.

Examples: $\mathsf{HasPart}^- = \mathsf{PartOf}$, $\mathsf{StudentOf}^- = \mathsf{TeacherOf}$

Other common extentions are *(qualified) number restrictions* ($(\mathcal{Q})\mathcal{NR}$), *self concepts* and *role inclusions* ($\mathcal{H}$). Those build the basis of the DLs $\mathcal{SHIQ}$ and $\mathcal{SHOIQ}$ that are related to the OWL languages.

2.2.3 Range Restriction and Disjointness Axioms

We further define two types of axioms that will be relevant for the ontologies discussed in this thesis.

Definition 2.2.3 (Range Restriction)**.**
A range restriction *for a role R and a concept C can be expressed as*

$$\exists R^-.\top \sqsubseteq C \tag{2.1}$$

We denote this with the short form $Ran(R, C)$.

Definition 2.2.4 (Disjointness)**.**
Disjointness *of concepts C, D can be expressed as*

$$C \sqcap D \sqsubseteq \bot \tag{2.2}$$

We denote this with the short form $Disj(C, D)$.

A set of concepts $\mathcal{C} = \{C_1, \ldots, C_n\}$ is called disjoint *if and only if all C_i are* pairwise disjoint*, that is*

$$C_i \sqcap C_j \sqsubseteq \bot \quad \text{for all } C_i \neq C_j \in \mathcal{C} \tag{2.3}$$

CHAPTER 3

Topic Ontologies over Knowledge Graphs

In this thesis we introduce a construction method for *topic ontologies*. For this we provide a formal definition of a topic ontology. Additionally we explain some general notions that are important for the construction process and establish required terminology. The presented notions are independent from the chosen DL. In the next chapter we will provide a technique for building topic ontologies in $\mathcal{ALCIO}$ based on those notions.

Throughout the thesis, all relation triples that originate from a knowledge graph, in particular ConceptNet, are displayed in cursive font as *(Concept1, Relation, Concept2)* and we use sans serif font for DL expressions, for example $\mathsf{Concept1} \sqsubseteq \forall\ \mathsf{RoleName}.\mathsf{Concept2}$.

3.1 Topic Ontologies

We define a *Topic Ontology* as a DL ontology that is modeled around a specific *topic*. The topic specifies the area of interest and thereby the domain. All contained axioms revolve around the topic.

Definition 3.1.1 (Topic Ontology)**.**
A topic ontology *is a pair* $\mathcal{X} = \langle \mathcal{T}, tc \rangle$ *where tc is a concept name that we call the* topic concept *and* $\mathcal{T}$ *is the TBox of a DL ontology, where tc occurs in* $\mathcal{T}$*.*

According this definition the Manchester Pizza Ontology is an example of a topic ontology, where $tc = Pizza$ [1]. Other examples are the refined OWL version of the DAML Wine Ontology [2] and LUBM (Lehigh University Benchmark)[GPH05], a university domain ontology that can be used to evaluate systems and their reasoning capabilities.

[1] https://protege.stanford.edu/ontologies/pizza/pizza.owl
[2] https://www.w3.org/TR/2003/PR-owl-guide-20031215/wine

For our tasks purpose the *tc* is a single or compound word that is chosen in advance and defines the domain expected to be covered by the constructed ontology. It acts as a starting point for the effective vocabulary and axioms of $\mathcal{X}$, and should be a general term that covers a sufficient area.
Topic concepts that fulfill those requirements and were used in the example ontologies we constructed are *vehicle, fruit, animal, university* and *fast food*. The respective topic ontologies contain information about types of vehicles, varieties of fruits, species of animals, and typical fast-food items. They express associated colors and behavioral characteristics. The university ontology models parts of a typical administration hierarchy and includes representations of well-known universities.

3.1.1 The Taxonomy

The general definition of a taxonomy is a classification schema for concepts, terms or artifacts. Most taxonomies are hierarchical and obey a single type of parent-child relationship, such as whole-part or genus-species [webb].

The topic ontologies we build have large taxonomies, hence they provide a fitting skeletal structure and are interesting to examine.

For our purpose we consider the taxonomy of an ontology to be the subset of TBox axioms that includes only ACIs. As such it defines a hierarchy of named concepts, such that $\mathcal{O} \vDash A(x) \Rightarrow \mathcal{O} \vDash B(x)$ holds for an individual x and concepts A, B. It is subject to the supposition that A is a more specialized concept of B.

Definition 3.1.2 (Taxonomy)**.**
The taxonomy Tax *of an ontology* $\mathcal{O}$ *is defined as*

$$\mathrm{Tax}(\mathcal{O}) = \{A \sqsubseteq B \in \mathcal{T} \mid A, B \text{ atomic concepts other than } \top\}.$$

$N_C(\mathrm{Tax})$ *is the set of all concept names occurring in* Tax*.*

While it directly follows that $\mathcal{O} \vDash A \sqsubseteq B$ if $A \sqsubseteq B \in \mathrm{Tax}(\mathcal{O})$, the converse is not true. In what follows, we define the closure of $\sqsubseteq$ over Tax. This allows for weaker statements about concept subsumptions that hold within the taxonomy.

Definition 3.1.3 (Closure of $\sqsubseteq$ over Tax)**.**
We define $\sqsubseteq^*_{\mathrm{Tax}(\mathcal{O})}$ *as the reflexive transitive closure of* $\sqsubseteq_{\mathrm{Tax}(\mathcal{O})} = \{(A, B) \mid A \sqsubseteq B \in \mathrm{Tax}(\mathcal{O})\}$ *over the taxonomy* Tax *of* $\mathcal{O}$*.*

Corollary 3.1.1.

$$A \sqsubseteq^*_{\mathrm{Tax}(\mathcal{O})} B \Leftrightarrow \mathrm{Tax}(\mathcal{O}) \vDash A \sqsubseteq B \tag{3.1}$$

$$A \sqsubseteq^*_{\mathrm{Tax}(\mathcal{O})} B \text{ and } B \sqsubseteq^*_{\mathrm{Tax}(\mathcal{O})} A \Leftrightarrow \mathrm{Tax}(\mathcal{O}) \vDash A \equiv B \tag{3.2}$$

Definition 3.1.4 (Concept Equivalence)**.**
Let $\mathcal{O}$ be an ontology with atomic concepts $A, B \in \text{Tax}(\mathcal{O})$.
Then A and B are called (semantically) equivalent *if and only if* $\text{Tax}(\mathcal{O}) \models A \equiv B$.

We use $\sqsubseteq^*_{\text{Tax}(\mathcal{O})}$ to navigate the taxonomy of $\mathcal{O}$, and establish the concept names that are most relevant to the topic. For our purpose Definition 3.1.2 of a taxonomy suffices, as we do not use it for entailment or reasoning, but for syntactic traversal of axioms.

Definition 3.1.5 (Graph Representation of Tax)**.**
A taxonomy Tax *can be visualized as a directed graph $\mathcal{G}_{\text{Tax}} = (\mathcal{V}, \mathcal{E})$, with a set $\mathcal{V}$ of vertices where $\mathcal{V} := N_C(\text{Tax})$ and $\mathcal{E}$, a set of directed edges (arcs) such that $(A, B) \in \mathcal{E} \Leftrightarrow A \sqsubseteq B \in \text{Tax}$.*

Figure 3.1 shows selected parts of $\mathcal{G}_{\text{Tax}}$ of the Animal Ontology. All graphical taxonomy visualizations in this thesis were done with the Protegé plug-in OwlViz. An `is-a` arc between concepts A and B is equivalent to $A \sqsubseteq B$. Black arrowheads indicate that further concepts exist that are not displayed. Purple and green arrows emphazise the incoming and outgoing connections for the highlighted topic concept.

Definition 3.1.6 ((Immediate) Subclass)**.**
Let $\mathcal{O}$ be an ontology with atomic concepts $A, B \in \text{Tax}(\mathcal{O})$, such that $\mathcal{O} \not\models A \equiv B$.
A is called an (immediate) subclass *of B in $\mathcal{O} \Leftrightarrow A \sqsubseteq B \in \text{Tax}(\mathcal{O})$.*

(Immediate) subclasses correspond to direct successors *(also called* children*) in $\mathcal{G}_{\text{Tax}}$.*

Definition 3.1.7 ((Immediate) Superclass)**.**
Let $\mathcal{O}$ be an ontology with atomic concepts $A, B \in \text{Tax}(\mathcal{O})$, such that $\mathcal{O} \not\models A \equiv B$.
B is called an (immediate) superclass *of A in $\mathcal{O} \Leftrightarrow A \sqsubseteq B \in \text{Tax}(\mathcal{O})$.*

(Immediate) superclasses correspond to direct predecessors *(also called* parents*) in $\mathcal{G}_{\text{Tax}}$.*

Definition 3.1.8 (Ancestors and Descendants)**.**
Let $\mathcal{O}$ be an ontology with atomic concepts $A, B \in \text{Tax}(\mathcal{O})$, such that $\mathcal{O} \not\models A \equiv B$.
If $A \sqsubseteq^*_{\text{Tax}(O)} B$ then B is called an ancestor *of A and A is called a* descendant *of B.*

An ancestor B' of A is greater *than $B \Leftrightarrow B \sqsubseteq^*_{\text{Tax}(O)} B'$ and $B' \neq B$.*
A descendant A' of B is smaller *than $A \Leftrightarrow A' \sqsubseteq^*_{\text{Tax}(O)} A$ and $A' \neq A$.*

Remarks.

- The greatest ancestor of any ontology taxonomy is the concept *Top* ($\top$, Thing). It is the concept that subsumes every other concept by default. For reasons of simplicity, we do not consider $A \sqsubseteq \top$ part of Tax for any $A \in N_C(\mathcal{O})$ and assume $\top$ to not be part of $\text{N}_C(\text{Tax})$.
- $\text{Tax}(\mathcal{O})$ is hierarchical if descendants are more specialized concepts than their ancestors.
- Atomic concepts that have no further subclasses correspond to *leafs* in $\mathcal{G}_{\text{Tax}}$.
- An atomic concept can have more than one immediate superclass. Therefore Tax defines a poly-hierarchy and the graph $\mathcal{G}_{\text{Tax}}$ in general does not have a tree-structure.

Definition 3.1.9 (Levels of an Ontology Taxonomy)**.**
The topmost level (level 1) *of* Tax *is the set of all atomic concepts from* $N_C(\text{Tax})$ *that do not have further superclasses. A concept* A *is on* level n of the taxonomy *if* $\max_i(level(B_i)) = n - 1$ *for all immediate superclasses* B_i *of* A*.*
In case of a cycle in Tax*, all atomic concepts* A_i *that are part of the cycle are considered to be on the same level, namely* $\min_i(level(A_i))$*.*

We also say that an atomic concept A *is* k levels lower/downwards with respect to another concept B *if there exist exactly* $k - 1$ *concepts* $A_i \in N_C(\text{Tax})$ *with* $\{A \sqsubseteq A_1, A_1 \sqsubseteq A_2, \ldots, A_{k-1} \sqsubseteq B\} \subseteq \text{Tax}$*, building a path in* $\mathcal{G}_{\text{Tax}}$*. Comparably* B *is* k level higher/topwards with respect to A*.*

Definition 3.1.9 assigns a unique level to each atomic concept, but also allows to talk about all atomic concepts that occur on a specific level w.r.t. another concept. Note that in the second case, one concept can be on more than one level w.r.t. another concept, dependent on the structure of Tax, as multiple connecting paths may exist in $\mathcal{G}_{\text{Tax}}$. The following example should make the distinction clearer.

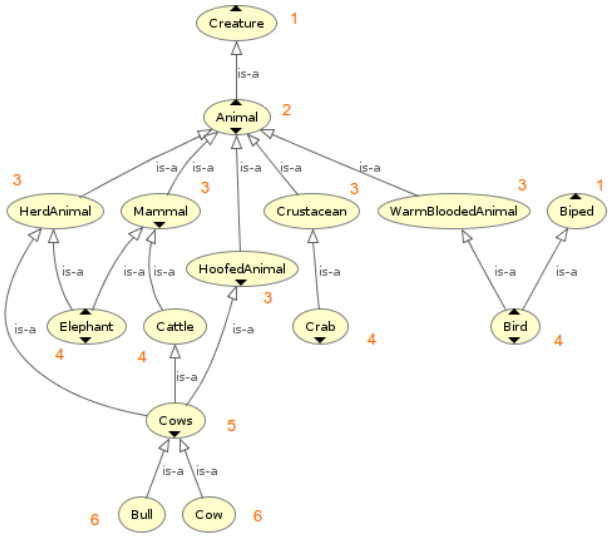


Figure 3.3: Subgraph of Figure 3.2. Unique taxonomy levels of the displayed subpart of the Animal Taxonomy are depicted in orange.

Consider the Animal Ontology in Figure 3.2. Six concepts, namely Creature, Sand, Part, Habitat, Species and Biped are at the topmost level of the taxonomy. Figure 3.3 shows the subgraph with respective (unique) levels annotated in orange next to each concept. As can be seen HerdAnimal and HoofedAnimal are assigned level 3 and Cows is assigned level 5, even though $\text{Cows} \sqsubseteq \text{HerdAnimal}$, $\text{Cows} \sqsubseteq \text{HoofedAnimal}$ makes Cows an immidiate subclass of both HerdAnimal and HoofedAnimal. The concept Cattle pushes Cows one level lower in the overall taxonomy. Cows is then two levels lower than Mammal, but simultaneously two and three levels lower w.r.t the concept Animal, based on the existence of paths Animal-HerdAnimal-Cows, Animal-HoofedAnimal-Cows and Animal-Mammal-Cattle-Cows.

3.2 Topic Ontologies Linked to Knowledge Graphs

To build topic ontologies, our approach exploits an existing knowledge graph to generate thematic suggestions that match the topic. Here we introduce the notion of knowledge graph that we employ, and formalize the method we use to link concept names that occur in the topic ontology to the knowledge graph for information extraction.

Definition 3.2.1 (Knowledge Graph)**.**
We assume a knowledge graph K *to be a triple* $K = (G, \ell, \omega)$*, where* $G = (N, E)$ *is a directed graph with* nodes N *and a set* E *of connecting* edges*. Nodes are labeled through the* labeling function ℓ *such that* $\ell(n)$ *is a short* description *of the* entity n*. Edges* $e \in E$ *are labeled by* relation types $r \in R$*, where* R *is a fixed finite set defined by* K*. We presume that at least a kind of subsumption relation (is-a) exists.*

If there is an edge $e = (n, n') \in E$ *with label* $\ell(e) = r$*, we call the triple* $(n, r, n') \in K$ *an* assertion*, in which* $n \in N$ *is the* start entity*, and* $n' \in N$ *the* end entity*. Assertions are a priori assumed to be non-symmetric and non-transitive. A* weight function ω *assigns an associated weight* $\omega(a) \in \mathbb{R}$ *to each assertion* a *that measures connection strength, respectively correctness.*

Remark.
With this definition a taxonomy graph $\mathcal{G}_{\text{Tax}}$ can itself be viewed as a minimal knowledge-graph with one relation type $R = \{\sqsubseteq\}$. Each node $n \in N_C(\text{Tax})$ is then identified with $\ell(n) = n$.

Definition 3.2.2 (Request)**.**
Let $n \in N$ *be an entity and* $r \in R$ *a relation type in* K*. We define* requests *of three types (s=start, e=end, b=bidirectional) over* K*:*

$$\begin{aligned} Req_e(n, r) &= \{(n', \omega(a)) \mid a = (n, r, n') \in K\} \\ Req_s(n, r) &= \{(n', \omega(a)) \mid a = (n', r, n) \in K\} \\ Req_b(n, r) &= Req_e(n, r) \cup Req_s(n, r) \end{aligned} \tag{3.3}$$

3.2.1 Suggestions

Suggestions are extracted from the entity labels of the knowledge graph and are intended to help during the ontology construction process. We define a *filtered request* $\widetilde{Req}(n, \mathcal{R}e\ell)$ for a given set $\mathcal{R}e\ell$ of pairs (r, t), where r is a relation type and t a request type, as the union over multiple requests filtered by a *threshold function* Θ. The function Θ guarantees that results that originate from assertions with high weights are favored while limiting suggestions to a manageable amount.

Definition 3.2.3 (Filtered Request)**.**
Let $n \in N$ *be a entity from* K *and* $\mathcal{R}e\ell = \{(r, t) \mid r \in R \text{ and } t \in \{s, t, b\}\}$*. For a given*

threshold function Θ *we define*

$$\widetilde{Req}(n, \mathcal{R}e\ell) = \bigcup_{(r,t)\in\mathcal{R}e\ell} \{(n', \omega(a)) \in Req_t(n, r) \mid \omega(a) \geq \max_{(r,t)\in\mathcal{R}e\ell} \Theta(Req_t(n, r))\} \quad (3.4)$$

From a filtered request we obtain a set of concept names that we call *suggestions*. A naming function *onto* converts the labels into a more concise form, so that they can be immediately selected as concept names to build axioms. In what follows we introduce a naming function and provide some details on the threshold function Θ.

3.2.1.1 The Naming Function *onto*

The *naming function* is a conversion function between K and the ontology vocabulary.

Definition 3.2.4 (Naming Function)**.**
A naming function onto *is a function that takes a node* n *in* K *and returns a concept name* $A \in N_C$*.*

Different conversion techniques are possible, dependent on the entity encoding of the knowledge graph. We assume the knowledge graph to have representative node labels, so that these can be utilized by *onto*, but any other transformation method that generates matching concept names serves the purpose.

If labels are frequently composed of a few words, a conversion with *onto* could apply CamelCase to receive concept names, as follows. For each $n \in K$ the label $\ell(n)$ is extracted and the subsequent steps are applied: at first all articles are removed; then all individual words are capitalized; at last white spaces are removed. Following this method, $\ell(n) =$ "A romance novel" is converted to $onto(n) =$ RomanceNovel. In fact this is the method that was implemented in the *CN2TopicOnto* tool.
We want to note as a downside that it is possible for *onto* to be surjective, if distinguishing filler words are removed from the labels during the process.

3.2.1.2 The Threshold Function Θ

In accordance with Definition 3.2.2 a request returns a set of tuples $(n, \omega(a))$ where ω is the weight of the underlying assertion a. After a request is sent, Θ is applied to remove elements with low weight in cases where a lot of elements with high weight exist. Cut-off points are added at selected limits to ensure that, if existent, at least a certain amount of results are retained by doing the following:

The function Θ assigns a real number to each individual request $Req_t(n, r)$ for an entity n and a request type r, and $t \in \{s, t, b\}$, based on a set of fixed pairs (k_i, m_i) where $k_i \in \mathbb{R}$ define *weight cut-off points* and $m_i \in \mathbb{Z}$ define respective *cardinality limits*.
If $Req_t(n, r)$ returns a set of tuples $(n, \omega(a))$ then $|\{(n, \omega(a)) \mid \omega(a) \geq k_i\}|$ is the cardinality of the subset of $Req_t(n, r)$, whose elements' weight is greater (or equal) to k_i. If this cardinality is greater (or equal) to the respective m_i, then the assumption is that the

request returns enough results of high confidence (according to ω) to disregard those with lower ω-values. As a result all n with ω lower than k_i are cut off.
The function Θ assigns the maximum k_i, for which the subset cardinality exceeds the cardinality limit. This assures that assertions with high weight are valued over assertions with low weight, but also guarantees a certain number of results.

An example of a very simple threshold would be,

$$\Theta(Req_t(n,r)) = \begin{cases} 1 & if\ |\{(n',\omega(a)) \mid \omega(a) \geq 1\}| \ \geq 3 \\ 0 & else \end{cases} \tag{3.5}$$

with a single pair $(1,3)$ assigned. In this example, if more than three respective assertions that contain n' and have weight greater or equal to one exist, then the assigned Θ is 1. As a result all assertions that have lower weight than 1 will be cut-off and not considered as relevant. If less than three assertions with weight greater or equal to one exist, then the threshold is set to 0 and all assertions are included (assuming the knowledge graph assigns no negative weights).

The pairs (k_i, m_i) need to be selected based on the weight distribution of the knowledge graph. If m_i increases with increasing k_i then more results with high confidence are needed to increase Θ, whereas if m_i decreases with increasing k_i, results with low confidence are more likely to be cut off. However in the second case, constraints can easily become too restrictive, such that high confidence results are required to retain any results at all. For this reason a constraint might be desirable to ensure that some results are kept in any case.
The function and specific values (k_i, m_i) used in our implementation are explained in section 5.3.1 together with other specifics of ConceptNet.

We now formally define $\mathcal{S}(n, \mathcal{R}e\ell)$ as the set of *suggestions*:

Definition 3.2.5 (Suggestions)**.**
Let $\widetilde{Req}(n, \mathcal{R}e\ell)$ be the filtered request for an entity n and a set $\mathcal{R}e\ell$ and onto a naming function. Then

$$\mathcal{S}(n, \mathcal{R}e\ell) = \{onto(n') \mid (n', \omega(a)) \in \widetilde{Req}(n, \mathcal{R}e\ell)\}. \tag{3.6}$$

$\mathcal{S}(n, \mathcal{R}e\ell)$ is a set of concept names that are associated with the entity n through a thematically connected group of relation types, defined in $\mathcal{R}e\ell$. Choosing appropriate sets $\mathcal{R}e\ell$, based on the employed knowledge graph is an important preliminary step to guarantee that good suggestions can be extracted.

Figure 3.4 shows an example of a filtered request using $\hat{\omega}$, the threshold function Θ that is employed in *CN2TopicOnto*, to obtain possible subclasses of the entity *animal* with the subset relation type IsA. For clarity, in this example we identify a node n with its label $\ell(n)$.

$$
\begin{array}{lcl}
\Theta(\mathit{Req}_s(\text{'animal'}, \mathit{IsA})) & = & 2.0 \\
\widetilde{\mathit{Req}}(\text{'animal'}, \{(\mathit{IsA}, s)\}) & = & \{(\text{'a rabbit'}, 2.82842712474619), (\text{'A beaver'}, 2.0), \\
 & & (\text{'A ferret'}, 2.0), (\text{'A primate'}, 2.0), (\text{'a rodent'}, 2.0), \dots\} \\
\mathcal{S}(\text{'animal'}, \{(\mathit{IsA}, s)\} & = & \{\ \mathsf{Beaver}, \mathsf{Ferret}, \mathsf{Primate}, \mathsf{Rabbit}, \mathsf{Rodent}, \dots\}
\end{array}
$$

Figure 3.4: Requests to obtain suggestions for subclasses with $\ell(n) =$ 'animal'

3.2.2 Linking the Ontology with the Knowledge Graph

Any concept name that is suggested through $\mathcal{S}(n, \mathcal{R}e\ell)$ for some $n, \mathcal{R}e\ell$ can be added to the ontology. We define a mapping that, if a concept name is added to $N_C(\mathcal{O})$, links it back to the respective knowledge graph node; the node with whose label it was created by *onto*.

Definition 3.2.6 (KG-Mapping)**.**
Let $\mathcal{X}$ be a topic ontology and $N_C(\mathcal{X})$ the set of its atomic concepts.
A bijective mapping ϕ from a subset $\widetilde{N_C} \subseteq N_C(\mathcal{X})$ to entities $\tilde{N} \subseteq N$ of K is called a K-mapping.

In a KG-mapping $A \in N_C(\mathcal{X})$ and $\phi(A)$ are assumed to share a similar semantic meaning.

We now extend Definition 3.1.1 of a topic ontology to incorporate the K-mapping.

Definition 3.2.7 (Topic Ontology over K)**.**
A topic ontology over K *is a triple $\mathcal{X} = \langle \mathcal{T}, tc, \phi \rangle$, where $\langle \mathcal{T}, tc \rangle$ has been extended with a KG-mapping ϕ such that*

(a) tc has to occur in $\mathcal{T}$
(b) $\phi(tc)$ is defined

For the construction, the topic ontology is initialized with $\mathcal{T} = \{tc \sqsubseteq \top\}$ and some associated $\phi(tc)$. Then requests for $\phi(tc)$ are sent to the knowledge graph to successively receive suggestions that are conceptually related to tc. The more connected tc is within the knowledge graph, the more concept suggestions can be offered. If a suggestion is added to the ontology, new requests are sent to receive respective further suggestions. This way the ontology is expanded in an iterative fashion.

3.2.3 The Central Taxonomy

To allow for an expansion of the ontology in a structured way, a few further definitions are necessary.

Definition 3.2.8 (Central Taxonomy)**.**
Assume a topic ontology $\mathcal{X} = \langle \mathcal{T}, tc, \phi \rangle$ and its taxonomy $\mathrm{Tax}(\mathcal{X})$.
We consider a subset $\mathcal{CT} \subseteq$ Tax of ACIs where

(a) $tc \in N_C(\mathcal{CT})$
(b) $\mathcal{G}_{\text{Tax}}$ *is connected*
(c) $\phi(A)$ *is defined for every* $A \in N_C(\mathcal{CT})$

This set is called the central taxonomy *of* $\mathcal{X}$*. We call the set of atomic concepts* $N_C(\mathcal{CT})$ *the* central concepts $\mathcal{CC}$*.*

The term expresses the notion that all concepts in $\mathcal{CC}$ are centered around the topic concept. As such the central concepts are assumed to have close semantic resemblance to the topic concept *tc*. Intuitively the *tc* represents the semantical "center", as it is surrounded by concepts with more general meaning and concepts that are more restrictive.

Building an initial central taxonomy will be the first step of the topic ontology construction process. The central taxonomy builds the initial 'skeleton' of Tax and will later be expanded with further axioms. We introduce a queue representation of the taxonomy that will be useful for the structured navigation of $\mathcal{CT}$.

3.2.3.1 A Flat Representation of the Taxonomy

The *iteration order queue (IOQ)* is a limited representation of the taxonomy structure, with the only requirement that all ancestors of a concept A get listed previous to A. Algorithm 3.1 describes the queue building process for an arbitrary ontology. We use it on $\mathcal{CT}$ of a topic ontology $\mathcal{X} = (\mathcal{T}, tc, \phi)$ to generate a queue for the contained central concepts. We refer to this queue as $Q_{\mathcal{CT}}(\mathcal{X})$.

The algorithm starts at level 1 and collects all concepts that have no further ancestors. Then with each queue update, the first element is popped, while a subset of its children is added to the queue. Added subsets only include those concepts that were not processed yet and whose parents were all already appended to the queue in a previous step.
If the respective taxonomy graph is connected then the queue will always list concepts with more general semantic meaning previous to concepts that describe a more specific set of individuals.

If the initial central taxonomy is very large, navigation becomes exhaustive. As a mitigation we define a smaller subset of selected central concepts $\mathcal{CC}_{sel}$ that are contained in $\mathcal{CT}$ but do not branch out as far in $\mathcal{G}_{\text{Tax}}$. Those concepts will always be offered to the user to use in axioms.

Definition 3.2.9 (Selected Central Concept)**.**
Let $\mathcal{CT}(X)$ *be the central taxonomy of* $\mathcal{X} = (\mathcal{T}, tc, \phi)$*.*
$A \in \mathcal{CC}_{sel} \Leftrightarrow$

(a) $A \sqsubseteq tc$
(A is an immidate subclass of tc)
(b) $tc \sqsubseteq^*_{\mathcal{CT}(\mathcal{X})} A$
(A is an ancestor of tc)

Algorithm 3.1: Iteration Order Queue algorithm

Input: A subset $\mathcal{T}'$ of Tax($\mathcal{O}$) of an ontology $\mathcal{O}$
Output: An ordered queue $\mathcal{R}$, in which all ancestors of a concept $A \in N_C(\mathcal{T}')$ appear previous to A

$\mathcal{Q} = \{\ \}$;
$\mathcal{R} = \{\ \}$;
if $\mathcal{Q} = \{\ \}$ **then**
  /* Ancestors(x) returns all ancestors of x in $\mathcal{T}'$ */
  forall $A \in N_C(\mathcal{T}')$ with Ancestors(A) $= \{\ \}$ **do**
    $\mathcal{Q} = \mathcal{Q} \cup \{A\}$;
  end
end
while $\mathcal{Q} \neq \{\ \}$ **do**
  $\ell = \mathcal{Q}$.popleft( );
  $\mathcal{R} = \mathcal{R} \cup \{\ell\}$;
  add = true;
  /* SubClasses(x) returns all subclasses of x in $\mathcal{T}'$ */
  forall $x \in$ SubClasses(ℓ) **do**
    /* SuperClasses(x) returns all superclasses of x in $\mathcal{T}'$ */
    forall $y \in$ SuperClasses(x) **do**
      if $y \neq \ell$ **and** $y \notin \mathcal{Q} \cup \mathcal{R}$ **then**
        add = false;
        break;
      end
    end
    if add == true **then**
      $\mathcal{Q} = \mathcal{Q} \cup \{x\}$;
      /* Equiv(x) returns all concepts equivalent to x in $\mathcal{T}'$ */
      forall $z \in$ Equiv(x) **do**
        $\mathcal{Q} = \mathcal{Q} \cup \{z\}$;
      end
    end
  end
end

(c) $\exists A'$ *s.t.* $A' \sqsubseteq tc$ *and* $A' \sqsubseteq^*_{\mathcal{CT}(\mathcal{X})} A$
(A is an ancestor of an immediate subclass of tc)
(d) $\exists B'$ *s.t.* $tc \sqsubseteq^*_{\mathcal{CT}(\mathcal{X})} B'$ *and* $A \sqsubseteq B'$
(A is an immediate subclass of an ancestor of tc)

The importance of $\mathcal{CC}_{sel}$ will become clearer when the complete topic ontology building process is described in the next chapter.

For now we have introduced all necessary terminology. Equations (3.7) and (3.8) recap the new denominations and their relation. Vertical alignment indicates the connection between sets of terminological axioms (3.7) and contained vocabulary (3.8).

Remark. Let $\mathcal{X} = (\mathcal{T}, tc, \phi)$.

$$\mathcal{CT}(\mathcal{X}) \subseteq \text{Tax}(\mathcal{X}) \subseteq \mathcal{T} \tag{3.7}$$

$$tc \in \mathcal{CC}_{sel} \subseteq \mathcal{CC} \subseteq \text{N}_C(\text{Tax}) \subseteq \text{N}_C(\mathcal{X}) \tag{3.8}$$

Selected parts of Tax of the Animal topic ontology. **Bold text** indicates concept names that are connected to tc in $\mathcal{G}_{\text{Tax}}$. We will call this part of Tax the central taxonomy in the future (see Definition 3.2.8).

Figure 3.1: $\mathcal{G}_{\text{Tax}}$ of selected parts of the Animal Ontology.

$\mathcal{T} = \{$**Animal** $\sqsubseteq$ **Creature**, **HerdAnimal** $\sqsubseteq$ **Animal**, **HoofedAnimal** $\sqsubseteq$ **Animal**, **WarmBloodedAnimal** $\sqsubseteq$ **Animal**, **Mammal** $\sqsubseteq$ **Animal**, **Crustacean** $\sqsubseteq$ **Animal**, **Elephant** $\sqsubseteq$ **HerdAnimal**, **Cows** $\sqsubseteq$ **HerdAnimal**, **Elephant** $\sqsubseteq$ **Mammal**, **Cattle** $\sqsubseteq$ **Mammal**, **Cows** $\sqsubseteq$ **HoofedAnimal**, **Cows** $\sqsubseteq$ **Cattle**, **Bull** $\sqsubseteq$ **Cows**, **Cow** $\sqsubseteq$ **Cattle**, **Crab** $\sqsubseteq$ **Crustacean**, **Bird** $\sqsubseteq$ **WarmBloodedAnimal**, **Bird** $\sqsubseteq$ **Biped**, AnimalSpecies $\sqsubseteq$ Species, ArthropodSpecies $\sqsubseteq$ AnimalSpecies, BirdSpecies $\sqsubseteq$ AnimalSpecies, MammalSpecies $\sqsubseteq$ AnimalSpecies, LandBody $\sqsubseteq$ Habitat, WaterBody $\sqsubseteq$ Habitat, Forest $\sqsubseteq$ LandBody, BeachArea $\sqsubseteq$ LandBody, TropicalForest $\sqsubseteq$ Forest, RainForest $\sqsubseteq$ Forest, MammaryGland $\sqsubseteq$ AnatomicalPart, AnatomicalPart $\sqsubseteq$ BodyPart, BodyPart $\sqsubseteq$ Part$\}$

Figure 3.2: Selected parts of Tax of the Animal ontology.

CHAPTER 4

Ontology Construction

In this chapter we look more closely at the actual process of constructing a topic ontology. We start with a general overview of the process and its basic steps, and list all the axiom types that are generated. Later sections describe each individual step in detail and take the reader through the complete construction procedure. Finally, in Sections 4.3.2 to 4.3.5 we illustrate the construction process of all the different types of supported axioms with the help of examples.

4.1 Overview

We assume to have a common-sense or general-knowledge knowledge graph K, as defined in Definition 3.2.1, with labeled node entities denoted as $n \in N$. Semantic assertions a link two entities through one of multiple relation types $r \in R$ to form the contained knowledge. By exploiting assertions, our algorithm constructs a topic ontology $\mathcal{X} = (\mathcal{T}, tc, \phi)$ on a general-knowledge topic, which is defined by the seeding concept name tc. The process is semi-automatic, in the sense that suggestions for concept names are extracted and refined automatically, but the user has to confirm and build the axioms that are then added to $\mathcal{T}$. As a result $\mathcal{T}$ contains axioms which convey common-sense knowledge about the topic domain and permit further reasoning with standard DL algorithms.

Axioms are defined in a stepwise process. Requests $Req_t(n, r), t \in \{s, e, b\}$ to K extract those entities n' that form assertions $a = (n, r, n')$ and hold information related to the topic. As detailed in section 3.2.1 the naming function *onto* subsequently transforms entity labels into sets of suggestions $\mathcal{S}(n, \mathcal{R}e\ell)$. Suggestions are grouped into sets $\mathcal{R}e\ell$ based on different subject areas and assist during the ontology construction process. The mapping ϕ is automatically expanded as concept names in $\mathcal{X}$ are linked to respective K entities for iterative information extraction. If an axiom is defined it is added to $\mathcal{X}$, and all associated concept, individual and role names are created as part of the effective vocabulary.

The workflow we propose resembles the process of hand-engineering ontologies in which knowledge elicitation is heavily supported by the suggestions. The KG-mapping offers the possibility to generate onward suggestions and explore the knowledge graph further whenever concept names are added as central concepts.

4.1.1 Supported Axioms

Suggestions can be used to formulate different types of axioms. At the moment the following axiom types are supported for a topic ontology $\mathcal{X} = (\mathcal{T}, tc, \phi)$:

$$A_1 \sqsubseteq B_1 \text{ or } B_1 \sqsubseteq A_1 \tag{AxSub}$$

$$A_1 \equiv A_2 \tag{AxEquiv}$$

$$A_1 \sqsubseteq \bigsqcup_k B_k \sqcup \bigsqcup_j \exists R_j.(B_{j_1} \sqcup \ldots \sqcup B_{j_n}) \tag{AxDisjun}$$

$$A_1 \sqsubseteq \forall \mathsf{HasPart}.A_1\mathsf{Part} \tag{AxAllPart}$$

$$\exists R^{-}.\top \sqsubseteq \bigsqcap_i A_i \tag{AxRan}$$

$$A_1 \sqcap A_2 \sqsubseteq \bot \tag{AxDisj}$$

where A_i are atomic concepts and B_i are basic concepts, i.e. atomic or nominals.

A deliberate choice was made to only support a limited selection of axiom types that seem natural for capturing a lot of knowledge contained in a knowledge graph. The permitted axioms include concept inclusions with disjunctions of form (AxDisjun). In such axioms it is possible to have the right-hand-side contain descriptive characteristics that express common-sense knowledge about individual concepts and their properties. To fully support $\mathcal{ALCIO}$ some concept constructors, for example negation ($\neg$), are missing, which is a consequence of knowledge graphs not employing respective semantics. Therefore negated statements are infrequent or inconclusive and we did not consider their inclusion integral for the moment. If needed they can be added later in an ontology editor of choice. Universal axioms are not supported fully, but (AxAllPart) is a custom axiom that can be added for the relation HasPart. Similarly inverses are for the present only captured through the use of $\mathsf{HasPart}^{-} \equiv \mathsf{PartOf}$. Nominal constructions over individuals on the other hand are directly supported, as they are suitable for human-perceived impressions about things, such as physical states (i.e. colors, material), as well as for named entities, which are an integral part of many knowledge graphs. Axioms (AxRan) and (AxDisj) allow for range restrictions and disjointness, respectively, which is important for engineering conclusive ontologies.

4.1.2 Construction Process

The building process of $\mathcal{X} = (\mathcal{T}, tc, \phi)$ consists of three consecutive steps:

Step 1: Building the Central Taxonomy (see Section 4.2)
$N_C(\mathcal{X})$ is initialized with tc. Then $\mathcal{CT}$, the central taxonomy linked to tc,

is created as an initial subset of $\mathcal{T}$. To do this axioms of types (AxSub) or (AxEquiv) are added by selecting from sets of suggestions to determine suitable sub- and superclasses. Inapplicable suggestions can be saved for later use.

Step 2: Adding complex GCIs (see Section 4.3)
In four consecutive substeps, *C1*,*C2*,*C3* and *C4*, $\mathcal{T}$ is expanded. The first three focus on relations which are likely to be relevant in several domains and which form thematic contexts. Our approach incorporates the following three contexts:

- *C1* *part-whole relations*: meronym relations of the form *"X is part of Y"* and *"Y has X as part"*;
- *C2* relations that indicate typical *locations*, either geographical or functional; and
- *C3* active or passive object *capabilities*: a range of suggestions for applications and defining characteristics, such as colors or shapes.

Suggestions $\mathcal{S}(n, \mathcal{R}e\ell)$, for some n are proposed for each context and the user can create axioms, such as (AxAllPart). Dynamic extensions allow for a quick way to add axioms (AxSub) and introduce individuals. Under specific conditions axioms (AxRan) and (AxDisj) are automatically added.

In the last substep,

- *C4* suggestions are formed from *generic relation types* and are not thematically restricted.

Suggestions that were saved for later use during Step 1 are available to be used.
In this step the creation of axioms is not limited. This allow the user to add axioms of his own choice and to personalize the ontology.

Step 3: Clean-Up (see Section 4.4)
A postprocessing step that removes redundant relations and allows to further modify and restructure the taxonomyof $\mathcal{T}$. In particular it is possible to add ACIs, (AxSub) and (AxEquiv), between concept names added in Step 2, and introduce new concept names into the taxonomy.

During Step 1 an initial $\mathcal{CT}$ is constructed as the skeleton of $\mathcal{T}$. Afterwards the queue $Q_{\mathcal{CT}}$ is utilized to iterate through the concepts that were previously added to $\mathcal{CT}$. This way in Step 2 suggestions are presented for one concept $\tilde{A} \in \mathcal{CC}$ at the time. The user can add axioms that contain $\tilde{A}$ by selecting and completing axiom patterns. This way the knowledge about $\tilde{A}$ is *expanded*. It is possible to immediately add ACIs for new concept names and appropriately place them into the taxonomy. When needed, individuals and nominals can be created on the spot. We will subsequently refer to these forms of ontology

expansion as *dynamic extensions*. Through dynamic extensions axioms are created that relate to $\tilde{A}$ more loosely. We consider those to be axioms that do not contain $\tilde{A}$ itself. We will continue to use $\tilde{A}$ to refer to the atomic concept that is currently dealt with.

By enforcing the presentation order induced by $Q_{\mathcal{CT}}$, an incentive is given to specify axioms that hold for more general concepts first. Those will then be naturally inherited by all existing subclasses.
If $\mathcal{CT}$ is large, it becomes infeasible to propose every $\tilde{A} \in \mathcal{CC}$. As a remedy $\mathcal{CC}_{sel}$ defines the set of concepts that are considered more important for expansion. This way suggestions for deeply nested concepts from $\mathcal{CC}$ are not proposed by default. If a proposed $\tilde{A} \in \mathcal{CC}$ has descendants that are in $\mathcal{CC} \backslash \mathcal{CC}_{sel}$, they can be navigated to, one level at a time, to request corresponding suggestions.

The construction of the central taxonomy and expansion with axioms follows a similar structure to the editing process that Protegé offers with its Asserted Class Hierarchy [1] and Class Descriptions[2]. Through the taxonomy a class hierarchy is defined that can optionally be extended whenever new concept names are added. Each time a concept name $\tilde{A}$ is proposed, axioms can be added that contain $\tilde{A}$, similar to the Class Descriptions that associate further axioms with each concept name.

The example in Figure 4.2 illustrates the construction flow and the different steps that are involved. For the sake of the example the presentation and selection of suggestions is not shown. The selection of suggestions and creation of axioms will be explained in detail in the next sections.

4.2 Step 1: Building the Central Taxonomy

The central taxonomy is the main part of the ontology concept name hierarchy and comprises concept names that are strongly correlated to the topic. By Definition 3.2.8, $\mathcal{CT}$ includes the topic concept tc and contains exclusively ACIs $A_1 \sqsubseteq A_2$ for $A_1, A_2 \in N_C$.

Upon creation, the ontology $\mathcal{X}$ is initialized with $N_C = \{tc\}$ which implies $\mathcal{T} = \{tc \sqsubseteq \top\}$. The mapping sets $\phi(tc) = n$ where n is the entity in K whose label $\ell(n)$ can be identified with tc. ACIs are then iteratively built based on tc by extracting suggestions for suitable sub- and superclasses from K. To achieve an adequate hierarchical structure for the initial central taxonomy two separate substeps are needed, in which at first a provisional taxonomy is created and subsequently modified to refine the structure.

4.2.1 A provisional 5-level Taxonomy

During the first step the relation type $r_{is-a} \in R$ that defines the is-a connection in K is used. It should represent a hyponym-hyperonym or subsumption relation. With this relation type we define the first two sets $\mathcal{R}e\ell_{\sqsubseteq} = \{(r_{is-a}, s)\}$ and $\mathcal{R}e\ell_{\sqsupseteq} = \{(r_{is-a}, e)\}$

[1] http://protegeproject.github.io/protege/views/class-hierarchy/
[2] http://protegeproject.github.io/protege/views/class-description/

Minimal example of the construction of $\mathcal{X} = (\mathcal{T}, \text{Fruit}, \phi)$. **Bold text** indicates $\mathcal{CT}$ and respective concept names. Underlined text indicates Tax.

- Initialization with tc =Fruit:
  $N_C(\mathcal{X}) = \{\text{Fruit}\}$
  $N_R(\mathcal{X}) = N_I(\mathcal{X}) = \{\ \}$
  $\mathcal{T} = \{\ \}$

- Building the central taxonomy $\mathcal{CT}$ (Step 1):
  $N_C(\mathcal{X}) = \{\text{Produce, Fruit, Lemon, Apple, Banana, CitrusFruit, GoldenDeliciousApple}\}$
  $N_R(\mathcal{X}) = N_I(\mathcal{X}) = \{\ \}$
  $\mathcal{T} = \text{Tax} = \mathcal{CT} = \{$**Fruit** $\sqsubseteq$ **Produce**, **Apple** $\sqsubseteq$ **Fruit**, **Banana** $\sqsubseteq$ **Fruit**, **Lemon** $\sqsubseteq$ **CitrusFruit**, **CitrusFruit** $\sqsubseteq$ **Fruit**, **GoldenDeliciousApple** $\sqsubseteq$ **Apple**$\}$

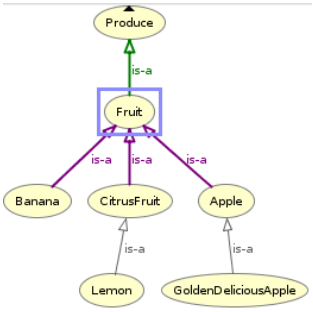


Figure 4.1: $\mathcal{G}_{\mathcal{CT}}(\mathcal{X})$

- Calculating the iteration order queue:
  $Q_{\mathcal{CT}} = \{$Produce, Fruit, Apple, Banana, CitrusFruit, (GoldenDeliciousApple), (Lemon)$\}$
  Indicated by the brackets, GoldenDeliciousApple and Lemon are not part of $\mathcal{CC}_{sel}$ and will only be proposed if requested.

- Traversing $Q_{\mathcal{CT}}$ and adding axioms for each $\tilde{A} \in \mathcal{CC}$ (Step 2):

  Produce $\sqsubseteq \exists$BoughtAt.Supermarket
  FruitJuice $\sqsubseteq \exists$MadeFrom.**Fruit**, FruitJuice $\sqsubseteq$ Juice (dynamic extension: subclass)
  AppleJuice $\sqsubseteq \exists$MadeFrom.**Apple**, **Apple** $\sqsubseteq \exists$Contains.Fibre
  Banana $\sqsubseteq \exists$Contains.Sugar
  CitrusFruit $\sqsubseteq \exists$HasTaste.{sour}, $\{sour\} \sqsubseteq$ Taste (dynamic extension: individual)

- Modifying the taxonomy (Step 3):
  The atomic inclusions Sugar $\sqsubseteq$ FoodComponent and Fibre $\sqsubseteq$ FoodComponent are added.
- The final result:
  $N_C(\mathcal{X}) =$ {Produce, Fruit, Lemon, Apple, Banana, CitrusFruit, GoldenDeliciousApple, Supermarket, FruitJuice, AppleJuice, Juice, Fibre, Sugar, Taste}
  $N_R(\mathcal{X}) =$ {BoughtAt, MadeFrom, Contain, HasTaste}
  $N_I(\mathcal{X}) =$ {sour}
  $\mathcal{T} =$ {**Fruit $\sqsubseteq$ Produce**, **Apple $\sqsubseteq$ Fruit**, **Banana $\sqsubseteq$ Fruit**, **Lemon $\sqsubseteq$ CitrusFruit**, **CitrusFruit $\sqsubseteq$ Fruit**, **GoldenDeliciousApple $\sqsubseteq$ Apple**, FruitJuice $\sqsubseteq$ Juice, Sugar $\sqsubseteq$ FoodComponent, Fibre $\sqsubseteq$ FoodComponent, {sour} $\sqsubseteq$ Taste, Produce $\sqsubseteq \exists$BoughtAt.Supermarket, FruitJuice $\sqsubseteq \exists$MadeFrom.Fruit, AppleJuice $\sqsubseteq \exists$MadeFrom.Apple, Apple $\sqsubseteq \exists$Contain.Fibre, Banana $\sqsubseteq \exists$Contain.Sugar, CitrusFruit $\sqsubseteq \exists$HasTaste.{sour}}

Figure 4.2: Example of the process flow illustrating Step 1 to Step 3 during the construction of the Fruit ontology.

that generate suggestions for possible sub- and superclasses, respectively. Additionally we define a set $\mathcal{R}e\ell_{\cong}$ that generates suggestions that might be suitable as either subclass or superclass from a relation $r \in R$. In our implementation we use the ConceptNet relation *RelatedTo* that is used in knowledge graph assertions where no appropriate relation type was identified. Relations such as *AssociatedWith* or *SimilarTo* are possible alternatives.

Every time suggestions are presented, concept names can be selected to create ACIs following choice by index (see Section 5.2.3). Three different choices are possible: 1. select suggestions by choosing wanted suggestions or by deleting unwanted suggestions, 2. choose suggestions to move from sub- to superclass (or vice versa), or 3. add suggestions to the *saved suggestions* for later use. For those suggestions no axioms are added.

The provisional 5-level taxonomy is built as follows:

(a) Suggestions $\mathcal{S}(\phi(tc), \mathcal{R}e\ell_{\sqsubseteq})$ are prompted. 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 $\phi(A) = n$. For each moved A, $tc \sqsubseteq A$ is added instead.

(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 $\phi(A) = n$. For each moved A, $A \sqsubseteq tc$ is added instead.

(c) Next, suggestions in $\mathcal{S}(\phi(tc), \mathcal{R}e\ell_{\cong})$ are prompted and two selections are possible: choose those to treat as in in (a), and those to treat as in(b).

(d) For each $A_1 \in N_C(\mathcal{CT})$ introduced in items (a) to (c), items (a) and (b) are repeated, adding axioms $A_1 \sqsubseteq A_2$ and $A_2 \sqsubseteq A_1$ for A_2 in $\mathcal{S}(\phi(A_1), \mathcal{R}e\ell_{\sqsupseteq})$ respectively $\mathcal{S}(\phi(A_1), \mathcal{R}e\ell_{\sqsubseteq})$, and extending ϕ accordingly.

In the Fruit example in Figure 4.2 Banana, Lemon and Apple are among the suggested subclasses $\mathcal{S}(\phi(\mathsf{Animal}), \mathcal{R}e\ell_{\sqsubseteq})$, while Produce is recommended as superclass through $\mathcal{S}(\phi(\mathsf{Animal}), \mathcal{R}e\ell_{\sqsupseteq})$. GoldenDeliciousApple is then proposed as subclass of Apple by $\mathcal{S}(\phi(\mathsf{Apple}), \mathcal{R}e\ell_{\sqsubseteq})$ as suggestions for existing concept names are recursively generated in (d). Analogously CitrusFruit is suggested as a superclass for Lemon by $\mathcal{S}(\phi(\mathsf{Lemon}), \mathcal{R}e\ell_{\sqsupseteq})$. The axiom $\mathsf{Fruit} \sqsubseteq \mathsf{CitrusFruit}$ is not directly induced through the suggestions, but can be added in the subsequent modification step that is described in Section 4.2.2.

In case a cyclic dependency is created within Tax, the cycle is recognized and after additional confirmation all concept names that take part in the cycle are explicitly made semantically equivalent by replacing all subsumptions with respective equivalences $A_1 \equiv A_2$ (AxEquiv). If the user does not confirm the detected cycle, the inclusion that was added second is retracted.

For the general case we assume that concept names are all different and that at least one ACI is added in items (a) to (d). Furthermore the resulting taxonomy is assumed to be acyclic.
Under those assumptions, the process corresponds to an upwards and downwards expansion of Tax where for each new concept A there exists an undirected path with at most length two to tc in $\mathcal{G}_{\text{Tax}}$. Thereby the result of the first step is a provisional taxonomy with five levels in which the center concept is located on the third level. An illustration of the five level taxonomy is depicted in Figure 4.3, which shows a larger taxonomy on vehicles.

4.2.2 Modification

The modification step tries to enforce a narrower and deeper structure on $\mathcal{G}_{\mathcal{CT}}$ by encouraging the user to add additional ACIs to $\mathcal{CT}$ that enhance the existent information and reduce the number of concepts on the topmost level.
It is reasonable to assume that the information in the underlying knowledge graph is incomplete. Hence concepts that are situated on one level of the provisional taxonomy will not semantically belong to the same hierarchical level, because important connections are not captured by K. To illustrate this using the example in Figure 4.2, the set of suggestions $\mathcal{S}(\phi(\mathsf{Lemon}), \mathcal{R}e\ell_{\sqsupseteq})$ includes CitrusFruit and Fruit, while $\mathcal{S}(\phi(\mathsf{CitrusFruit}), \mathcal{R}e\ell_{\sqsupseteq})$ does not include Fruit, despite a citrus fruit obviously being a fruit.
To correct this, this step allows to modify the provisional result by extending $\mathcal{T}$ with additional ACIs between the available concept names in $\mathcal{CT}$. Through this process $\mathcal{CT}$ is restructured and additional levels are created.

In the mentioned case the user can decide to expand $\mathcal{T} = \{\mathsf{Fruit} \sqsubseteq \top, \mathsf{Lemon} \sqsubseteq \mathsf{Fruit}, \mathsf{Lemon} \sqsubseteq \mathsf{CitrusFruit}\}$ with $\mathsf{CitrusFruit} \sqsubseteq \mathsf{Fruit}$. The ACI $\mathsf{Lemon} \sqsubseteq \mathsf{Fruit}$ is removed, as it is now redundant, and Lemon is thereby moved from level 1 to level 2 w.r.t. Fruit.

A complete example of a provisional taxonomy with $tc =$Vehicle, and the final result after modification are illustrated in Figure 4.5.

For the modification step Algorithm 4.1 is applied to $\mathrm{Tax}(\mathcal{X})$ for tc. The algorithm iterates over each named concept $A \in N_C(\mathcal{CT})$ and collects for each A a set $\mathcal{S}_A$ of suggestions for possibly missing inclusions. S_A contains all B, B', such that $A \sqsubseteq^*_{\mathcal{T}} B$ and $A \sqsubseteq^*_{\mathcal{T}} B'$ (all ancestors of A), but neither $B \sqsubseteq^*_{\mathcal{T}} B'$ nor $B' \sqsubseteq^*_{\mathcal{T}} B$ (B and B' occur on different branches of $\mathcal{G}_{\mathcal{CT}}$). It is possible that $B \sqsubseteq^*_{\mathcal{T}} B'$ for some $B, B' \in \mathcal{S}_A$, as constraints are caclulated for each pair B, B' at a time. For each pair $(B, B') \in S_A \times S_A$ selected by the user, $B \sqsubseteq B'$ is added to $\mathcal{CT}$.
Subsets $\mathcal{S}_{A'}$ of $\mathcal{S}_A$ for some A that was already dealt with are not proposed again. Not that sets may still differ in one concept name only, which is the one that induces new potential axioms. All ACIs, where it holds that $\mathcal{X} \models B \sqsubseteq B'$ for $B, B' \in \mathcal{S}_A$ for the current $\mathcal{S}_A$ are listed in addition, so that the user knows that they are already in $\mathcal{T}$ and need not be added again. Every time a new ACI is added, lines 21-23 in Algorithm 4.1 make sure that ACIs that follow from reasoning are removed to minimize the redundancy in Tax for presentation.

After all axioms are added the reasoner HermiT is run once. This step is not necessary, but provides for a nicer graphical representation of Tax in Protegé or similar tools. The resulting $\mathcal{T}$ is now considered the initial taxonomy and all contained atomic concepts build the basis for further axiom specifications.

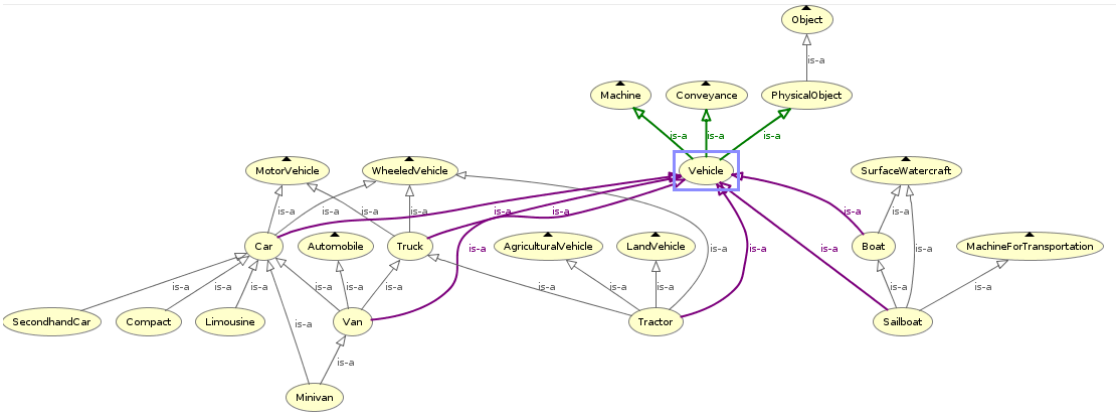


Figure 4.3: A 5-level taxonomy before modification.

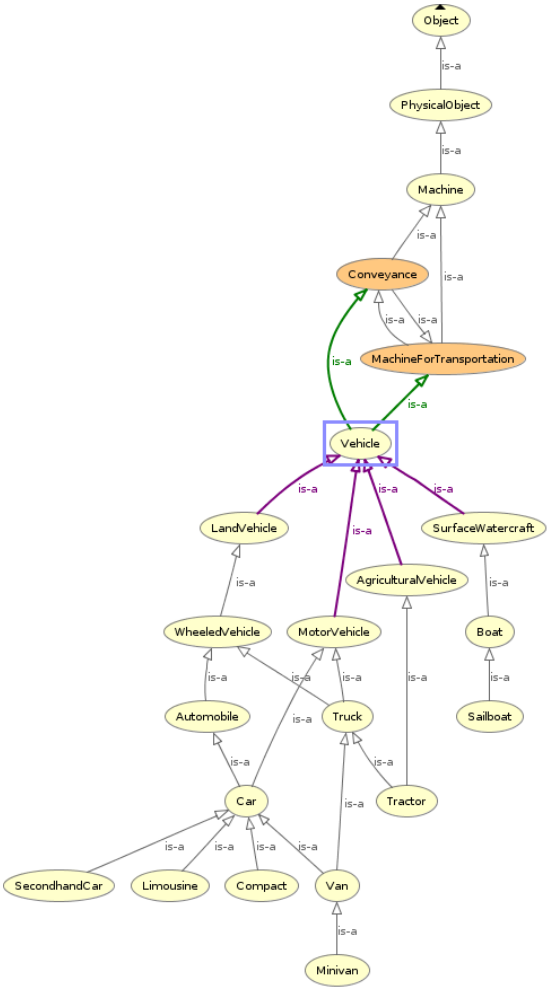


Figure 4.4: The same taxonomy after modification and reasoning.

Figure 4.5: A comparison between a taxonomy graph before and after modification. Both figures show the same selected part of a topic ontology created with *CN2TopicOnto* with the topic concept Vehicle.

Algorithm 4.1: Taxonomy Modification algorithm

Input: Tax($\mathcal{O}$) of an ontology structure $\mathcal{O}$ and an atomic concept C
Output: A modified Tax$'(\mathcal{O})$ such that Tax$'(\mathcal{O}) \models A \sqsubseteq B$ if Tax$(\mathcal{O}) \models A \sqsubseteq B$ for atomic concepts A and B.

$\mathcal{P}_{saved} = \{\ \}$
forall $A \in$ SubClasses$(C) \cup$ SuperClasses$(C) \cup \{C\}$ **do**
  $\mathcal{S}_A = \{\ \}$
  if |Ancestors(A)|≥ 2 **then**
    $\mathcal{S}_A = \{A\}$;
    get all combinations $(B_1, B_2), B_1 \neq B_2$ with $B_1, B_2 \in$ Ancestors(A);
    if $B_1 \notin$ Ancestors(B_2) and $B_2 \notin$ Ancestors(B_1) **then**
      $\mathcal{S}_A := \mathcal{S}_A \cup \{B_1, B_2\}$
    end
    if $|\mathcal{S}_A > 1|$ and $\mathcal{S}_A \nsubseteq \mathcal{S}_{A'}$ for any $\mathcal{S}_{A'} \in \mathcal{P}_{saved}$ **then**
      / print options to the user */*
      print $\mathcal{S}_A$
      forall combinations of $(S_1, S_2) \in \mathcal{S}_A$ **do**
        if $S_1 \in$ Descendants(S_2) **then**
          print "S_1 is already a descendant of S_2"
        end
        if $S_2 \in$ Descendants(S_1) **then**
          print "S_2 is already a descendant of S_1"
        end
      end
      / user choice */*
      ask the user for a ordered pair of concepts $(\tilde{A}, \tilde{B}) \in \mathcal{S}_A$
      / remove redundant ACI */*
      forall $E \in$ Descendants$(\tilde{A}) \cap$ Descendants$(\tilde{B})$ **do**
        Tax := Tax$\backslash\{E \sqsubseteq \tilde{B}\}$
      end
      / add new ACI */*
      if $\tilde{A} \notin$ Descendants$(\tilde{B})$ and $\tilde{B} \notin$ Descendants$(\tilde{A})$ **then**
        Tax := Tax $\cup \{\tilde{A} \sqsubseteq \tilde{B}\}$
      end
      if $\tilde{B} \in$ Descendants$(\tilde{A})$ **then**
        Tax := Tax $\cup \{\tilde{A} \equiv \tilde{B}\}$
      end
    end
  end
  / $\mathcal{P}_{saved}$ stores all lists and makes sure none is presented twice */*
  add $\mathcal{S}_A$ to $\mathcal{P}_{saved}$
end
return Tax

A =Tractor
$\mathcal{S}_A$ = {WheeledVehicle, AgriculturalVehicle, Conveyance, PhysicalObject, LandVehicle, Vehicle, Object, Machine, MotorVehicle, Truck}

The following ACIs are already implied:
Truck $\sqsubseteq$ Conveyance, Vehicle $\sqsubseteq$ Conveyance, Truck $\sqsubseteq$ WheeledVehicle, Truck $\sqsubseteq$ Vehicle, Truck $\sqsubseteq$ MotorVehicle, Truck $\sqsubseteq$ PhysicalObject, Truck $\sqsubseteq$ Object, Truck $\sqsubseteq$ Machine, Vehicle $\sqsubseteq$ PhysicalObject, Vehicle $\sqsubseteq$ Object, Vehicle $\sqsubseteq$ Machine, PhysicalObject $\sqsubseteq$ Object,

User Selections and respective axioms that are added to $\mathcal{T}$:
(AgriculturalVehicle, Vehicle) adds AgriculturalVehicle $\sqsubseteq$ Vehicle,
(LandVehicle, Vehicle) adds LandVehicle $\sqsubseteq$ Vehicle,
(WheeledVehicle, Vehicle) adds WheeledVehicle $\sqsubseteq$ Vehicle,
(WheeledVehicle, LandVehicle) adds WheeledVehicle $\sqsubseteq$ LandVehicle and removes the now redundant WheeledVehicle $\sqsubseteq$ Vehicle
(MotorVehicle, Vehicle) adds MotorVehicle $\sqsubseteq$ Vehicle

Figure 4.6: Modification algorithm example for Figure 4.3

4.3 Step 2: Adding Complex GCIs

During substeps *C1*,*C2*,*C3* suggestions are grouped based on the subject of the knowledge they express. We refer to those substeps as suggesting *general relations*. The term refers to two different notions:

1. a domain-independent relation, such as HasPart or PartOf,
2. a set $\mathcal{R}el$, that expresses one general aspect of the topic domain

An example of item 2 is the set $\mathcal{R}el$ = {*AtLocation, HasLocation, LocatedNear*} of ConceptNet relation types. All associated assertions form triples that capture information about possible locations. However without rigid semantics, the individual assertions often capture substantially different meanings. For example, the assertion (*handbrake, AtLocation, car*) expresses a statement of physical place, while (*SanFrancisco, AtLocation, California)* describes geographical information between two individuals. Even further (*seed, AtLocation, apple*) is a linguistically reasonable statement, but would be better expressed syntactically as a meronym relation (*appleseed, PartOf, apple*). In DL ontologies role names are typically more precise than in knowledge graphs to appropriately handle contextual discrepancies. As meaningful role replacements for the above cases, HasVehicleComponent or HasGeographicalLocation, LocatedInState could be chosen.

The number of relation types in contrast to roles, accounts for one of the fundamental design differences between knowledge graphs and engineered ontologies. Good ontologies

incorporate distinct and fitting role names on a case to case basis. Generic relations are in general not preferential for DL models as they dilute semantics and easily lead to error prone and unmanageable reasoning. To accommodate the creation of distinctive and contextualized role names, we introduced $\mathcal{S}(n, \mathcal{R}e\ell)$ over sets of relation types, where $\mathcal{R}e\ell$ can have more than one element.

The process we propose integrates *part-whole*, *locations* and *capabilities* as three different general relations. Appropriate subsets $\mathcal{R}e\ell$ need to be selected and assigned to each category in advance. We have chosen those three categories based on the premise that they represent information that is typically captured in many knowledge graphs. Additionally associated information is significant in a wide range of domains, such that manifold topic ontologies can be built. We will use $\mathcal{R}e\ell_{part}$, $\mathcal{R}e\ell_{loc}$ and $\mathcal{R}e\ell_{cap}$ to denote the abstract sets of relation types for each general relation, respectively. We want to emphasize that the choice of general relations and sets may be adapted or extended based on the underlying knowledge graph and expected ontology results.

During the construction process, each general relation is processed in succession. Once confirmed one atomic concept $\tilde{A} \in \mathcal{CC}_{sel}$ is proposed for expansion at a time, following the order induced by $\mathcal{Q}_{\mathcal{CT}}$, and respective suggestions are provided. We now describe the individual steps and process in more detail.

Step C1: part-whole

Part-whole relations are frequently incorporated in ontologies and a lot of unintended issues can arise with careless usage. Alan Rector presents a working draft, related to the W3C design patterns [3], in which he addresses some of those issues, their origin in linguistics and philosophy, and their ties to the research area of merology.
The main problem is that part-whole relations express a variety of information: from physical parts (*"A hand is part of an arm."*), over geographical regions (*Vienna is a part of Austria.*), to functional parts (*A CPU is part of a computer.*). Winston et al. present a classification of six types of meronymic relations [WCH87]. A possible solution is defining a role hierarchy in an relational *RBox* that distinguished between the differing meanings. Such a hierarchy might look like this [4]:

$$\begin{aligned}\mathcal{R} = \{&\mathsf{isStructuralPartOf} \sqsubseteq \mathsf{isPartOf}, \mathsf{isFunctionalPartOf} \sqsubseteq \mathsf{isPartOf},\\ &\mathsf{isSubdivisionOf} \sqsubseteq \mathsf{isStructuralPartOf}, \mathsf{isComponentOf} \sqsubseteq \mathsf{isStructuralPartOf}\}\end{aligned}$$

An often overlooked issue is based on the assumption that part-whole relations generally are transitive and invertible (HasPart vs.PartOf). There exist cases though, where transitivity might produce unwanted reasoning results. For example *membership* is often expressed as a part-whole relation, as the example *"Books are part of a library. The Main Library is part of the city infrastructure."* shows. With transitivity {Library $\sqsubseteq \exists$HasPart.Book,

[3] https://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole/
[4] adapted from slides by Alan Rector [Rec]

$\mathsf{CityInfrastructure} \sqsubseteq \exists\, \mathsf{HasPart}.\mathsf{MainLibrary}, \mathsf{Library} \sqsubseteq \mathsf{MainLibrary}\}$ would imply, that $\mathsf{CityInfrastructure} \sqsubseteq \mathsf{HasPart}.\mathsf{Book}$, which is not very intuitive. In the end the correct choice resides with the ontology designer, who needs be aware of the common pitfalls.

To offer flexibility we select the role names HasPart and PartOf by default, but offer the option to define a role name of choice that better captures the required type of part-whole relation.
In detail the algorithm proceeds as follows:

The user is asked whether s/he wants to create HasPart and PartOf relations. If the answer is positive, then for each $\tilde{A} \in N_C(\mathcal{CT})$ the two sets $\mathcal{S}(\phi(\tilde{A}), \mathcal{R}e\ell_{part})$ and $\mathcal{S}(\phi(\tilde{A}), \mathcal{R}e\ell_{part^-})$ of suggestions are retrieved, where $\mathcal{R}e\ell_{part^-}$ uses the reciprocal request types to retreive the backwards assertions.

For each $\tilde{A} \in N_C(\mathcal{CT})$, we define the following combined set of suggestions

$$\mathcal{S}_{\tilde{A}} := \bigcup_{\substack{\phi(A)=n' \\ A \sqsubseteq^*_{\mathcal{CT}(\mathcal{X})} A}} \mathcal{S}(n', \mathcal{R}e\ell_{part}). \tag{4.1}$$

Section 4.3.1 explains the intuition behind the choice of $\mathcal{S}_{\tilde{A}}$.

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

1. It asks if the user wants suggestions of parts of $\tilde{A}$. If the answer is no, then it moves to the next concept in the queue. If further subclasses of $\tilde{A}$ exist that are not part of $\mathcal{CC}_{sel}$ the user can decide to view those before moving to the next item in $\mathcal{Q}_{\mathcal{CT}}$. If all subclasses have been proposed the progression returns to the order given by the queue. It is important to note that concepts are only prompted once and will be skipped if they were already dealt with.

2. If the answer to view suggestions is yes, then the user is prompted with the suggestions in $\mathcal{S}_{\tilde{A}}$. He or she can select from the list, or type fresh concept names, and the axiom
$$\tilde{A} \sqsubseteq \exists \mathsf{HasPart}.(B_1 \sqcup \cdots \sqcup B_n)$$
is added, where $(B_1, \ldots, B_n)$ is the list of the selected and typed concept names. This step can be repeated to create as many analogous axioms
$$\alpha_i = A \sqsubseteq \exists \mathsf{HasPart}.(B_1^i \sqcup \cdots \sqcup B_{n_i}^i)$$
as desired, with different disjunctions $(B_1^i \sqcup \cdots \sqcup B_{n_i}^i)$.

3. The user is given the option of automatically adding an axiom $\tilde{A} \sqsubseteq \forall \mathsf{HasPart}.\tilde{A}\mathsf{Part}$ with $\tilde{A}\mathsf{Part}$ a fresh concept name. Additionally axioms of the form $B_k \sqsubseteq \tilde{A}\mathsf{Part}$ for each B_k, $1 \leq k \leq n$, that is part of the disjunction are added to $\mathcal{T}$.

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

After the whole queue has been processed, the user is asked if s/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 HasPart, and the process is repeated, using the role PartOf and the following combined set of suggestions for each $\tilde{A} \in N_C(\mathcal{CT})$:

$$\mathcal{S}'_{\tilde{A}} := \bigcup_{\substack{\phi(A)=n' \\ A \sqsubseteq^*_{\mathcal{CT}(\mathcal{X})} A}} \mathcal{S}(n', \mathcal{R}e\ell_{part^-}). \tag{4.2}$$

Step C2: locations

Suggestions for possible or typical locations are generated with $\mathcal{S}(\phi(A), \mathcal{R}e\ell_{loc})$ and $\mathcal{S}(\phi(A), \mathcal{R}e\ell^-_{loc})$. They might include distinct, physical locations (*kitchen, church*), as well as more abstract locations (*nature, dream*), which can be hard to represent. The easiest to represent are geographical locations, such as countries or cities, as respective suggestions can be neatly integrated into the taxonomy or transformed into individuals.

The user can first select to get suggestions s of the form '$\tilde{A}$ at location s' and '$\tilde{A}$ is possible location of s'. Iteration through $\mathcal{Q}_{\mathcal{CT}}$ and the process of adding axioms in each case is the same as detailed in step *C4* (topic-specific roles), where role names are specified by the user.

Step C3: capabilities

Suggestions for object capabilities are generated with $\mathcal{S}(\phi(A), \mathcal{R}e\ell_{cap})$ and $\mathcal{S}(\phi(A), \mathcal{R}e\ell^-_{cap})$. Capability is used as a broad term to pool properties of concepts that are linked to descriptive characteristics and action-related features of concepts. This includes suggestions for actions that can be performed by (active) or upon (passive) the current concept $\tilde{A}$. Possible relations that fit into $\mathcal{R}e\ell_{cap}$ are *HasShape, HasColor, evokesEmotion, Causes, CapableOf, hasRevenue*.

Same as in *C1* and *C2* the user can select to get suggestions s of the form '$\tilde{A}$ capable of s' and '$\tilde{A}$ is capability of s'. The process is then analogue to *C2*.

4.3.1 Adjusting Information over Hierarchy Levels

If knowledge graph content is automatically gathered from different resources or through text mining of text written by humans, assertions can be semantically correct, but differ in their level of precision. Both triples *(Spatula, TypeOf, Tool)* and *(Spatula, TypeOf, Hand-held Kitchen Utensil)* are correct, but the second is more precise than the first. Unless the knowledge graph is carefully revised, it is prone to contain multiple assertions that convey the same fundamental information, but are expressed through different entities. Oftentimes multiple occurences are deliberate as they increase information

content and do not necessarily induce inconsistencies. Especially ConceptNet, that was used for our implementation of *CN2TopicOnto*, contains multi-word, human-like phrases and as such produces assertions which differ a lot in expressivity.
As a result suggestions extracted for a concept might not be restricted to this concept, but applicable to a more general concept as well. Conversely multiple sub-variants can exist for the same concept, that differ in verbalization, specificity or spelling.
For example a Bicycle might have as a part a Wheel as well as a BycicleWheel. Compound words can be more intuitive for some concepts and not for others. While Unicycle and Tricycle are similar concepts to Bycicle, a UnicycleWheel or a TricycleWheel are a lot less common and likely to be included in a knowledge graph.
Even if a type of Vehicle has as part a Wheel it might not be suggested a WheeledVehicles due to missing information in the knowledge graph.In a DL ontology it is simple to retrospectively add an axiom $\mathsf{WheeledVehicle} \equiv \mathsf{Vehicle} \sqcap \exists \mathsf{HasPart}.\mathsf{Wheel}$.
One challenge for the representation of common-sense knowledge is the distinction between universally true facts and intuitive or likely true statements. For a knowledge graph it is reasonable to state that a Vehicle has as part a Wheel, as this is true in the majority of cases, but in a strict environment this statement leads to semantical incorrectness if a Boat or Ship are considered subclasses of Vehicle. Allowing inconsistencies and forgoing completeness are design choices that common-sense knowledge graphs need to make to keep information simple and preserve processability for large amounts of data.

For our algorithm we considered two different approaches to rectify assertions that lead to consistency in reasoning or promote ambiguity of concept names. The first approach has the user verify for each axiom before creation, that it does not apply to an existing, more general superconcept of $\tilde{A}$ in the taxonomy and offers the option to change and adapt suggestions for individual cases. This produces a lot of redundant inquiries. Therefore we chose an alternative approach and propose the pooling of suggestions for central concepts in $\mathcal{CC}$, to always propose general as well as specific options for each concept. This way the user can select from a wide variety of appropriate ones.

In a preprocessing step suggestions for all descendants of a concept are unified. For the atomic concept $\tilde{(A)}$ the resulting list of suggestions

$$\tilde{\mathcal{S}}(\phi(\tilde{A}), \mathcal{R}e\ell_s) := \bigcup_{\substack{\phi(A)=n' \\ A \sqsubseteq^*_{\mathcal{CT}(\mathcal{X})} A}} \mathcal{S}(n', \mathcal{R}e\ell_s) \tag{4.3}$$

is presented for $s \in \{part, loc, cap\}$. With this approach higher-level concepts have very exhaustive lists of suggestions, while suggestions become better and more precise for specific concept. To account for inheritance, suggestions that were already selected in an axiom for a concept $\tilde{A}$ are not suggested for respective subconcepts anymore.
Instead, under three different conditions, some basic concepts are kept in a separate list of suggestions that might be suitable for *specification*:

1. If an GCI is added, whose LHS is $\tilde{A}$ and whose RHS is a disunction $B_1 \sqcup \cdots \sqcup B_n$ of basic concepts then the individual concepts $B_1, \ldots, B_k$ are added to the list.

2. If an axiom contains $\exists R_j.(B_{j_1} \sqcup \ldots \sqcup B_{j_n})$ and further axioms $B_{\ell_1} \sqsubseteq B_{j_k}, \ldots, B_{\ell_m} \sqsubseteq B_{j_k}$, $1 \leq k \leq n$ are created through dynamic extensions then $B_{\ell_1}, \ldots, B_{\ell_m}$ are added to the list.

3. The basic concept B_1 on the LHS of an axiom $B_1 \sqsubseteq \tilde{A}$ is added to the list.

Suggestions suitable for specification might fit into more specific axioms for a subclass of $\tilde{A}$.

For example, assume $\tilde{A}$ =Produce. Upon creating Produce $\sqsubseteq \exists$SoldAt.(Market$\sqcup$Store$\sqcup$Shop) it is possible to specify concept names GroceryStore, FurnitureStore and FarmersMarket through dynamic extensions with subclasses. This extends $\mathcal{T}$ with {GroceryStore $\sqsubseteq$ Store, FurnitureStore $\sqsubseteq$ Store, FarmersMarket $\sqsubseteq$ Market}. Now Market, Store, Shop as well as GroceryStore and FurnitureStore are removed from the general suggestions for each subconcept of Produce. They now appear as suggested for *specification* and can be selected to build an axiom Fruit $\sqsubseteq \exists$SoldAt.(GroceryStore $\sqcup$ FarmersMarket) if $\mathcal{CT}$ contains Fruit $\sqsubseteq$ Produce, because Fruit as a subclass, is then queried after Produce.

For $\tilde{A}$ =Animal, if an inverse axiom Savannah $\sqsubseteq \exists$ NaturalHabitatOf.Animal is created, Savannah is listed again for specification for all further subclasses of Animal. This way a more specific axiom Savannah $\sqsubseteq \exists$ NaturalHabitatOf.Lion can also be added. Lion is here assumed to be a subclass of Animal.

Step C4: user-specified roles

During this step suggestions are kept very general and the user has the option to create axioms freely. All constructions listed under Section 4.1.1 are supported, with the exception of (AxAllPart).
Again the queue $Q_{\mathcal{CT}}$ is iterated and for each $\tilde{A}$ the user is optionally prompted with suggestions $\mathcal{S}(\phi(A), \mathcal{R}e\ell_{sp})$, where $\mathcal{R}e\ell_{sp}$ is a set of very general relation types such as *RelatedTo, AssociatedWith, SimilarTo*. Suggestions that were saved during step 1 (*saved suggestions*) are also proposed.

Then role and concept names can be inserted and used to create GCIs as follows:

1. The user is asked if s/he wants to add a new axiom containing $\tilde{A}$. If not, the algorithm moves to the next item. Again central concepts that are not proposed by default as part of $\mathcal{CC}_{sel}$ can be viewed and explicitly selected.

2. If yes, the user is prompted to enter a set of relation names $(R_1, \ldots, R_m)$, which may include IsA or TMPRelated. Each such selection creates one axiom that contains exactly the role names contained in the set. Once entered, role names are saved and can be chosen through the auto-complete function for future prompts.

3. Following, m three-part prompts are issued, one for each R_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 $\mathcal{S}(\phi(\tilde{A}), \mathcal{R}e\ell_{sp})$, the saved suggestions, and all individuals in $N_I(\mathcal{T})$ (see *dynamic extensions* in Section 4.3.2.2 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 $(B^{j_1}, \ldots, B_{j_n})$ be the entered basic concepts for R_j. An input $\tilde{A}.R_j.B_{j_1} \ldots B_{j_n}$ is converted into $\exists R_j.(B_{j_1} \sqcup \cdots \sqcup B_{j_n})$. If $R_j = IsA$, then the input is converted into $\bigsqcup_k B_k$, without an existential.

After all m prompts are completed, they are combined into the final axiom

$$\tilde{A} \sqsubseteq \bigsqcup_k B_k \sqcup \bigsqcup_j \exists R_j.(B_{j_1} \sqcup \cdots \sqcup B_{j_n})$$

which is added to $\mathcal{T}$. Steps 1-3 can be repeated as often as wanted for each $\tilde{A}$ to create multiple axioms

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

4.3.2 Disjunctive Axioms, Dynamic Extensions and Reverted Axioms

In this part we explain in more detail how to create the different types of axioms with the help of examples. The examples assume a command-line setting, as this is what was used for the implementation of *CN2TopicOnto*. Text written in `typewriter` refers to terminal input and output. Each example shows terminal input and the respective axiom that is added to $\mathcal{T}$ below.
We illustrate how to introduce new concepts and roles, create individuals, and add range restrictions and disjointness axioms. We also explain the purpose of the two special role names, IsA and TMPRelated, which are by default included in $N_R(\mathcal{X})$.

First we want to summarize the use of auto-complete, as it is used to quickly select basic concepts and role names.

Auto-complete To simplify the creation of axioms, every time an input is expected, the console auto-complete function (»TAB«) may be used to select R_j from $N_R(\mathcal{X})$ or B_{ij} from a list of proposed basic concepts. We denote the lists of options, that is offered for $\tilde{A}$, $\mathcal{A}uto(\tilde{A})$. In an abstract sense $\mathcal{A}uto(\tilde{A})$ is a list of propositions similar to the suggestions, but further adjusted to the expected input.

There are three different cases in which $\mathcal{A}uto(\tilde{A})$ changes based on the expected input:

Case 1: The input is expected to be an existing or new role name R_j.
Then $\mathcal{A}uto(\tilde{A}) = N_R(\mathcal{X})$.

Case 2: The input is expected to be an existing individual or a new or existing concept name B_{j_k}.

Then $\mathcal{A}uto(\tilde{A}) = N_I(\mathcal{X}) \cup \mathcal{S}(\phi(\tilde{A}), \mathcal{R}e\ell) \cup \mathcal{R}em$, where

$$\mathcal{R}em = \{A \mid A \sqsubseteq^*_{\mathrm{Tax}(\mathcal{X})} A' \textit{ for some } A' \textit{ s.t. there is } \tilde{A} \sqsubseteq \exists \mathsf{TMPRelated}.A' \in \mathcal{T}\} \tag{4.4}$$

is the set of *remembered* concept names.
As a consequence any individual that was added to $N_I(\mathcal{X})$ can be selected from $\mathcal{A}uto(\tilde{A})$ in the prompt for some B_{j_k}.

Case 3: A range restriction $Ran(R_j, B_{j_k})$ was enforced for some B_{j_k}.
This case is explained in the section on range restrictions.

4.3.2.1 Basic Disjunctive Axioms

Axioms of type (AxDisjun), $\tilde{A} \sqsubseteq \bigsqcup_k B_k \sqcup \bigsqcup_j \exists R_j.(B_{j_1}, \ldots, B_{j_n})$, are the most common axioms we want to construct as they establish necessary conditions for $\tilde{A}$. The definition conjoins multiple subforms.

We will focus on axioms of the following form that only contain existential components for now:

$$\tilde{A} \sqsubseteq \bigsqcup_j \exists R_j.(B_{j_1}, \ldots, B_{j_n}) \tag{AxDisjun-a}$$

where $B_{j_1}, \ldots, B_{j_n}$ are basic concepts.

The following examples demonstrate how different axioms that fall under the scope of (AxDisjun-a) are created. **Simple Axiom:** $\tilde{A} \sqsubseteq \exists R_1.B_1$

This is the simplest axiom. It contains a single role name and a single concept name.

Example 1 (Simple Axiom):

```
1 relation name = MadeFrom
2 Pizza.MadeFrom.PizzaDough
```

Pizza $\sqsubseteq \exists$ MadeFrom.PizzaDough

Example 2 (Simple Axiom):

```
1 relation name = Contains
2 Dictionary.Contains.WordDefinition
```

Dictionary $\sqsubseteq \exists$ Contains.WordDefinition

Note that a conjunction over multiple existentials can be achieved with multiple $\tilde{A} \sqsubseteq \exists R_1.B_1, \ldots, \tilde{A} \sqsubseteq \exists R_1.B_n$, where each is a single simple axiom.

Example 3 (Simple Axioms):

```
1 relation name = HasGenre
2 HistoricalRomance.HasGenre.History
3
4 relation name = HasGenre
5 HistoricalRomance.HasGenre.Romance
```

HistoricalRomance $\sqsubseteq \exists$ HasGenre.History
HistoricalRomance $\sqsubseteq \exists$ HasGenre.Romance
or equivalently
HistoricalRomance $\sqsubseteq \exists$ HasGenre.History $\sqcap \exists$ HasGenre.Romance)

Disjunctive Axiom 1: $\tilde{A} \sqsubseteq \exists R_1.(B_1 \sqcup \ldots \sqcup B_n)$

To add a disjunction over multiple basic concepts, the inputs $B_1, \ldots, B_n$ have to be separated by blanks.

Example 4 (Disjunctive Axiom):

```
1 relation name = Contains
2 WrittenText.Contains.Character Word Sentence
```

WrittenText $\sqsubseteq \exists$ Contains.(Character $\sqcup$ Word $\sqcup$ Sentence)

Disjunctive Axiom 2: $\tilde{A} \sqsubseteq \bigsqcup_j \exists R_j.(B_{j_1} \sqcup \ldots \sqcup B_{j_n})$

To add a conjunction with multiple existential restrictions, all occuring $(R_1, \ldots, R_m)$ have to be entered in succession, separated by blanks. Then m single prompts need to be filled, one for each j where $1 \leq j \leq m$.

Example 5 (Disjunctive Axiom):

```
1 relation name = HasIngredient Contains
2 Cake.HasIngredient.Sugar
3 Cake.Contains.SugarSyrup ArtificialSweetener
```

Cake $\sqsubseteq \exists$ HasIngredient.Sugar $\sqcup \exists$ Contains.(SugarSyrup $\sqcup$ ArtificialSweetener)

The two role names HasIngredient and Contains induce the two prompts on line 2 and line 3.

4.3.2.2 Dynamic Extensions

When an axiom for $\tilde{A}$ is added that contains some atomic concept $A \in N_C$, it is possible to dynamically extend Tax with a set of axioms $B_1 \sqsubseteq A, \ldots, B_\ell \sqsubseteq A$ (AxSub).

ACIs added to Tax through dynamic extensions do not fulfill subrequirement of item (c) in Definition 3.2.8 of the central taxonomy, and are not part of $\mathcal{CT}$.

Dynamic extensions are the only way to add new individuals to $N_I(\mathcal{X})$. If input is selected from the suggestions, the concept name is automatically transformed into a corresponding individual name that uses lower case letters.

Dynamic Extension: Subclass $\tilde{A} \sqsubseteq \exists R_1.A_1$ and $A_i \sqsubseteq A_1,\ 2 \leq i \leq \ell$

To create subclasses A_i of A_1 and add the respective ACIs to Tax, the suffix `:` is utilized. It can be useful if suggestions are not suitable to be used in the main axiom, but should be covered by the RHS. For example the axiom can introduce a new atomic concept A_i and some suggestions are then selected as subconcepts of A_i.
A colon (`:`) after a concept name signalizes that further subclasses ought to be created for this concept. Subsequently a y/n prompt verifies the axiom that is in the end added to $\mathcal{T}$.

Example 6 (Dynamic Extension: Subclass):

```
1 relation name = BoughtAt
2 Produce.BoughtAt.Store:
3 Enter subclasses for Store:  GroceryStore Supermarket
4 Create an existential axiom for Produce?  [y/n] y
5 Produce.BoughtAt.Store
```

Produce $\sqsubseteq \exists$ BoughtAt.Store
GroceryStore $\sqsubseteq$ Store, Supermarket $\sqsubseteq$ Store

The prompt on line 4 verifies the axiom that is actually created. In this case it contains the general concept Store.

Example 7 (Dynamic Extension: Subclass):

```
1 relation name = BoughtAt
2 Produce.BoughtAt.Store:
3 Enter subclasses for Store:GroceryStore HardwareStore
4 Create an existential axiom for Produce?  [y/n] y
5 Produce.BoughtAt.GroceryStore
```

Produce $\sqsubseteq \exists$ BoughtAt.GroceryStore
GroceryStore $\sqsubseteq$ Store, HardwareStore $\sqsubseteq$ Store

In this case the prompt on line 4 creates an axiom for the more specific concept GroceryStore. No axiom Produce $\sqsubseteq \exists$ BoughtAt.Store is created.

Example 8 (Dynamic Extension: Subclass):

```
1 relation name = BoughtAt
2 Produce.BoughtAt.Store:
```

```
3 Enter subclasses for Store:GroceryStore Supermarket
4 HardwareStore
5 Create an existential axiom for Produce?  [y/n] n
```

GroceryStore $\sqsubseteq$ Store, Supermarket $\sqsubseteq$ Store, HardwareStore $\sqsubseteq$ Store

In this case no axiom for Produce is created at all.

Dynamic Extension: Individual $\tilde{A} \sqsubseteq \exists R_1.A_1$ and $\{a_i\} \sqsubseteq A_1,\ 1 \leq i \leq \ell$

To create individuals a_i the suffix `()` is utilized.
Adding double brackets `()` after a concept name A_1 signals that axioms $\{a_i\} \sqsubseteq A_1$ ought to be created for some a_i. Subsequently a y/n prompt decides if a complex axiom for $\tilde{A}$ with a nominal construction should be created. If yes, then a subset of individuals $\mathcal{B} \subseteq \{a \in N_I(\mathcal{X}) |\ \{a\} \sqsubseteq A_1 \in \mathcal{T}\}$ can be specified and the resulting axiom is added to $\mathcal{T}$.

Example 9 (Dynamic Extension: Individual):

```
1 relation name = HasColor
2 Apple.HasColor.Color()
3 Enter instances for Color:red orange green blue
4 Create an existential axiom for Apple?  [y/n] y
5 Apple.HasColor.red green
```

{red} $\sqsubseteq$ Color, {orange} $\sqsubseteq$ Color, {green} $\sqsubseteq$ Color, {blue} $\sqsubseteq$ Color,
Apple $\sqsubseteq \exists$ HasColor.{red, green}

The prompt on line 4 adds the complex axiom with nominal construction. No axiom Apple $\sqsubseteq \exists$ HasColor.Color is created.

Example 10 (Dynamic Extension: Individual):

```
1 relation name = HasColor
2 Apple.HasColor.Color()
3 Enter instances for Color:red orange green blue
4 Create an existential axiom for Apple?  [y/n] n
```

{red} $\sqsubseteq$ Color, {orange} $\sqsubseteq$ Color, {green} $\sqsubseteq$ Color, {blue} $\sqsubseteq$ Color,

In this case no axiom for Apple is created at all.

4.3.2.3 Disjunctive Axioms with Dynamic Extensions

The above constructions can be arbitrarily combined to add axioms through dynamic extenstions for disjunctive axioms, as is shown in the following examples.

Example 11 (Combination):

```
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
5 LibraryBook.InTemporalStorage.Table:
6 Enter subclasses for Table:PresentationTable RollingTable
7
8 relation name = InStorage
9 RomanceBook.InStorage.bc1 bc2
```

{bc1} ⊑ Bookcase,{bc2} ⊑ Bookcase,{bc3} ⊑ Bookcase,{bc4} ⊑ Bookcase,
LibraryBook ⊑ ∃ InStorage.(Shelf ⊔ Bookcase) ⊔ ∃ InTemporalStorage.(Table)
Bookshelf ⊑ Shelf, PresentationTable ⊑ Table, RollingTable ⊑ Table
RomanceBook ⊑ ∃ InStorage.{bc1, bc2}

Example 12 (Combination):

```
1 relation name = HighIn ContainSugar ContainVitamin
2 Fruit.HighIn.Fibre
3 Fruit.ContainSugar.Sugar:
4 Enter subclasses for Sugar:Monosaccharide Disaccharide
5 Fruit.ContainVitamin.Vitamin:
6 Enter subclasses for Vitamin:VitaminA VitaminB VitaminC
7 VitaminD VitaminE
```

Fruit ⊑ ∃ HighIn.Fibre ⊔∃ ContainSugar.Sugar ⊔∃ ContainVitamin.Vitamin
Monosaccharide ⊑ Sugar, Disaccharide ⊑ Sugar
VitaminA ⊑ Vitamin, VitaminB ⊑ Vitamin, VitaminC ⊑ Vitamin, VitaminD ⊑ Vitamin,
VitaminE ⊑ Vitamin,

Example 13 (Combination):

```
1 relation name = CanBeFoundIn CanBeFoundUnder
2 Crustacean.CanBeFoundIn.BeachArea() Sand Waterbody:
3 Enter instances for BeachArea:laguna_beach malibu_beach
4 Enter subclasses for Waterbody:  Lake Ocean Pond River Sea
5 Crustacean.CanBeFoundUnder.Rock Stone
```

Crustacean ⊑ ∃ CanBeFoundIn.(BeachArea ⊔ Sand ⊔ Waterbody) ⊔ ∃ CanBeFoundUnder.(Rock ⊔ Stone)
{laguna_beach} ⊑ BeachArea, {malibu_beach} ⊑ BeachArea
Lake ⊑ Waterbody, Ocean ⊑ Waterbody, Pond ⊑ Waterbody, River ⊑ Waterbody,
Sea ⊑ Waterbody,

4.3.2.4 Reverted Axioms

So far $\tilde{A}$ was fixed on the LHS of all axioms (AxDisjun). We call the axioms that have $\tilde{A}$ on the RHS *reverted* axioms.

Reverted Axiom: $B_1 \sqsubseteq \exists R_1.\tilde{A}$

By attaching a minus – to the role name it is possible to invert the position of $\tilde{A}$ in the axiom, such that the RHS has the currently considered concept fixed, and the LHS can be specified. At the current time no concept constructors or extensions are supported on the LHS.

Example 14 (Reverted Axiom):

```
1 relation name = HasOffer-
2 x.HasOffer.ReadingMaterial, x = Library
```

Library $\sqsubseteq \exists$ HasOffer.ReadingMaterial

In this example $\tilde{A} =$ ReadingMaterial and Library is part of the suggestions $\mathcal{S}(\phi(\tilde{A}), \mathcal{R}e\ell^{-}_{loc})$.

Example 15 (Reverted Axiom):

```
1 relation name = HasEngine-
2 x.HasEngine.DieselEngine, x = volvoC70D5 volvoC30D5
```

{volvoC70D5} $\sqsubseteq \exists$ HasEngine.DieselEngine
{volvoC30D5} $\sqsubseteq \exists$ HasEngine.DieselEngine
or equivalently
{volvoC70D5, volvoC30D5} $\sqsubseteq \exists$ HasEngine.DieselEngine

Note that a disjunction over multiple concept names $B_1 \sqcup \ldots \sqcup B_n \sqsubseteq \exists R_1.\tilde{A}$, can be achieved with multiple single axioms $B_1 \sqsubseteq \exists R_1.\tilde{A}, \ldots, B_n \sqsubseteq \exists R_1.\tilde{A}$.

4.3.3 Range Restrictions and Concept Disjointness

In this subsection we illustrate how range restriction axioms $\exists R_1^{-}.\top \sqsubseteq \bigsqcap_i A_i$ and concept disjointness axioms $A_1 \sqcap A_2 \sqsubseteq \bot$ are automatically added to $\mathcal{T}$ under certain conditions.

4.3.3.1 Range Restrictions

If a single role name R_1 and a single atomic concept A_1 are entered and individuals or subclasses are created for A_1 through dynamic extensions, a range restriction $Ran(R_1, A_1)$ is automatically set.
For Examples 6 to 10 this means that $Ran($BoughtAt, Store$)$ and $Ran($HasColor, Color$)$.
With this knowledge we can revisit Example 6, Example 7 and Example 9:

Example 16 (Range Restriction):

```
1 relation name = BoughtAt
2 Produce.BoughtAt.Store:
3 Enter subclasses for Store:  GroceryStore Supermarket
4 Create an existential axiom for Produce?  [y/n] y
5 Produce.BoughtAt.Store
```

$\mathsf{Produce} \sqsubseteq \exists\, \mathsf{BoughtAt.Store}$
$\mathsf{GroceryStore} \sqsubseteq \mathsf{Store}$, $\mathsf{Supermarket} \sqsubseteq \mathsf{Store}$
$\exists\, \mathsf{BoughtAt}^{-}.\top \sqsubseteq \mathsf{Store}$

The axiom $\exists\, \mathsf{BoughtAt}^{-}.\top \sqsubseteq \mathsf{Store}$ expresses the range restriction $Ran(\mathsf{BoughtAt}, \mathsf{Store})$.

Example 17 (Range Restriction):

```
1 relation name = BoughtAt
2 Produce.BoughtAt.Store:
3 Enter subclasses for Store:  GroceryStore HardwareStore
4 Create an existential axiom for Produce?  [y/n] y
5 Produce.BoughtAt.GroceryStore
```

$\mathsf{Produce} \sqsubseteq \exists\, \mathsf{BoughtAt.GroceryStore}$
$\mathsf{GroceryStore} \sqsubseteq \mathsf{Store}$, $\mathsf{HardwareStore} \sqsubseteq \mathsf{Store}$
$\exists\, \mathsf{BoughtAt}^{-}.\top \sqsubseteq \mathsf{Store}$

Example 18 (Range Restriction):

```
1 relation name = HasColor
2 Apple.HasColor.Color()
3 Enter instances for Color:red orange green blue
4 Create an existential axiom for Apple?  [y/n] y
5 Apple.HasColor.red green
```

$\mathsf{Color(red)}$, $\mathsf{Color(orange)}$, $\mathsf{Color(green)}$, $\mathsf{Color(blue)}$
$\mathsf{Apple} \sqsubseteq \exists \mathsf{HasColor.}\{\mathsf{red}, \mathsf{green}\}$
$\exists\, \mathsf{HasColor}^{-}.\top \sqsubseteq \mathsf{Color}$

When the prompt is answered negatively, no axiom is created, but the range restriction is still enforced.

Example 19 (Range Restriction):

```
1 relation name = HasColor
2 Apple.HasColor.Color()
3 Enter instances for Color:red orange green blue
4 Create an existential axiom for Apple?  [y/n] n
```

$\mathsf{Color(red)}$, $\mathsf{Color(orange)}$, $\mathsf{Color(green)}$, $\mathsf{Color(blue)}$
$\exists\, \mathsf{HasColor}^{-}.\top \sqsubseteq \mathsf{Color}$

If a role name $R_1 \in N_R(\mathcal{X})$ with $Ran(R_1, A_1)$ for some atomic concept A_1 is entered, and a range restriction on a (new) atomic concept A_2 is put into effect, then $Ran(R_1, A_1 \sqcap A_2)$.

The Effect of Range Restrictions on Auto-complete If a range restriction was imposed (line 1-3 in Examples 16 to 18) and an existential axiom $\tilde{A} \sqsubseteq \exists R_1.(B_1 \sqcup \ldots \sqcup B_n)$ is being created (line 5 in Examples 16 to 18), $\mathcal{A}uto(\tilde{A})$, the proposed input for all B_i, $1 \leq i \leq n$ changes to account for the range restriction on R_1.

Assume a restriction $Ran(R_1, A_1)$ is set. Then there are two different cases.

Case Individuals: If individuals were created (examples 16 and 17), the expected input for the B_i, $1 \leq i \leq n$, is $a \in N_I(\mathcal{X})$ such that $Ran(R_1, A_1)$ holds, therefore $\mathcal{A}uto(\tilde{A}) = \{a \in N_I(\mathcal{X}) \mid \{a\} \sqsubseteq A_1\}$.
At this point is it only possible to add $a \in N_I(\mathcal{X})$. If anything else, that is not part of the effective vocabulary, is inserted, the user is warned that some input was invalid and only existing individuals are added to the nominal construction.

Case Subclasses: If subclasses were created (example 18), the expected input for the B_i, $1 \leq i \leq n$, is $A \in N_C(\mathcal{X})$ such that $Ran(R_1, A_1)$ holds, therefore $\mathcal{A}uto(\tilde{A}) = \{A \in N_C(\mathcal{X}) \mid A \sqsubseteq^*_{\mathrm{Tax}(\mathcal{X})} A_1\}$.
Any input string is permitted. If it is not yet part of the effective vocabulary, it is assumed to be a new concept name and it is added to $N_C(\mathcal{X})$. A warning is displayed, but in contrast to the individuals case, the axiom is created as specified.

We now assume that $\mathcal{T}$ already contains multiple axioms, including a range restriction $Ran(R_1, A_1)$. If $R_1 \in N_R(\mathcal{X})$ is now chosen as the relation name for a new axiom for any $\tilde{A} \in \mathcal{CC}$ all individuals and concepts that are covered by the range of R_1 are proposed as the expected input. This means

$$\mathcal{A}uto(\tilde{A}) = \{a \in N_I(\mathcal{X}) \mid A_1(a)\} \cup \{A \in N_C(\mathcal{X}) \mid A \sqsubseteq^*_{\mathrm{Tax}(\mathcal{X})} A_1\}.$$

Three different inputs are now possible for the user:

Input 1: Something from $\mathcal{A}uto(\tilde{A})$ is selected. This is the expected case.

Input 2: One or multiple individuals $a \in N_I(\mathcal{X})$ or concept names $A \in N_C(\mathcal{X})$ that are not in $\mathcal{A}uto(\tilde{A})$ are entered. Then it will hold that $\mathcal{X} \vDash A_1(a)$ for all i and $\mathcal{X} \vDash A \sqsubseteq^*_{\mathrm{Tax}(\mathcal{X})} A_1$ for all A.

Input 3: A completely new string that is not in $N_I(\mathcal{X})$ or $N_C(\mathcal{X})$ is entered. Then the input is assumed to be a new concept name. It will hold that $\mathcal{X} \vDash A \sqsubseteq^*_{\mathrm{Tax}(\mathcal{X})} A_1$ for every newly entered atomic concept A .

To summarize: When a role name with a range restriction is selected, only individuals and atomic concepts from $\mathcal{X}$, that are compliant to the restriction, get proposed. However

any meaningful input is permitted. By restraining the proposed input, the user is made aware of the restriction and the possible consequences for reasoning, if something different is entered.

Example 20 (Range Restriction on Input):

```
1 relation name:  HasModeOfTransportation
2 Vehicle.HasModeOfTransportation.ModeOfTransportation:
3 Enter subclasses for ModeOfTransp.:  Air Land Water Space
4 Create an existential axiom for Vehicle?  [y/n] n
5
6 relation name:  HasModeOfTransportation
7 Rocket.HasModeOfTransportation.
8 Air    Land    ModeOfTransportation    Space    Water
9 Rocket.HasModeOfTransportation.Space Air
```

$\mathsf{Air} \sqsubseteq \mathsf{ModeOfTransportation}$, $\mathsf{Land} \sqsubseteq \mathsf{ModeOfTransportation}$,
$\mathsf{Water} \sqsubseteq \mathsf{ModeOfTransportation}$, $\mathsf{Space} \sqsubseteq \mathsf{ModeOfTransportation}$,
$\exists\, \mathsf{HasModeOfTransportation}^{-}.\top \sqsubseteq \mathsf{ModeOfTransportation}$

$\mathsf{Rocket} \sqsubseteq \exists\, \mathsf{HasModeOfTransportation}.(\mathsf{Space} \sqcup \mathsf{Air})$

Here lines 1-4 set the range restriction on HasModeOfTransportation. When creating the axiom $\mathsf{Rocket} \sqsubseteq \exists\, \mathsf{HasModeOfTransportation}.A$ the concept names and individuals (here none) that get proposed by auto-complete for A (line 8), fulfill the previously set restriction.

4.3.3.2 Concept Disjointness

Defining proper disjointness axioms between concepts, is a tedious task that rapidly introduces logical contradictions. There are a few approaches that specifically aim to learn and enrich ontologies with disjointness axioms. The most prominent ones are based on *inductive logic programming* or *association rule mining* [FV11]. *Semantic clarification* is a heuristic introduced by Schlobach [Sch05] that automatically adds appropriate disjointness statements, such that the logical functionality is reintroduced to incoherent ontologies. It makes use of the *Strong Disjointness Assumption (SDA)*, which states that "In a well-modeled terminology the direct siblings, i.e. children of a common parent in the subsumption hierarchy should be disjoint.". This is a strong statement that is hard to conform to, especially in ontologies with ambiguous general-knowledge concepts where the taxonomy structure is not apparent from the outset. As a solution we propose a weaker form that selectively allows the addition of disjointness axioms.

To facilitate disjointness of atomic concepts, we automatically assume that atomic concepts $A_2, \ldots, A_\ell$ that are dynamically added as subclasses of some atomic concept A_1 at the same time, are pairwise disjoint. A set of respective disjointness axioms $Disj(A_i, A_j)$ for $i, j = 2, \ldots, \ell,\ i \neq j$ is then added to $\mathcal{T}$.

If there already exist atomic concepts $A_2, \ldots, A_\ell$ with $A_i \sqsubseteq A_1,\ 2 \leq i \leq \ell$ for some atomic concept A_1 and a new single axiom $A_k \sqsubseteq A_1$ is added with some atomic concept A_k, then the choice can be made to additionally create $Disj(A_k, A_i)$ for all $i = 1, \ldots, \ell$. This makes it possible to specify a number of disjoint concept names at the same time, while keeping the freedom to deviate from the SDA and not add disjointness, by splitting individual subconcept declarations over separate axiom construction steps.

Example 21 (Disjointness):

```
relation name:  HasTopping
Pizza.HasTopping.PizzaTopping:
Enter subclasses for PizzaTopping:  Pepperoni Olive Onion
Create an existential axiom for Pizza?  [y/n] y
Pizza.HasTopping.PizzaTopping

relation name:  HasTopping
Pizza.HasTopping.PizzaTopping:
Enter subclasses for PizzaTopping:  MeatBasedTopping
Make Meat-BasedTopping disjoint from all existing
subclasses?  (Olive, Onion, Pepperoni) [y/n] n
Create an existential axiom for Pizza?  [y/n] n

relation name:  HasTopping
Pizza.HasTopping.PizzaTopping:
Enter subclasses for PizzaTopping:  Pineapple
Make Pineapple disjoint from all existing subclasses?
(Olive, Onion, Pepperoni, MeatBasedTopping) [y/n] y
Create an existential axiom for Pizza?  [y/n] n

relation name:  HasTopping
Pizza.HasTopping.PizzaTopping:
Enter subclasses for PizzaTopping:  Corn Ham
Create an existential axiom for Pizza?  [y/n] n
```

$\mathsf{Pizza} \sqsubseteq \exists\ \mathsf{HasTopping.PizzaTopping}$
$\mathsf{Pepperoni} \sqsubseteq \mathsf{PizzaTopping}$, $\mathsf{Olive} \sqsubseteq \mathsf{PizzaTopping}$, $\mathsf{Onion} \sqsubseteq \mathsf{PizzaTopping}$
$\mathsf{Pepperoni} \sqcap \mathsf{Olive} \sqsubseteq \bot$
$\mathsf{Olive} \sqcap \mathsf{Onion} \sqsubseteq \bot$
$\mathsf{Onion} \sqcap \mathsf{Pepperoni} \sqsubseteq \bot$
$\exists\ \mathsf{HasTopping}^{-}.\top \sqsubseteq \mathsf{PizzaTopping}$

$\mathsf{Meat\text{-}BasedTopping} \sqsubseteq \mathsf{PizzaTopping}$

$\mathsf{Pineapple} \sqsubseteq \mathsf{PizzaTopping}$
$\mathsf{Pineapple} \sqcap \mathsf{Olive} \sqsubseteq \bot$

Pineapple ⊓ Onion ⊑ ⊥
Pineapple ⊓ Pepperoni ⊑ ⊥
Pineapple ⊓ MeatBasedTopping ⊑ ⊥

Corn ⊑ PizzaTopping, Ham ⊑ PizzaTopping
Corn ⊓ Ham ⊑ ⊥

The axiom that is added on lines 21-24 highlights that disjointness is only enforced between subconcepts that are created at the same time. This prevents the user from accidentally introducing inconsistencies when multiple subclass axioms for the same concept are created at different iteration steps. If all newly added concepts were assumed to be disjoint from all already existing subclasses, it would hold that $Disj(\mathsf{MeatBasedTopping}, \mathsf{Ham})$ in the example above. The set of disjointness axioms is kept minimal to assure independence from the sequence in which axioms are added, with the downside of losing some semantical information. Missing disjointness axioms can always be added in an ontology editor at a later point.

4.3.4 The IsA Role

The IsA role name is predefined in $N_R(\mathcal{X})$ and its semantics are equivalent to $\sqsubseteq$. It is used to integrate the generation of axioms

$$\tilde{A} \sqsubseteq \bigsqcup_i B_i \tag{AxDisjun-b}$$

into the general axiom creation process for $\tilde{A} \sqsubseteq \bigsqcup_j \exists R_j.(B_{j_1}, \ldots, B_{j_n})$ (AxDisjun-a) by chosing $R_j = \mathsf{IsA}$. The following convertion is applied

$$\tilde{A} \sqsubseteq \exists\, \mathsf{IsA}.(B_{j_1}, \ldots, B_{j_n}) \longmapsto \tilde{A} \sqsubseteq \bigsqcup_{k=j_1}^{j_n} B_k$$

No range restrictions are ever enforced on IsA.

The support of IsA allows for a retroactive extension of $\mathcal{CT}$ as (AxDisjun-a) becomes the simple axiom $\tilde{A} \sqsubseteq B_1$ if $i = 1$ (AxSub).
Any entered atomic concept B_1 is connected to tc in $\mathcal{G}_{\mathrm{Tax}}$ by construction. To guarantee that $B_1 \in N_C(\mathcal{CT})$, condition (c) needs to be fulfilled in Definition 3.2.8 of $\mathcal{CT}$. This requires $\phi(B_1)$ to be defined. If B_1 is selected from $\mathcal{S}(\phi(\tilde{A}), \mathcal{R}e\ell)$ for some set $\mathcal{R}e\ell$, then $\phi(B_1) := \ell(n')$ such that $B_1 = onto(n')$.
If a completely new concept name B_1 is entered, the mapping can not be reconstructed from the suggestions. In this case it is possible to employ a search procedure that tries to find the best matching node in K. The ConceptNet API, for example, offers the option to search for a matching URI for a given string. This was applied in *CN2TopicOnto* to propose a matching node to the user and extend the mapping in if a new concept name is entered. If the mapping is not extended, then $\mathcal{G}_{\mathcal{CT}}$ possibly becomes disconnected.

This results in a reduction of the iteration order queue $\mathcal{Q}_{\mathcal{CT}}$, which in turn reduces the number of concept names offered for expansion. On the flipside $\mathcal{CT}$ can be extended indefinitely with this method, as newly added concepts are appended to $\mathcal{Q}_{\mathcal{CT}}$ and also prompted for expansion.

Example 22 (IsA):

```
1 relation name:  IsA
2 Bulldozer.IsA.TrackedVehicle WheeledVehicle
```

Bulldozer $\sqsubseteq$ (TrackedVehicle $\sqcup$ WheeledVehicle)

Example 23 (IsA):

```
1 relation name:  IsA
2 Candy.IsA.SweetFood
3 There was no respective entity for SweetFood.  We are trying
4 to find the most suitable.
5 Store /c/en/sweet_food as respective CN id?  [y/n] y
```

Candy $\sqsubseteq$ SweetFood
ϕ(SweetFood)=/c/en/sweet_food

The most fitting entity for SweetFood in ConceptNet is searched and proposed to the user.

Example 24 (IsA):

```
1 relation name:  IsA
2 StoneFruit.IsA.Drupe
3
4 relation name:  IsA-
5 x.IsA.StoneFruit, x = Drupe
```

StoneFruit $\equiv$ Drupe

Adding both ACIs $\tilde{A} \sqsubseteq B_1$, and $B_1 \sqsubseteq \tilde{A}$, through a reverted axiom, creates the equivalence $\tilde{A} \equiv B_1$.

4.3.4.1 Complex Disjunctive Axioms with IsA

The following examples illustrate how all so far described creation methods can be combined to construct complex disjunctive axioms $\tilde{A} \sqsubseteq \bigsqcup_k B_k \sqcup \bigsqcup_j \exists R_j.(B_{j_1} \sqcup \ldots \sqcup B_{j_n})$.

Example 25 (Complex Disjunction):

```
1 relation name:  IsA MakeSick
2 Food.IsA.Edible
3 Food.MakeSick.Person
```

Food $\sqsubseteq$ (Edible $\sqcup \exists$ MakeSick.Person)

Example 26 (Complex Disjunction):

```
1 relation name:  HasLivingState
2 Organism.HasLivingState.LivingState()
3 Enter instances for LivingState:alive dead
4 Create an existential axiom for Organism?  [y/n] y
5 Organism.HasLivingState.alive
6
7 relation name:  IsA HasLivingState
8 Animal.IsA.Organism
9 Animal.HasLivingState.dead
```

$\mathsf{LivingState(dead)}$, $\mathsf{LivingState(alive)}$
$\mathsf{Organism} \sqsubseteq \exists\ \mathsf{HasLivingState.\{alive\}}$
$\exists\ \mathsf{HasLivingState}^{-}.\top \sqsubseteq \mathsf{LivingState}$

$\mathsf{Animal} \sqsubseteq \mathsf{Organism} \sqcup \exists\ \mathsf{HasLivingState.\{dead\}}$

Here $\{\mathsf{Animal} \sqsubseteq \mathsf{Organism}\} \in \mathcal{CT}$ and $\mathcal{S}(\phi(\mathsf{Organism}), \mathcal{R}el_{cap})$ contains the suggestion Alive. In Example 26, adding a more specialized axiom for Animal prevents that *"All animals are alive"* is reasoned from *"All organisms are alive"*.

Example 27 (IsA):

```
 1 relation name:  HasComponent
 2 Sushi.HasComponent.Rice
 3
 4 relation name:  HasComponent IsA
 5 Sushi.HasComponent.Fish:
 6 Enter subclasses for Fish:  Tuna Salmon
 7 Create an existential axiom for Sushi?  [y/n] y
 8 Sushi.HasComponent.Fish
 9 Sushi.IsA.NonFishSushi
10
11 relation name:  HasComponent
12 TunaSushi.HasComponent.Tuna
```

$\mathsf{Sushi} \sqsubseteq \exists\ \mathsf{HasComponent.Rice}$

$\mathsf{Sushi} \sqsubseteq (\mathsf{NonFishSushi} \sqcup \exists\ \mathsf{HasComponent.Fish})$
$\mathsf{Tuna} \sqsubseteq \mathsf{Fish}$, $\mathsf{Salmon} \sqsubseteq \mathsf{Fish}$

$\mathsf{TunaShushi} \sqsubseteq \exists\ \mathsf{HasComponent.Tuna}$

In Example 27 a new atomic concept NonFishSushi is created to model that not all Sushi need to have a Fish component.

4.3.5 The TMPRelated Role

In the examples presented until now there exist a few limitation that were not yet addressed.

So far there is no way to add an axiom $\tilde{A} \sqsubseteq \exists R_1.A_1$ and specify subclasses for A_1 without enforcing a range restriction. If we revisit the PizzaTopping example (Example 21), the axiom $\exists\ \mathsf{HasTopping}^{-}.\top \sqsubseteq \mathsf{PizzaTopping}$ might be too strict. What if a new concept HamburgerTopping and an axiom $\mathsf{Hamburger} \sqsubseteq \exists\ \mathsf{HasTopping}.\mathsf{HamburgerTopping}$ is added? Then it is necessary to relax the range restriction axiom to $\exists\ \mathsf{HasTopping}^{-}.\top \sqsubseteq \mathsf{Topping}$ and add new subclass axioms $\mathsf{HamburgerTopping} \sqsubseteq \mathsf{Topping}$, $\mathsf{PizzaTopping} \sqsubseteq \mathsf{Topping}$.

Another observation shows that there are a lot of redundant repetitions in the first lines of every new input that have no impact on the changes to $\mathcal{T}$.

Both of those problems can be addressed with the help of the role name TMPRelated. The role TMPRelated is in $N_R(\mathcal{X})$ by default and can be used as a wildcard to create basic concepts without adding an associated axiom to the final ontology. This can be relevant when suggestions should be part of the vocabulary, but are not suitable for an axiom construction for the concept $\tilde{A}$ that is currently addressed.

No range restrictions are ever enforced on TMPRelated.

Concept names that are created with TMPRelated are added to $\mathcal{A}uto(\tilde{A})$ for any $\tilde{A} \in \mathcal{CC}$, as part of the rembered concepts $\mathcal{R}em$, as defined in equation (4.4).

All axioms where some $R_j =$TMPRelated are temporarily created and later removed from $\mathcal{T}$ during the clean-up step, while basic concepts, as well as respective subclass and nominal axioms, remain part of $\mathcal{X}$.

In the following we describe some scenarios in which the use of TMPRelated is handy or necessary.

Application Scenario 1

The revisited and extended PizzaTopping example (Example 21) as described above to add further subconcepts of the new atomic concept Topping.

Example 28 (Scenario):

```
1 relation name:  HasTopping
2 Pizza.HasTopping.Topping:
3 Enter subclasses for Topping:  PizzaTopping
4 Create an existential axiom for Pizza?  [y/n] y
5 Pizza.HasTopping.PizzaTopping
6
7 relation name:  TMPRelated
8 Pizza.TMPRelated.PizzaTopping:
```

```
 9 Enter subclasses for PizzaTopping:  Pepperoni Olive Onion
10
11 relation name:  HasTopping
12 Hamburger.HasTopping.HamburgerTopping
13 relation name:  TMPRelated
14 Hamburger.TMPRelated.HamburgerTopping:
15 Enter subclasses for HamburgerTopping:  Avocado Bacon Onion
```

Pizza $\sqsubseteq \exists$ HasTopping.PizzaTopping
$\exists$ HasTopping^{-}.$\top \sqsubseteq$ Topping

Pepperoni $\sqsubseteq$ PizzaTopping, Olive $\sqsubseteq$ PizzaTopping, Onion $\sqsubseteq$ PizzaTopping
Pepperoni $\sqcap$ Olive $\sqsubseteq \bot$
Olive $\sqcap$ Onion $\sqsubseteq \bot$
Onion $\sqcap$ Pepperoni $\sqsubseteq \bot$

Hamburger $\sqsubseteq \exists$ HasTopping.HamburgerTopping

Avocado $\sqsubseteq$ HamburgerTopping, Bacon $\sqsubseteq$ HamburgerTopping, Onion $\sqsubseteq$ HamburgerTopping
Avocado $\sqcap$ Bacon $\sqsubseteq \bot$
Bacon $\sqcap$ Onion $\sqsubseteq \bot$
Onion $\sqcap$ Avocado $\sqsubseteq \bot$

Application Scenario 2

Soup is proposed as a suggestion for $\tilde{A}$ =Chicken through $\mathcal{S}(\phi(\text{Chicken}),\mathcal{R}e\ell_{loc})$. Now TMPRelated can be used to immediately create the atomic concept ChickenSoup as a subclass of Soup and add an existential axiom for ChickenSoup. Furthermore the suggestion of Soup might inspire to integrate other types of soup into the ontology as well.

Example 29 (Scenario):

```
1 relation name = TMPRelated
2 Chicken.TMPRelated.Soup:
3 Enter subclasses for Soup:  ChickenSoup EggDropSoup
4 NoodleSoup
5
6 relation name = HasIngredient-
7 x.HasIngredient.Chicken, x = ChickenSoup
```

ChickenSoup $\sqsubseteq$ Soup, EggDropSoup $\sqsubseteq$ Soup, NoodleSoup $\sqsubseteq$ Soup

ChickenSoup $\sqsubseteq \exists$ HasIngredient.Chicken

With TMPRelated axioms do not need to be verified in a separate step and any concept name added can be immediately placed within the hierarchy of Tax.

Alternatively it is possible to solely create the existential axiom and add ChickenSoup $\sqsubseteq$ Soup later during the clean-up step. Both options achieve the same result, however using TMPRelated as done in Example 29 allows to preemptively create concepts at the correct place in the hierarchy before they are integrated into other complex axioms.

Examples 30 and 31 illustarte the alternative version of Example 29, where proper placement in the taxonomy is done during the clean-up step.

Assume $\mathcal{T} = \{\mathsf{Chicken} \sqsubseteq \mathsf{Food}\}$.

Example 30 (Create Axiom):

```
1 relation name = TMPRelated
2 Chicken.TMPRelated.Soup
3
4 relation name = HasIngredient-
5 x.HasIngredient.Chicken, x = ChickenSoup
```

ChickenSoup $\sqsubseteq \top$, Soup $\sqsubseteq \top$
ChickenSoup $\sqsubseteq \exists$ HasIngredient.Chicken

Example 31 (Clean-Up):

```
1 ['Food','ChickenSoup', 'Soup']
2 Those are classes at the top level at the moment (only
3 subclass of Thing).
4 Enter superclass(es):  Soup
5 Enter subclass(es) for Soup:  ChickenSoup EggDropSoup
6 NoodleSoup
```

ChickenSoup $\sqsubseteq \exists$ HasIngredient.Chicken
ChickenSoup $\sqsubseteq$ Soup, EggDropSoup $\sqsubseteq$ Soup, NoodleSoup $\sqsubseteq$ Soup

Adding ACIs during the clean-up step is intended as a downstream measure to improve $\mathcal{CT}$ for all atomic concepts that were added to $N_C(\mathcal{X})$ by creating complex axioms.

Application Scenario 3

Asumme the following $\mathcal{T}$ was constructed.

$\mathcal{T}$ ={Body $\sqsubseteq$ $\exists$HasHeight.BodyHeight, {tiny} $\sqsubseteq$ BodyHeight, {average} $\sqsubseteq$ BodyHeight, {tall} $\sqsubseteq$ BodyHeight}

At some later point, Short, which appears to be a suitable suggestion for a body height description, gets suggested for some $\tilde{A} \in \mathcal{CC}$. TMPRelated can now be used to add a further nominal axiom for BodyHeight.

Example 32 (Scenario):

```
1 relation name = TMPRelated
2 A.TMPRelated.BodyHeight()
3 Enter instances for BodyHeight:  short
```

{short} ⊑ BodyHeight

Application Scenario 4

TMPRelated can be utilized as a suggestion rendition function. Assume Juice was suggested as a superconcept of Fruit during Step 1 when building the taxonomy (see Section 4.2). Fruit ⊑ Juice is semantically incorrect, but Juice is a thematically fitting concept name, therefore we store it in the *saved suggestions*. It is now suggested again in *C4* (user-specified roles) and can be selected for an axiom. Inspired by the suggestion of Juice, we decide to add some specific types of juice to the ontology. Using TMPRelated we are reminded that those concepts were added to $N_C(\mathcal{X})$, as they are now contained in $\mathcal{R}em$ as remembered concepts.

Example 33 (Scenario):

```
1 relation name = MadeFrom-
2 x.MadeFrom.Fruit, x = Juice
3
4 relation name = TMPRelated
5 Fruit.TMPRelated.Juice:
6 Enter subclasses for Juice:  AppleJuice GrapeJuice
7 OrangeJuice
```

Juice ⊑ ∃ MadeFrom.Fruit
AppleJuice ⊑ Juice, GrapeJuice ⊑ Juice, OrangeJuice ⊑ Juice,

Juice, as well as AppleJuice, GrapeJuice and OrangeJuice will be subsequently contained in $\mathcal{A}uto(\tilde{A})$ for any $\tilde{A} \in \mathcal{CC}$. They can be used to create further axioms straightforwardly, e.g. AppleJuice ⊑ ∃ MadeFrom.Apple.

4.4 Step 3: Clean-Up

In this step the ontology taxonomy Tax can be restructured to reduce the amount of top-level concepts. Concept names that are selected from the suggestions or newly introduced are always added at the top level unless specified otherwise, for example through dynamic extensions or the use of IsA or TMPRelated. During this last step of the construction process further ACIs, containing either existing or new atomic concepts, can be added, to improve Tax. Simultaneously, redundant axioms are removed. An algorithm starts at the top level and lets the user navigate through Tax, moving from one atomic

concept to the next.
This is done as follows:

1. Initially the concept $A = \top$ is selected and $\mathcal{P}_\top = \{A \mid$ there is no B s.t. $A \sqsubseteq B\}$.

2. Then the user is asked if s/he wants to create a new ACI. If the answer is negative the algorithm jumps to step 4.

3. If the answer is yes, then the user is prompted to input two sets $\{A_1, \cdots, A_m\}$ and $\{B_1, \cdots, B_n\}$ of atomic concepts, by either selecting from $\mathcal{P}_A$ or by 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 $A_i \sqsubseteq A$ is removed,
   - if $B_j \in N_C(\mathcal{T})$ and $B_j \sqsubseteq A \notin \mathcal{T}$ then the user can choose to add the axiom $B_j \sqsubseteq A$.

   While the first removes redundant axioms that are implied by $\mathcal{T}$, the second gives a choice to the user if ACIs that contain some B_j already exist in Tax. Then the algorithm jumps to step 2 again and further ACIs can be added.

4. The user is asked if s/he wants to select any $A' \in \mathcal{P}_A$ and get a new set $\mathcal{P}_{A'} = \{A \mid A \sqsubseteq A' \in \mathcal{T}\}$. If yes, the algorithm moves to step 2 with the new set $\mathcal{P}_{A'}$. If no, $\mathcal{P}_A$ is displayed and the algorithm jumps back to step 2 with the current set $\mathcal{P}_A$. In case of a negative answer and if $A = \top$ the algorithm terminates and $\mathcal{X} = (\mathcal{T}, tc, \phi)$ is complete.

Example 34 (Clean-Up):

```
1 These are all concepts at the top level at the moment (only
2 subclass of Thing).
3 [ Action, Color, Container, Fuel, ModeOfTransportation,
4 Object, Part, Petrol, PublicTransport, ...  ]
5
6 Do you want to nest concepts on the current level?[y/n] y
7 Define a new (atomic) concept inclusion?[y/n] y
8 Enter superclass(es):  Fuel
9 Enter subclass(es) for Fuel:  Petrol
```

Petrol $\sqsubseteq$ Fuel

Example 35 (Clean-Up):

```
1 These are all concepts at the top level at the moment (only
2 subclass of Thing).
```

```
 3 [ Activity, CalorificValue, Condiment, Container,
 4 HamburgerJoint, Meal, Restaurant, ... ]
 5 Do you want to nest concepts on the current level?[y/n] n
 6 Go to a deeper level for further nesting?[y/n] y
 7
 8 Those are subclasses of Thing that have further subclasses:
 9 [ Activity, CalorificValue, Container, Restaurant, ... ]
10 Enter the concept you want to jump into:  Container
11
12 » Moving into Container
13
14 These are all concepts that are subclasses of Container.
15 [ Box, Can, FoodContainer, Jar, PizzaBox, StyrofoamContainer ]
16
17 Do you want to nest concepts on the current level?[y/n] y
18 Define a new (atomic) concept inclusion?[y/n] y
19 Enter superclass(es):  Box FoodContainer
20 Enter subclass(es) for Box FoodContainer:  PizzaBox
21
22 Define a new (atomic) concept inclusion?[y/n] n
23 Go to a deeper level for further nesting?[y/n] n
24
25 « Moving out of Container and back into Thing
```

Container $\sqsubseteq \top$, Box $\sqsubseteq$ Container,
PizzaBox $\sqsubseteq$ Box, PizzaBox $\sqsubseteq$ FoodContainer

Here the messages on lines 12 and 25 indicate the move to a new and back to the previous $\mathcal{P}_A$ after no further axioms are added and no new concept is chosen.

Example 36 (Clean-Up):

```
 1 These are all concepts at the top level at the moment (only
 2 subclass of Thing).
 3 [ Blue, Property ]
 4
 5 Do you want to nest concepts on the current level?[y/n] y
 6 Define a new (atomic) concept inclusion?[y/n] y
 7 Enter superclass(es):  Color
 8 Enter subclass(es) for Color:  Blue Red Green
 9 Warning:  Color already exists somewhere in the ontology.
10 Add an inclusion for the current level?[y/n] n
```

Property $\sqsubseteq \top$, Color $\sqsubseteq$ Property,
Blue $\sqsubseteq$ Color, Red $\sqsubseteq$ Color, Green $\sqsubseteq$ Color

Here, $\mathsf{Color} \sqsubseteq \mathsf{Property}$ was already in Tax. As Color was already in $N_C(\text{Tax})$ but $\mathsf{Property}$ not the current A for $\mathcal{P}_A$, a confirmation was asked on line 9.

As a last step in the clean-up process, all axioms in $\mathcal{X}$ that contain the temporal role name $\mathsf{TMPRelated}$ are removed from $\mathcal{T}$. This finalizes the topic ontology $\mathcal{X} = (\mathcal{T}, tc, \phi)$.

CHAPTER 5

The *CN2TopicOnto* Tool

This chapter introduces the *CN2TopicOnto*tool as an implementation of the topic ontology construction method described in chapter 4. *CN2TopicOnto* utilizes *ConceptNet 5.5* [SCH17] as the underlying knowledge graph.
In this chapter we go into some detail on the implementation. In the first section we introduce the ConceptNet data structure and go over some drawbacks of its design. Subsequently we give requirements and a short HOWTO for *CN2TopicOnto*. Later sections then include specifics to the implementation, such as the particular weight function that was applied, and improvement strategies that were employed with regard to the ConceptNet data.

5.1 ConceptNet

ConceptNet calls itself an open, multilingual knowledge graph. It was initially built as part of the *Open Mind Common Sense* project at the MIT Media Lab and had its first API launched in 2011. After multiple updates and structural changes it now contains approximately 28 million assertion statements spread over 304 languages. English is one of the 10 core languages with a vocabulary size of over 1.5 million different expressions. The current version contains dictionary knowledge, integrated from DBPedia, Wiktionary and WordNet, as well as common-sense knowledge, gathered through crowd-sourcing on the internet. Intuitive and human word associations were collected through *Verbosity*, a word game in which participants had to describe terms by filling in sentence patterns and reversely guessed the correct term for the presented sentence patterns [SHS10][HSA+10]. Thus it represents *human* knowledge, often in the form of multi-phrase, complex, natural language expressions.
The ConceptNet relation types capture common patterns from the different data sources. They range from generic, such as *UsedFor* or *HasA*, to rather unusual, with *HasFirstSubEvent, MotivatedByGoal*. Figure 5.1 depicts the internal structure of the ConceptNet

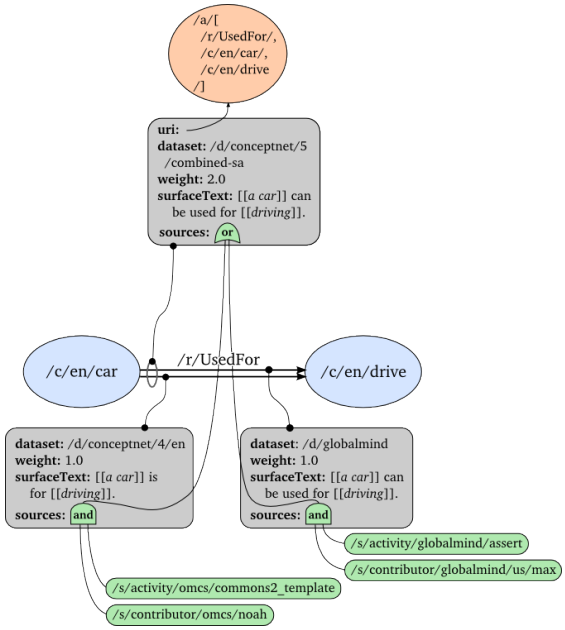


Figure 5.1: Visualization of a ConceptNet assertion. [weba]

assertion *(car, UsedFor, drive)*. Individual edges are first built from different sources and then combined to a final assertion. Each edge has an assigned weight and a short surface text that is associated with the relation type.

The following list summarizes some important design characteristics of ConceptNet. It was slightly adapted from a list of observations made by Caro et al. [CRCB15].

Ambiguity. Everything is considered a "concept". There is no distinction between concepts, named entities or individuals. (There exists an *InstanceOf* relation, extracted from DBPedia, but it only covers a small fraction of the data.)
Often multiple entities exist that express the same semantic concept.
Example: *(apple, CapableOf, fall_from_a_tree)* and *(apple, CapableOf, fall_off_a_tree)*
No distinction between words and their different meanings.
Example: *(orange, IsA, color)* and *(orange, IsA, citrus_fruit)*

Phrases. Many entities represent complete phrases. As a result, long entity names that were coined by humans do not have further connections, due to their specificity.
Example: *(passing_university_exams, HasPrerequisite, studying_for_classes)*
Here *studying_for_classes* is an end node with no further connections, i.e. is not part of any other assertion.

Specificity. It contains very specific semantic information that is difficult to integrate in automated tasks.
Example: *(knowledge, CapableOf, open_human_mind)*

Completeness. It is not complete (due to the methodology used to build it), since

semantic features are arbitrarily associated to only few of all the possible relevant concepts.
Example: ConceptNet contains *(jazz, IsA, styleof_music)* but not *(rock, IsA, styleof_music)*.

Correctness. It contains pragmatic statements which are not semantically correct, but comprehensible and intuitive for humans.
Example: *(cat, Antonym, dog)*, *(sun, Antonym, moon)*

Relativity. It contains human perception and not absolute knowledge.
Example: *(dog, HasProperty, larger_than_cat)* which is not universally true.

Additionally to ConceptNet there exist multiple freely-available knowledge graphs that target general knowledge with varying depth. Examples of knowledge graphs that could be considered for our method are DBpedia, Freebase, OpenCyc or YAGO [FEMR15, FBMR18]. YAGO [SKW07] heavily relies on the Wikipedia categories and WordNet to build its data hierarchy. Thus it contains a lot of non-conditional and encyclopedia information, such as timeline data and geographical locations. Its newly released sister network Webchild 2.0 [TdMW17] on the other hand is a collection of common-sense knowledge with a focus on fine-grained relations, such as tastes, shapes and associated emotions. All its data is automatically extracted from web content, which leads to a high contextual variety of properties that are assigned to individual concepts, and a mix between factual and stereotypical knowledge. WebChilds contents look very promising for our method. However it suffers from that same drawback that all text mining approaches exhibit; it only learns relations between concepts that co-occur in text [PR16]. Cyc [MCWD06] is often mentioned, as it is the only big logic-based ontology system. Unfortunately only a small fraction of the complete network is publicly available as OpenCyc. As such it has been criticized for being restrictive and formally inconsistent [MCWD06].

We chose ConceptNet for our implementation, because it contains a lot of descriptive assertions that link typical common-sense properties and actions to concepts. Through its crowd-sourcing and gamification approach, it contains a lot of knowledge that is too obvious to be explicitly stated in text. Furthermore knowledge is preserved as expressed by humans in a playful setting.
The newest version incorporates semantic vector embeddings, that were calculated and adjusted based on the ConceptNet data. Those embeddings have ranked very high in the SemEval 2017 task and SAT-equivalent question answering [SL17]. This reinforces our assumption that the information contained in the ConceptNet triples is a close representation of human knowledge, as a big part of its content was constructed as well as evaluated under human standards.
Its big variety in data and loose semantic structure makes ConceptNet an interesting underlying knowledge-graph for our topic ontology construction method. Its contents cover a wide range of topics and provide for interesting suggestions that keep the exploration process engaging for the user without getting to technical.

5.2 Requirements and HOWTO

CN2TopicOnto uses the ConceptNet 5.5 REST API to obtain and filter the JSON-LD objects that store its individual assertions. We chose Python as a programming language due to its simplicity and compatibility with the ConceptNet documentation. The ontology-oriented programming module *owlready2* [1] was employed to load and create the OWL 2.0 files. It was designed for the construction of biomedical ontologies, but provides all the means to dynamically create ontologies through Python methods [Lam17]. Additionally it can perform automatic classification via the reasoner HermiT.

To properly run the program Python3 and the latest version of owlready2 are required.

5.2.1 Settings

Before the program is executed for the first time, a target directory has to be set in the *settings.py* document. All OWL files that are created during a run of *CN2TopicOnto* will be created at this location. Note that files are prefixed with the respective topic concept to make them easy to identify.

5.2.2 HOWTO

A complete run can be started with the command `python3 main.py`. The user is prompted to enter a topic concept of choice. If files associated with that topic concept already exist in the set directory, they will be automatically loaded. Otherwise ConceptNet is queried for an entity that matches the chosen topic concept. If found the user needs to confirm the CN entity ID that is then mapped to the topic concept by ϕ. If no matching entity be found it will ask the user to choose something else. Afterwards Step 1 (Section 4.2: Building the Central Taxonomy), Step 2 (Section 4.3: Adding Complex GCIs) and Step 3 (Section 4.4: Clean-Up) are run in succession. Step 2 is split into two substeps, where the first comprises *C1, C2, C3* (General Relations) and the second contains *C4* (User-Specified Roles), which results in a total of four substeps.

Alternatively each of the four substeps can be started individually. In this case two optional parameters, *startstep* and *endstep*, need to be added to specify the individual substeps that ought to be executed. As an example, `python3 main.py 1 2` executes the first and second substep in succession while `python3 main.py 3` executes substep 3 and everything onwards.
After each substep the current ontology is saved in a separate OWL file in the chosen directory. File names are generated automatically and refer to the corresponding substep as well as the topic concept. For the program to run properly, at least the first substep has to be completed in order to execute any subsequent substep.
The main purpose of saving and reloading ontologies is to have the option to stop the process and continue at a later time. It also provides the possibility to exit and restart a substep in case something went wrong or the user is not satisfied with some content that

[1] Documentation is available at https://pypi.python.org/pypi/Owlready2

was created. When a substep is started and there already exists a file with the respective name, then the contained ontology will be loaded automatically and its contents can be further expanded. In order to start a substep from scratch the respective file has to be deleted or moved first.

Those are the files created for $tc = name$:

Mapping ϕ: *name_dic_tax_map.json*
Saved suggestions: *name_dic_dump.json*
Step 1 (run substep 1): *name_tax.owl*
Step 2.1 (run substep 2): *name_GR.owl*
Step 2.2 (run substep 3): *name_UR.owl*
Step 3 (run substep 4): *name_clean.owl*

The ontology files are in OWL/XML format and follow the W3C standards for knowledge representation. They can be loaded with any ontology editor, such as Protégé, for a graphical visualization of the constructed ontology. In such a setting can be used to make further additions and enhancements. It is also possible to directly view newly added axioms in the editor, as all changes are immediately saved.

5.2.3 User Input

Throughout the ontology construction process the user has to frequently decide which concepts, roles and associated axioms should be created. Input is either done by answering simple yes/no questions, by choosing atomic concepts from the suggestions or by entering role names and individuals following the patterns explained in Section 4.3.2. A run can be terminated at any time by typing `exit`.

We briefly summarize the two main input methods and provide an example for each:

Choice by Index The user is presented with a list of suggestions $\mathcal{S}(n, \mathcal{R}e\ell)$, where each suggestion $s \in \mathcal{S}(n, \mathcal{R}e\ell)$ is assigned a numerical index. Elements are then selected through their respective index $i(s) \in I \subseteq \mathbb{N}$. By construction $|\mathcal{S}| = |I|$. After each prompt, indices are entered in a row and need to be separated by a blank character. Order and repetition do not matter. It is possible to add any of the following tags at the beginning of the line to invert the selection: `-i, -I, not, NOT`. If this is done and $\tilde{I} \subseteq I$ is the list of entered integers, then all $i \in I \backslash \tilde{I}$ are selected.
Questions and prompts are designed to ask for events that are likely to require the least amount of input. If the underlying request is assumed to produce good suggestions, then the user will be asked to delete any outliers. If the request is rather universal, then the user will be asked to select the appropriate elements. The inverted selection is for cases where suggestions do not match as well as expected and entering the reverse is less cumbersome.

Choice by index occurs when the central taxonomy is build in Step 1.

Example (Choice by Index):

```
1 Possible subclasses of Animal
2 Beaver[0], Ferret[1], Primate[2], Rabbit[3], Rodent[4]
3 Which concepts do you want to delete?  4 2
```

This deletes Primate and Rodent from the list and the respective ontology concept names will not be created.

```
4 Possible superclasses of Animal
5 LivingCreature[0], Organism[1], Preditor[2]
6 Which concepts do you want to delete?  not 0
```

This will create a concept only for LivingCreature.

```
7 Those might be sub- or superclasses too:
8 Bear[0], Beast[1], Fox[2], LivingBeing[3], Skunk[4], Squirrel[5]
9 Choose subclasses of Animal:  0 2 4 5
```

The same selection can be achieved by:

```
9 Choose subclasses of Animal:  -I 3 1
```

Entering nothing and leaving the input blank implies that nothing will be adapted, deleted or chosen, dependent on the question asked. Similarly entering `all` selects everything.

Choice by Input The user is presented with a list of, often alphabetically ordered, suggestions $\mathcal{S}(n, \mathcal{R}e\ell)$. Those suggestions are then used to build the different types of axioms as explained in Section 4.3.2. Any string can be entered. Using «TAB» tries to auto-complete the input based on the choices offered by $\mathcal{A}uto(\tilde{A})$. Using auto-complete offers a structured visualization of the, often extensive, list of choices.

Choice by input is used to create complex concepts and axioms during Step 2 and Step 3. Note that if nothing is entered at any point, the partially complete axiom is discarded as unfinished.

Example (Choice by Input):

```
1 Do you want suggestions to add axioms for Cake?  [y/n] y
2 [ Sweet, GoodForDessert, False, NiceToEat, Spongy, Yummy ]
3 [ Birthday, Dessert, Food, Icing, Candles, Sweet, Frosting, Pastry,
4 Desert, Birthdays, Baked ...  ]
5 Generate a (new) axiom?  [y/n] y
6
7 relation name:  IsEatenFor
8 Cake.IsEatenFor.
```

```
9   Bake                Cream           LayeredDessert
10  Baked               Cylinder        Layers
11  BakedDessert        Delicious       Lie
12  BakedGood           Desert          NiceToEat
13  Bakery              Dessert         Oven
14  Baking              DessertFood     Parties
15  Birthday            Eating          Party
16  BirthdayCandles     Edible          PartyFood
17  BirthdayConfection  False           Pastry
18  BirthdayDesert      Flour           Pie
19  BirthdayDessert     Food            Pudding
20  BirthdayFood        ForBirthday     Round
21  BirthdayParties     ForBirthdays    Slices
22  BirthdayParty       Frosted         Sponge
23  BirthdayPastry      Frosting        Spongy
24  BirthdaySweet       Good            Sugar
25  BirthdayTreat       GoodForDessert  Sweet
26  Birthdays           Has             SweetBread
27  Bread               HasCandles      SweetFood
28  Candles             HasFrosting     Tasty
29  CandlesIcing        HasIcing        Treat
30  Celebration         Iced            Wedding
31  Chocolate           Icing           Weddings
32  Circle              IcingCandles    WithCandles
33  Confection          Item            Yummy
34  Cookie              Layered

35  Cake.IsEatenFor.Birthday Des
36  Desert    Dessert    DessertFood
37  Cake.IsEatenFor.Birthday Dessert Party Celebration
```

All suggestions are presented as a nice view and are selected through auto-complete. Line 35 and 36 show how starting an input with `Des` provides the remaining suggestions `Desert Dessert` and `DessertFood`.

5.3 Adjusted Functions

This section describes the ConceptNet assertion weights and the specific threshold function Θ that was implemented, as well as the implemented naming function *onto*. The final subsection contains a tabular overview and some details on the different sets of request types $\mathcal{R}el_{\sqsubseteq}, \mathcal{R}el_{\sqsupseteq}, \mathcal{R}el_{\cong}, \mathcal{R}el_{part}, \mathcal{R}el_{loc}$ and $\mathcal{R}el_{cap}$ that we used for our implementation.

5.3.1 ConceptNet Weights and the Employed Threshold Function

ConceptNet weights assertions based on how often respective contexts occur over different sources. Thereby an assertion a from reliable source is assigned a weight of $\omega(a) = 1$. Weights are additive over different linguistic structures that lead to the same assertion, consequently causing weights that are greater than 1. As a result, high weights are often assigned to frequently used concepts. Observations lead to the assumption that $\omega(a) = 1$ is the most prevalent for any assertion a, whereas assertions with weight strictly below one or above two rarely occur. Because it is so frequent, the accurateness of assertions a with $\omega(a) = 1$ varies a lot. A correlation between weight and significance only becomes visible, the more weights deviate from 1. This leads to our hypothesis that a statement about the relevance of an assertion can only be made if its weight is strictly lower or strictly greater than 1.
These observations motivate our choice for k_i in the aggregate function $\hat{\omega}$, the specific threshold function Θ that we employ. The m_i were selected with the intention to preserve ten suggestions with high weight per request, though we considered a value greater than 1 already relevant enough to justify the reduction of the limit to five. Ultimately we decided on two cardinality limits, that induce two different cut-off points, one at 2 and the other strictly above 1. Results with $\omega(a) < 1$ for the respective assertion a are always cut off.

Definition 5.3.1 (Threshold Function)**.**
Let $\omega(a)$ be the weight of a ConceptNet assertion $a = (n, r, n')$ with two nodes n, n' and associated relation type r, and $t \in \{s, e, b\}$ a request type.
Then we define the threshold function Θ *for ConceptNet as*

$$\hat{\omega}(Req_t(n,r)) = \begin{cases} 2.0 & if\ |\{(n', \omega(a)) \mid \omega(a) \geq 2.0\}| \ \geq 10 \\ 1.1 & if\ |\{(n', \omega(a)) \mid \omega(a) \geq 1.1\}| \ \geq 5\ and \\ & \quad |\{(n', \omega(a)) \mid \omega(a) \geq 2.0\}| \ < 10 \\ 1.0 & else \end{cases} \tag{5.1}$$

Corollary 5.3.1.
For every entity n and set $\mathcal{R}e\ell$ it holds that, if $|\{(n', \omega(a)) \in Req_t(n,r) \mid \omega(a) \geq 1\}| \geq 5$ for a single $(r,t) \in \mathcal{R}e\ell$ and onto $|_{\widetilde{Req}(n,\mathcal{R}e\ell)}$ is bijective, then $|\mathcal{S}(n, \mathcal{R}e\ell)| \geq 5$.

Proof. We make two observations about $\mathcal{S}(n, \mathcal{R}e\ell)$ and $\widetilde{Req}(n, \mathcal{R}e\ell)$ that hold for any n.

- Observation 1: $|\widetilde{Req}(n, \mathcal{R}e\ell)| \leq |\bigcup_{(r,t)\in\mathcal{R}e\ell} Req_t(n,r)|$
- Observation 2: $|\mathcal{S}(n, \mathcal{R}e\ell)| = |\widetilde{Req}(n, \mathcal{R}e\ell)|$ if *onto* $|_{\widetilde{Req}(n,\mathcal{R}e\ell)}$ is bijective.

Now three different cases can occur:

Case 1: $\max_{(r,t)\in\mathcal{R}e\ell} \hat{\omega}(Req_t(n,r)) = 2.0 \Leftrightarrow$ there exists some $(r,t) \in \mathcal{R}e\ell$ s.t. $|\{(n', \omega(a)) \in Req_t(n,r) \mid \omega(a) \geq 2.0\}| \geq 10$. Then $|\widetilde{Req}(n, \mathcal{R}e\ell)| \geq 10$. From Observation 2

it follows that $|\mathcal{S}(n, \mathcal{R}e\ell)| \geq 10 \geq 5$.

Case 2: $\max_{(r,t)\in\mathcal{R}e\ell} \hat{\omega}(Req_t(n,r)) = 1.1 \Leftrightarrow$ there exists some $(r,t) \in \mathcal{R}e\ell$ s.t. $|\{(n', \omega(a)) \in Req_t(n,r) \mid \omega(a) \geq 1.1\}| \geq 5$. Again it follows that $|\mathcal{S}(n, \mathcal{R}e\ell)| \geq 5$.

Case 3: $\max_{(r,t)\in\mathcal{R}e\ell} \hat{\omega}(Req_t(n,r)) = 1.0$. From the assumption it holds that $|\{(n', \omega(a)) \in Req_t(n,r) \mid \omega(a) \geq 1\}| \geq 5$ for some $(r,t) \in \mathcal{R}e\ell$. It follows that $|\widetilde{Req}(n, \mathcal{R}e\ell)| \geq 5$ and $|\mathcal{S}(n, \mathcal{R}e\ell)| \geq 5$.

□

Intuitively corollary 5.3.1 states that with $\hat{\omega}$, at least five suggestions will be proposed for any filtered request $\widetilde{Req}(n, \mathcal{R}e\ell)$ for a set $\mathcal{R}e\ell$, if the underlying knowledge graph contains enough reliable assertions whose entities represent concepts with sufficiently different labels.

5.3.2 Surjectivity of the Naming Function *onto*

Due to the phrase-based nature of ConceptNet labels, we chose to remove filler words for the transformation of entities into concept names. As a result, it is possible for *onto* to map two different ConceptNet entities to the same concept name. If this is the case then the concept name will be proposed to the user twice and the selection decides what ϕ^{-1} maps to. An alternative solution could be to choose one at random or, more sophisticated, to save both and later unify relevant assertions for both entities.

5.4 ConceptNet Relation and Request Types

The choice and number of relation types accounts for one of the fundamental design differences between knowledge graphs, and engineered ontologies. While the first contain a fixed, limited set of relation types defined by textual schemata, the latter focus on incorporating distinct and fitting role names on a case to case basis.
DBpedia, for example, automatically extracts its relation types (called properties) from Wikipedia. As a result, they have been criticized as unclear and semantically overlapping [FEMR15]. Similar to ConceptNet, which has a core set of 36 relation [SCH17], YAGO defines 106 distinct relations, yet incorporates very generic ones such as *has-Created*. In many knowledge graphs a discrepancy can be observed between small sets of relations that are used a lot and a majority of relations which are only used once or twice [FBMR18]. This is reinforced by the fact that unrecognizable patterns are mapped to unspecific relations, such as ConceptNets *RelatedTo*, which results in them being overrepresented. As a consequence relation types can not be straightforwardly converted into role names. For more granular and specific names, our method proposes the suggestions according to the sets $\mathcal{R}e\ell$ defined in Table 5.1 and the user can add role names as needed. Section 5.4 provides some detail on the meaning of the used ConceptNet relation types.

Taxonomy (Step 1)	
	$\mathcal{R}e\ell_{\sqsubseteq} = \{(IsA,s)\}$ $\mathcal{R}e\ell_{\sqsupseteq} = \{(IsA,e)\}$ $\mathcal{R}e\ell_{\cong} = \{(RelatedTo,b)\}^*$
Complex GCIs (Step 2)	
C1	$\mathcal{R}e\ell_{part} = \{(HasA,e),(PartOf,s)\}$ $\mathcal{R}e\ell_{part^-} = \{(HasA,s),(PartOf,e)\}$
C2	$\mathcal{R}e\ell_{loc} = \{(AtLocation,s)\}$, $\mathcal{R}e\ell_{loc^-} = \{(AtLocation,e)\}$
C3	$\mathcal{R}e\ell_{cap} = \{(CapableOf,e),(ReceivesAction,e)\}$ $\mathcal{R}e\ell_{cap^-} = \{(CapableOf,s),(ReceivesAction,s)\}$
C4	$\mathcal{R}e\ell_{sp} = \{(HasProperty,b),(RelatedTo,b)\}$

Table 5.1: Categorized pairs of ConceptNet relation types and request types that were used in the implementation of *CN2TopicOnto*.
* only applied to *tc*.

5.4.1 Taxonomy

To build the taxonomy, suggestions are proposed through $\mathcal{R}e\ell_{\sqsubseteq} = \{(IsA, s)\}$ and $\mathcal{R}e\ell_{\sqsupseteq} = \{(IsA, e)\}$. Additionally a second set of suggestions is proposed for the generation of sub- or superclasses of *tc* through $\mathcal{R}e\ell_{\cong} = \{(RelatedTo, b)\}$.
The symmetric relation *RelatedTo* collects all types of associations that could not be classified under a specific pattern. We observed that respective suggestions frequently contain additional concepts names that were not properly captured by the subsumption relation. The choice to only propose them for *tc* was made to increase the amount of central concepts, while restricting the exponential growth of prompts to the user. For additional optimization, the nountag */n*, that was adopted from WordNet into ConceptNet is included in some queries. It has shown to retrieve very good results for domain specific subtypes, such as concept varieties (for example fruit types) or named entities.

5.4.1.1 Optimization of Taxonomy Suggestions with Vector Embeddings

Together with the ConceptNet 5.5. knowledge graph, Rob Speer et al. released a hybrid semantic space named *ConceptNet Numberbatch*. Its vector embeddings of concepts (= terms) are learned from distributional semantics that are enhanced by ConceptNets knowledge with a generalization of the retrofitting method [SCH17]. Numberbatch outperformed other systems in the recent SemEval-2017 Task for "Multilingual and Cross-Lingual Semantic Word Similarity" [SL17] and ranked only a few percent lower than humans on a corpus of SAT-style analogy questions [SCH17].

Taxonomy (Step 1)	
IsA	A is a subtype or a specific instance of B; every A is a B. This is the hyponym relation in WordNet.
*RelatedTo**	The most general relation. There is some positive relationship between A and B, but ConceptNet can't determine what that relationship is based on the data. Symmetric Relation.
Complex GCIs (Step 2)	
HasA	B belongs to A, either as an inherent part or due to a social construct of possession. HasA is often the reverse of PartOf.
PartOf	A is a part of B. This is the part meronym relation in WordNet.
AtLocation	A is a typical location for B, or A is the inherent location of B. Some instances of this would be considered meronyms in WordNet.
CapableOf	Something that A can typically do is B.
ReceivesAction	[No description]
HasProperty	A has B as a property; A can be described as B.
RelatedTo	The most general relation. There is some positive relationship between A and B, but ConceptNet can't determine what that relationship is based on the data. Symmetric Relation.

Table 5.2: Overview of the ConceptNet relation types implemented in *CN2TopicOnto*. The rightmost column provides sources and semantic use of each relation type according to the ConceptNet Wiki.[1]

[1] https://github.com/commonsense/conceptnet5/wiki/Relations

The ConceptNet API offers access to the Numberbatch embeddings. It is possible to query for a list of most *related* terms w.r.t Numberbatch embeddings for any concept, as well as request a *relatedness* measure (in $[-1, 1]$) for a pair of concepts.

While semantic embeddings are not the focus of this thesis, we made an attempt to optimize the taxonomy suggestions utilizing Numberbatch, to eliminate those suggestions that do not score a relation to *tc* above a certain threshold. The objective was to remove suggestions with an incorrect meaning for disambiguous concepts. In general polysemous concept names are not considered good ontology engineering practice and designing them is listed as number one of common pitfalls that can not be automatically detected by ontology evaluation tools yet [PSG12].
Even for the small experimental domains we created, it was very hard to determine an appropriate threshold. Further research would be needed to assess the rates of false positives and true negatives in correlation to the frequency of occurrence of *tc* in

natural language and the chosen threshold. Nevertheless example data showed that some thresholds worked well for specific domains. A run with $theshold = 0.25$ for the Fruit topic ontology, worked well enough to remove ComputerBrand as a possible superclass of Apple, as well as Bird and Animal for Kiwi. BirdFromNewZealand, despite being very specific, was still suggested. For Date more than 30 suggestions that were related to the temporal meaning of the concept were correctly removed as well as 8 references to Orange as a color. In contrast the threshold for the Vehicle ontology had to be set much lower to prevent the removal of many good suggestions.
The user can experiment with the Numberbatch optimization by setting the activation tag and a threshold in the *settings.py* document.

5.4.2 Complex GCIs

For part-whole suggestions ($\mathcal{R}el_{part}$) we selected the two relation types *HasA* and *HasPart*. ConceptNet unfortunately does not distinguish between the different forms of meronym relations. We noticed that corresponding assertions often express linguistic context, where one single term is *part of* a coined term, as in *cranberry is part of american cranberry*. Such suggestions are hard to filter out without removing false positives, as the example of *wheel is part of wheeled vehicle* shows. A solution would be to remove all multi-word suggestions that have an exact match with the concept name, which they are suggested to have as part. On the downside this also removes any suggestion that was misclassified, and could be suitable to integrate into the taxonomy retroactively, e.g. *house is part of house boat*. After some consideration, we decided to only remove all concepts that are already part of the taxonomy from those suggestions, to prevent incorrect ambivalence between *is-a* and *part-of*.

For possible locations ($\mathcal{R}el_{loc}$) we chose only one relation type, namely *AtLocation*. The relation *LocatedNear* could be suitable, but was not considered because suggestions were plenty.

Suggestions with $\mathcal{R}el_{cap}$ use *CapableOf* and *ReceivesAction*, and contain lots of complete phrases. For example $\mathcal{S}$(Water,$\mathcal{R}el_{c}ap$) include ExtinguishFire and FillBucket. While phrases are not desirable for concept names, they can indicate fitting role names.

For the user-specified roles $\mathcal{R}el_{sp}$, we selected imprecise relation types, namely *HasProperty* and *RelatedTo*, which are used to assert descriptive adjectives and physical or temporal states.

CHAPTER 6

Ontology Engineering

This chapter discusses some related work in the area of *Ontology Engineering*, a vast research field that deals with decisions and activities that concern the ontology development process [CFG06]. It comprises research on *methods and methodologies* for ontology design and evaluation, and addresses the development of *ontology languages* and *ontology tools*, such as WebOnto [Dom98], Protégé [Mus15] and OntoEdit [SEA+02].

There are many works proposing methods and tools to support ontology development that are related to this work. We briefly discuss the ones we find most relevant.

As a first step, most *methodologies* emphasize *purpose identification* to make clear why the ontology in being built and what competency questions it needs to be able to answer. When building an ontology, the identification and integration of main concepts can either be done with a top-down, a bottom-up, or through a mixed approach. Many methodologies, such as the one by Grüninger and Fox [GF95] and the METHONTOLOGY methodology [FLGPJ97], have their roots in knowledge-based system development [FL99, CFG06]. Grüninger and Fox is a very formal method, that uses first-order-logic to define terms and constraints for objects. METHONTOLOGY proposes an ontology life cycle based on evolving prototypes that go through different stages, such as *knowledge acquisition*, *integration*, and *evaluation*.

Next to methods and methodologies, multiple guidelines for proper ontology engineering practices exist, for example the *Stanford 101 Ontology Development Guide* [NM01] or the overview of common pitfalls [PSG12]. Alan Rector [Rec03] and Ulrike Sattler [BHLS17] are established names in the field and their publications and course materials discuss ontology engineering with DLs from a practical perspective.

Ontology Learning is a huge field in the area of machine learning, dedicated to automatic improvement strategies that learn axiom schemata, such as class definitions, from existing partial ontologies or instance data. Well-known methods are based on *Inductive Logic Programming* and need positive and negative assertion examples to extract new hypotheses.

For example DL-FOIL [FdE08] and OCEL [LH10] are algorithms that learn new concept descriptions, and Fleischhacker et al. introduced a method to automatically learn disjointness axioms [FV11]. As machine learning algorithms they learn patterns and characteristics from examples, which presumes the availability of suitable initial ontologies. A newly published approach employs learning via queries to an oracle [KLOW17].

Knowledge Elicitation (KE) [SB89], as an area of research itself, focuses on standardizing strategies for humans to collect and structure knowledge in an approachable way, to bridge the gap between domain experts and ontology engineers that have limited knowledge on the domain, but need to construct an adequate representation. Methods, such as *card sorting* and *laddering* [HD03] are traditional forms of KE techniques, that are employed to structure domain knowledge. KE techniques are also used to develop competency questions [RGROB08] which in turn are important to evaluate and ensure the quality of an ontology.

Ontology Authoring tools, such as Protegé [Mus15], often come with integrated reasoning extensions for debugging and consistency checking. They are essential for ontology engineering, but suffer from complexity issues as content manipulation is oftentimes not straightforward for people that are not proficient in logic. *Competency Question-Driven Ontology Authoring* [RPM$^+$14] is a method that tries to give feedback to the user, based on patterns in natural language competency questions, to assure that the ontology models the knowledge as expected. The Protegé plug-in OntoComp [BDRR11] supports ontology completion by asking questions to check whether the considered ontology contains all the relevant domain information. If information is missing, it extends the ontology appropriately based on the user responses. Question-driven methods however require the ontology engineer to be knowledgeable about the domain.

At last, Text2Onto [CV05] should be mentioned. It is a text processing framework that learns concept names and term hierarchy from written text. It describes itself as an ontology learning tool, but incorporates user interaction and is overall geared to simplify the KE and submission process for the user. OntoGen [FGM07] achieves the same goal by combining text-mining techniques on a user-provided text corpus with an efficient user interface to reduce the time spent for the user to retrieve and structure relevant contents. The method presented and implemented in this thesis follows a similar idea, as it simplifies KE for non-expert users by providing suggestions, but in contrast suggestions are extracted from triples of a common-sense knowledge graph instead of unstructured text. We are not aware of any similar tool that takes advantage of existing knowledge graphs to support the ontology engineering process.

Below we discuss some modeling techniques and challenges of ontology engineering specifically in relation to the proposed ontology construction method and offers possible solutions to aid with the decisions that need to be made during the construction of topic ontologies with *CN2TopicOnto*.

6.1 Top-down and Bottom-Up Approach

The *Stanford Ontology Development 101 Guide* [NM01] states the following as one of the fundamental principles of ontology design:

> "There is no one correct way to model a domain — there are always viable alternatives. The best solution almost always depends on the application that you have in mind and the extensions that you anticipate."

Our method for topic ontology construction supports the user with suggestions of fitting concept names and restrains the types of available axioms, but the ultimate choices for content and structure remain with the user. Therefore it mainly assists with KE and proposes a broad structure determined by the relation types of the underlying knowledge graph.

There are two main elicitation philosophies for the construction of an ontology hierarchy from scratch [NM01][UG96]:

- *Top-down* development starts by defining the most general domain concepts that should be included. Then more specialized subconcepts are added to further categorize the existing concepts.
- A *bottom-up* approach, on the contrary, starts with definitions of the most specific concepts and groups those into further (super)concepts based on common properties.

Mixed approaches combine both methods by defining the most salient concepts first and generalizing and specializing them later as needed.

Standard manual KE techniques, such as *card-sorting* or *laddering*, have been developed in the early 90s to help organize domain knowledge and categorize concepts. They require the user to create physical cards for relevant concepts and subsequently sort them into the desired categories. Wang et al. [WSSR06] released a plug-in for Protegé, that integrates those manual processes into the ontology authoring tool and makes them more accessible for users.

Our method for ontology construction that we introduced in Chapter 4 utilizes a mixed approach in which the topic concept, as the "most salient" concept, is used as the sole base for any further expansion of the ontology. All other concept names are explored and added interactively during the construction process. Dynamic extensions with subclasses are a built-in procedure to recognize and add new generalizing concepts throughout the development process in a bottom-up fashion. As an example, assume that Africa, Paris, Box, Crate are suggested as different possible locations for a concept. The taxonomy can then be immediately extended with two superclasses, GeographicalLocation and Container, that subsume Africa, Paris and Box, Crate respectively. Subsequently the hierarchy can be extended with more specialized concepts for geographical locations, such as Continent and

City. This way the process iterates between two steps: 1. KE in the form of suggestions, and 2. creation and integration of concepts into the ontology.

Maintaining an overview may be challenging, as semantically similar concept names need to be integrated into an existing (partial) taxonomy. Due to the gradual construction process and the incompleteness of most knowledge graphs, the resulting concept hierarchy will likely have mixed levels of generality. This means that subconcepts of the same concept do not represent the same level of generality. The taxonomy modification step is an attempt to alleviate such inhomogeneity, but nonetheless intermediate concept categories might not be suggested and therefore never added to the taxonomy in the first place. For example, while concepts such as NocturnalAnimal, AquaticAnimal and ColdBloodedAnimal are suggested as subconcepts of Animal, not all immediate subclasses end up being animal categorizations. One could even argue that a further distinction is needed as the subtypes describe intrinsically different properties, i.e. active time, habitat and thermophysiology. In an optimal categorization, each animal would be subconcept of a concept partition into sibling concepts that group animals after a single descriptive property. Such a categorization is not the purpose of the created topic ontologies, as accurate and complete classifications quickly become infeasible. Instead the user should at first aim to select the most defining and common-sense properties. If further categorization is need, missing concepts (DiurnalAnimal, WarmBloodedAnimal) and their ϕ-mapping can be added using the IsA role name.

6.2 Equivalence and Normalization

Equivalences of the form $A \equiv C$, where A is an atomic concept and C any concept, are prevalent in well-designed ontologies and provide strict definitions for concept names. Examples of such definitions are $\mathsf{Herbivore} \equiv \forall\ \mathsf{HasFoodSource.Plant}$ or $\mathsf{CheeseBurger} \equiv \mathsf{Burger} \sqcap \exists\ \mathsf{HasTopping.Cheese}$. Knowledge-graphs, in contrast to encyclopedias, are not designed to provide closed definitions of concepts. For this reason our topic ontologies do not embed definitions as axioms. Instead we want the user to be able to add necessary conditions through GCIs ($\mathsf{CheeseBurger} \sqsubseteq \mathsf{Burger}$, $\mathsf{CheeseBurger} \sqsubseteq \exists\ \mathsf{HasTopping.Cheese}$). If needed, and after the topic ontology is finished, axioms can be extended or combined to definitions in an ontology editor of choice.

The only type of equivalence axiom that we support is of the form $A_1 \equiv A_2$, where $A_1, A_2 \in N_C$. It is debatable if such axioms should be allowed in an ontology. The *Catalogue of Common Pitfalls* [OOP] assembled by the Ontology Engineering Group of the Technical University of Madrid, which is used as a basis for the *OOPS!* pitfall scanner [PGS14], lists "creating synonyms as classes" as a common ontology engineering error. The *Ontology Development Guidelines* [NM01] state that "Synonyms for the same concept do not represent different classes." Our method still allows the user to create equivalent concepts for synonyms, as the axiom $\mathsf{Orca} \equiv \mathsf{KillerWhale}$ that is part of our Animal example ontology, shows. We made this decision to cover two possible occurrences: 1. the user accidentally creates a cycle in the taxonomy of $\mathcal{X}$ where $A_1 \sqsubseteq^*_{\mathrm{Tax}(\mathcal{X})} A_2$ and

$A_2 \sqsubseteq^*_{\mathrm{Tax}(\mathcal{X})} A_1$ through multiple single ACIs that s/he adds at different points of the construction process, or 2. the user does not remember that a conceptualization was already added to the taxonomy and subsequently introduces a second concept name. He or she can then add an equivalence axiom (using the IsA role) when the error is recognized. By allowing synonym equivalences, different synonymous concept names can be initially added and the decision on which name to keep, postponed for later. If one concept name is later on deleted in an editor, corresponding axioms are not lost. Many systems allow associating lists of synonyms to a concept, which would also be a retroactive solution that can be straightforwardly applied in an ontology editor.

In connection with equivalence and concept definitions, we want to call attention to, and briefly introduce, a mechanism called *normalization*. It was introduced by Alan Rector [Rec03] as a method to achieve modularity for OWL or DL-based ontologies. The essence of his proposal is that the primitive skeleton of an ontology, which he defines as the part of the ontology that contains only concepts described by necessary conditions, should consist of disjoint homogeneous trees. The primitive skeleton should clearly distinguish between *self-standing concepts*, which are defined as things and intangible notions, such as colors, and *refining concepts*, which are used to describe value types. While self-standing concepts are always open and their list of children not exhaustive, refining concepts define a closed partition of a concept into subconcepts, for example as "small", "medium" and "large". Normalization is definition-driven in the sense that it introduces definitions to link the independent branches of the untangled taxonomy skeleton. For example, instead of having two axioms RedApple $\sqsubseteq$ Apple and RedApple $\sqsubseteq$ Red, following normalization, Apple and Red should be self-standing concepts (Apple $\sqsubseteq$ Fruit, Red $\sqsubseteq$ Color). Then RedApple is defined separately through an axiom RedApple $\equiv$ Apple $\sqcap \exists$ HasColor.Red. Other examples are Mammal $\equiv \exists$ HasSpecies.MammalSpecies, where MammalSpecies is one of multiple (non-exhaustive) subclasses of Species and HotFood $\equiv$ Food $\sqcap \exists$ HasLevelOfSpiciness.High. Here there different levels of spiciness can build an exhaustive partition of a concept LevelOfSpiciness into subconcepts High, Medium and Low. With such a structure, RedApple, Mammal and HotFood are not part of the primitive skeleton of the respective ontology as they are more complex to describe.

We propose to keep the concept of normalization in mind when using *CN2TopicOnto* in case it is not straightforward how conceptual notions should be separated and structured when building and extending the taxonomy.

6.3 Modeling Choice: Concept or Individual

Defining a proper model is challenging in diversified domains that describe abstract concepts. While it is easy to include statements such as Female(mary) or Physicist(albert_einstein), it is often not clear and up to the engineer to decide if a term is better treated as a concept name or as an individual. *Modifiers*, which are adjective and adverbs that modify other concepts, such as *colors* or *sizes* occur as suggestions across topics and can be handled in different ways.

There exist three main approaches:

- Model modifiers as new atomic concepts.
  RedApple $\sqsubseteq$ Apple, GreenApple $\sqsubseteq$ Apple,
  RedApple(a1)

  Unless the domain is naturally partitionable into individual subconcepts this approach introduces a lot of new artificial concept names.

- Model modifiers as individuals.
  Color(red), Color(green), Apple(a1), Apple(a2),
  HasColor(a1, red), HasColor(a2, green),

  Apple $\sqsubseteq \exists$HasColor.{red, green}, Banana $\sqsubseteq \exists$HasColor.{yellow}

  If it does not seem natural to have multiple individuals of a modifier, it is best to group multiple modifiers and introduce them as individuals of one common atomic concept.

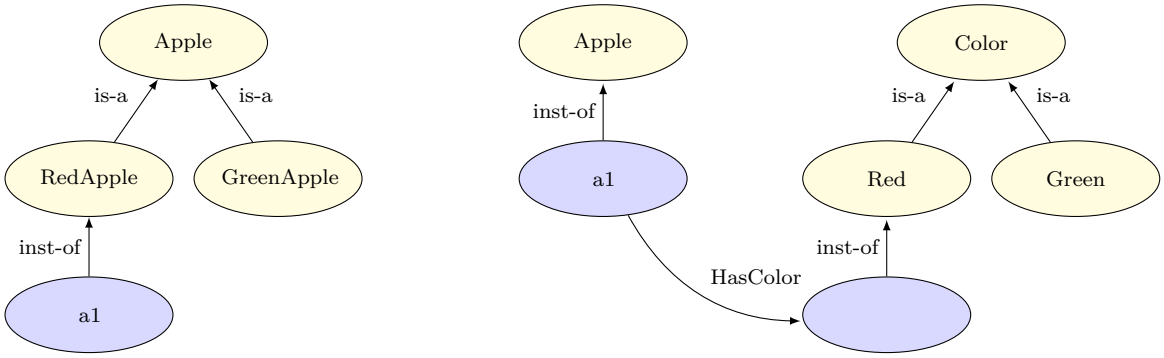
- Model the modifiers with anonymous individuals.
  Red $\sqsubseteq$ Color, Green $\sqsubseteq$ Color
  Apple(a1)

  Apple $\sqsubseteq \exists$HasColor.(Red $\sqcup$ Green)

  As a downside of this approach, it can be unintuitive to construct instances of some abstract modifying concepts. While it works well in Red(burgundy), it seems forced for Black(black).

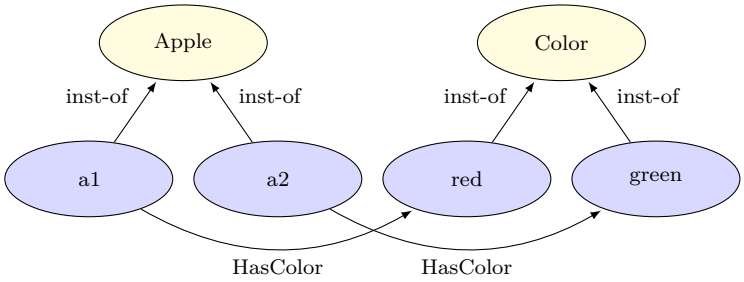
Note that for the topic ontologies we construct, an ABox assertion $A(a)$ is expressed through the TBox axiom $\{a\} \sqsubseteq A$. With the exception of role membership assertions, all axioms can be easily created through dynamic extensions. Figure 6.2 visualizes the different approaches for the modifying concept Color.

Metamodelling [BN03] is an approach in which an entity is allowed to be both, a concept and an individual, simultaneously. Thereby concepts may be defined as subsets of other concepts. In the Animal ontology this would permit the instantiation of AnimalSpecies with different concepts of species, such as Mammal and Fish. Then it is possible to state that every concept that is an instance of AnimalSpecies exhibits a biological classification. At the same time the information that Mammal and Fish are subclasses of Animal is not lost. To pick up the color example: with metamodelling, individual colors may be coded as concepts while making it possible to express that every object that has more than two colors, is considered multicolor.
While metamodelling is not allowed in standard DLs, it is supported by OWL2, and could be considered for integration into *CN2TopicOnto* in the future.



(a) Modifiers as specialized concepts

(b) Modifiers as anonymous individuals



(c) Modifiers as individuals

Figure 6.2: Modeling options for modifiers, adapted from slides by Ulli Sattler [Sat].

CHAPTER 7

Topic Ontology Examples

This chapter contains a statistical overview and excerpts from the topic ontologies that we constructed with *CN2TopicOnto*. The full ontologies can be downloaded at https://bitbucket.org/Tenebrumm/cn2topiconto.git.

The final topic ontologies have been revised and edited with Protégé 5.2.0 [Mus15]. Changes include the removal of axioms that were created by accident, minor changes in role and concept names for more clarity (this may have caused the loss of some mappings), and addition of axioms in extensions of DL that are not supported at the current time (such as number restrictions and role hierarchies), if adequate.

Table 7.1 shows some parameters of the constructed topic ontologies. The numbers in the first three columns refer to the amount of concept, role and individual names. The fourth column shows the amount of logical axioms contained in $\mathcal{T}$. $Dom(\phi)$ denotes the domain of ϕ. The last column gives a rough estimate of the time it took to create each ontology.

tc	$N_C(\mathcal{X})$	$N_R(\mathcal{X})$	$N_I(\mathcal{X})$	$\lvert\mathcal{T}\rvert$	$\lvert\mathsf{Dom}(\phi)\rvert$	Time
Animal	671	67	50	1253	347	∼6h
Vehicle	300	34	7	483	128	∼5h
Fruit	312	35	0	568	125	∼5h
NaturalDisaster	89	16	4	150	35	∼1.5h
FastFood	109	20	0	200	22	∼2.5h
University	153	15	5	233	46	∼3h

Table 7.1: Statistics of constructed topic ontology examples.

We briefly describe the first four listed ontologies, namely Animal, Vehicle, Fruit and NaturalDisaster, next. For each of them we give an informal description of its contents,

list the role names, and provide a visualization of a selected part of the taxonomy together with a few selected axioms. In the end some comments provide a few details and insights that are specific to the respective topic ontology.

7.1 Animal

The Animal ontology is the largest among the example ontologies that we constructed, with more than double the amount of atomic concepts and almost double the amount of role names than the second biggest Vehicle ontology.

Contents:
animal species, animal groupings (cold-blooded, herd animal etc.), habitats, geographical location, associated activities, colors, size, leg count, danger potential, animal sounds,

List of role names:
AbleToLayEgg, AidOfProfession, AidWithActivity, AssociatedWithActivity, BiologicalProcessOf, Build, CanBeFoundIn, CanBeFoundUnder, CanBeKeptIn, CapableOfNurse, CarriesPotentialDisease, Cause, CausedBy, FormedOf, FoundInGeographicalLocation, HarvestedBy, HasAnatomicalPart, HasBehaviourCharacterstic, HasBodyPart, HasBreed, HasCharacteristic, HasCharacteristicAnatomicalFeature, HasForSale, HasGender, HasLateStage, HasLegNumber, HasLivingState, HasNaturalHabitat, HasNaturalPrey, HasOrigin, HasPart, HasPreferredDrink, HasPreferredFood, HasPreviousStage, HasProfessionActivity, HasSenseOfHearing, HasSenseOfSight, HasSenseOfSmell, HasSizeValue, HasSpecies, HasTypicalColor, HeldIn, HeldOn, IsMemberOf, KeptForPurpose, KnownAs, LiveIn, LivesOn, MakeAnimalSound, MarkTerritory, MeatOf, NaturalHabitatOf, PartOf, Possess, ProducedBy, SignOf, SimilarFeatureTo, TransmitDisease, TypicalFoodOf, TypicallyHasBreed, UsableOn, UsedForFoodProductionOf, UsedToBreed, UsedToHold, UsedToScare

Figure 7.1: $\mathcal{CT}$ of the Animal ontology. Deeply nested animal subspecies have been omitted for space reasons.

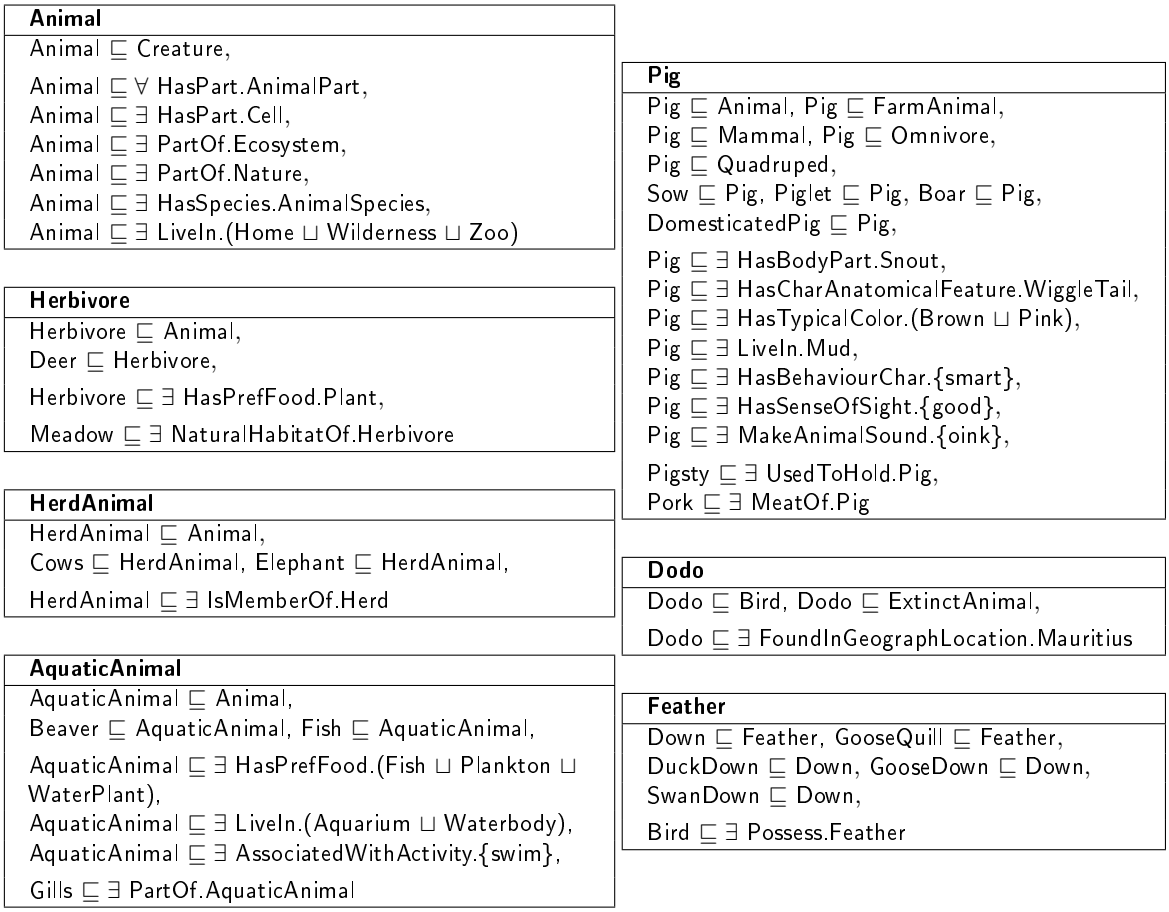


Figure 7.2: Selected axioms from the Animal ontology.

Comments:
Constructing the Animal ontology was a long time commitment, as the taxonomy grew very fast and was on the verge of becoming infeasible to manage. Suggestions included an excessive amount of biological and species-related terminology that were hard to relate for a layperson.
In the end some well-thought-out restructuring was needed to find a satisfactory way to model the different animal habitats and typical living locations.
There were a couple of nice opportunities to introduce number restrictions, which were added retrospectively via Protégé, for example by defining the number of legs of an animal, as in Quadruped $\sqsubseteq \exists$ =4 Leg.
Typical activities were grouped and modeled as individuals of the concept Activity, the same for BehaviourCharacteristics and AnimalSounds. This is why the vocabulary of the Animal ontology, in comparison to the other ontologies, contains a large number of individual names. Examples for axioms that contain individuals can be seen in Figure 7.2 in the bordered section for the concept Pig.

7.2 Vehicle

The Vehicle ontology contains exactly 300 atomic concepts and took almost as long to construct as the Animal ontology.

Contents:
different forms of vehicles (air, land, space, water), vehicle groupings by usage (agricultural, emergency etc.), vehicle groupings by type (electric, single-passenger, tracked etc.), professions, vehicle parts (especially car parts), engine types, land surfaces and water surfaces, events, colors, sizes, car brands

List of role names:
ArriveAtTime, DrivingAreaFor, EquippedWith, HasApplicationEnvironment, HasColor, HasDestination, HasEngine, HasForDisplay, HasForSale, HasModeOfTransportation, HasMotor, HasPart, HasPreviousOwner, HasSize, HasWorkEnvironment, HasWorkVehicle, Involve, LandingAreaFor, MadeFrom, MovedBy, Need, OptionalEquipmentOf, PartOf, StationedAt, Tow, TransportationMeanOfChoiceInCountry, UsedForPark, UsedToMove, UsedToTransport, UsedToUnlock, UsedWithPurpose, Utilize, WorkEnvironmentOf, WorkVehicleOf

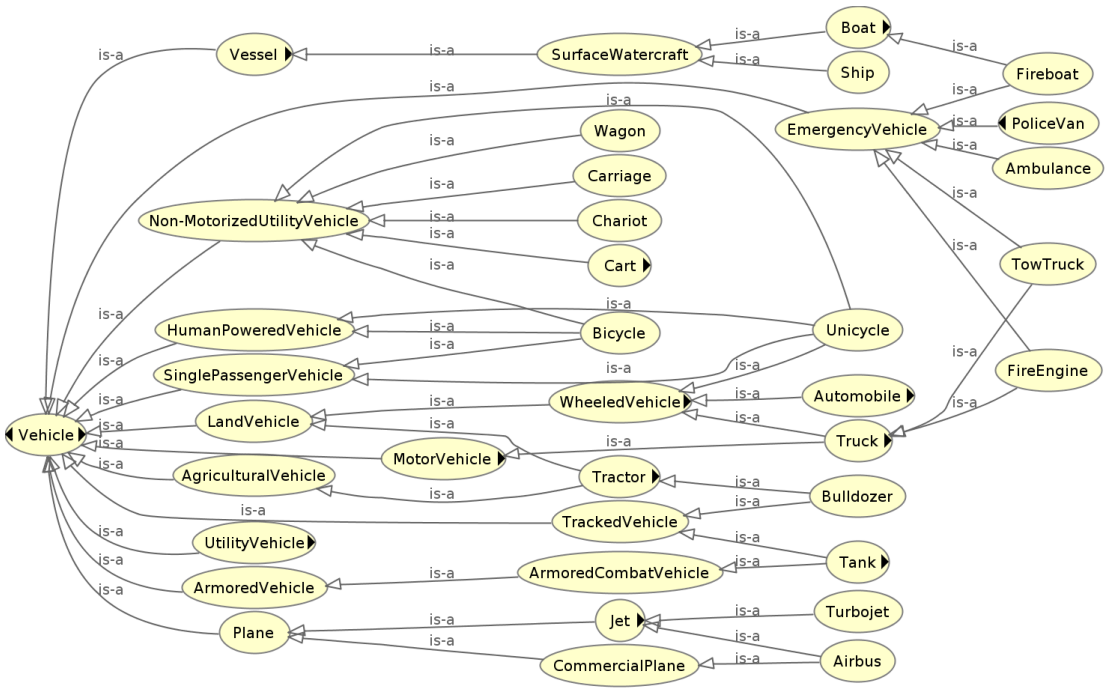


Figure 7.3: $\mathcal{CT}$ of the Vehicle Ontology. Some vehicles, for example all subclasses of Boat and Automobile, have been omitted for space reasons.

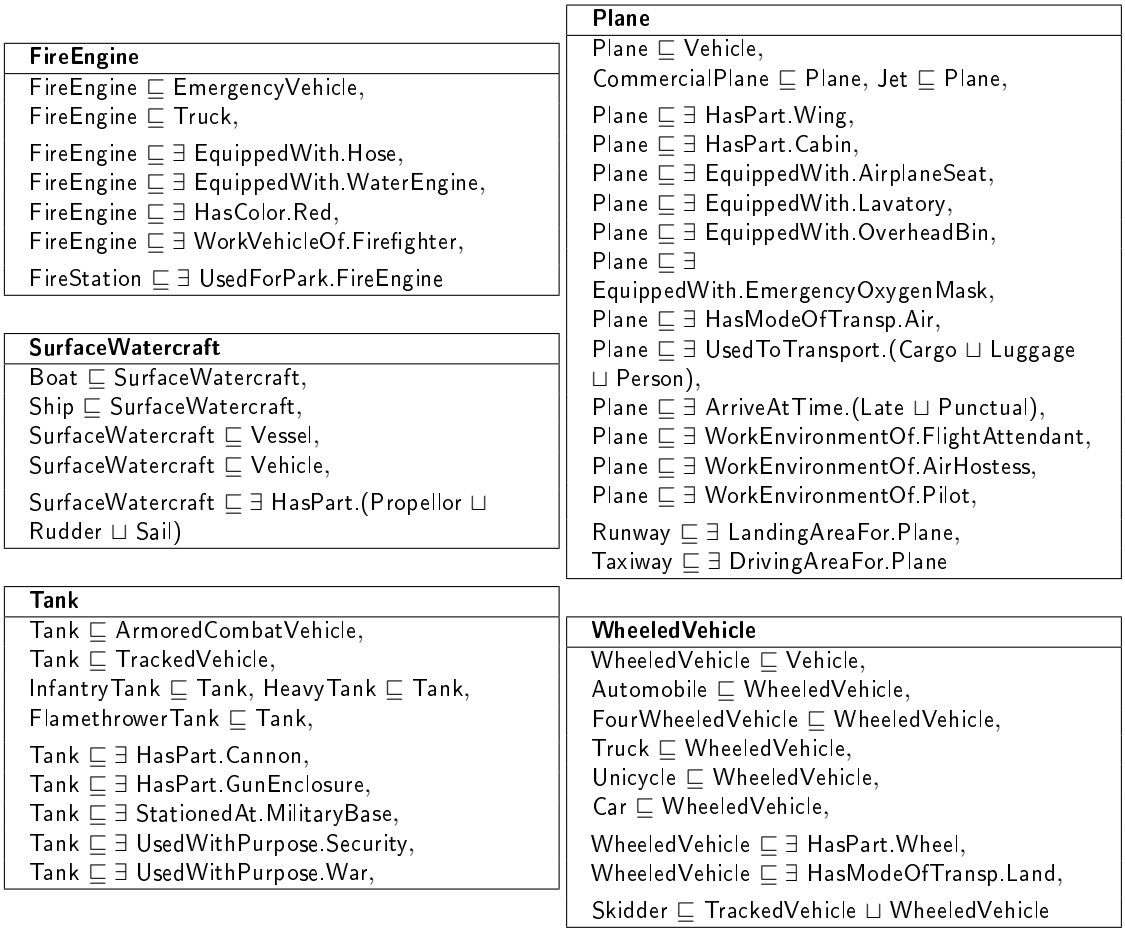


Figure 7.4: Selected axioms from the Vehicle ontology.

Comments:
The suggestions revealed some interesting concepts that describe subclasses of vehicles, such as UtilityVehicle and HumanPoweredVehicle. As a downside, the differentiation between some concepts was not very clear, for example between Car and Automobile.
The list of subconcepts of VehicleParts that was automatically created through the universal HasPart axiom is exhaustive and contains >30 concepts and further substructures.
Defining modes of transportation and correctly assigning the vehicles was challenging.
Interestingly a lot of different types of tanks were suggested, for example the CM11 and the CM12 battle tank, while no specific types of cars were suggested. This might be due to a larger amount of high-weighted assertions that contain car. Consequentially the weight threshold function filters out suggestions that stem from assertions with comparably lower weight.
A revised version of the ontology could extend the current structure and incorporate a strict partition into one-, two-, three- and four-wheeled vehicles together with a concept for wheel-less forms of transportation.

7.3 Fruit

The Fruit ontology was the first ontology that we built with *CN2TopicOnto*. The idea for this thesis project was initiated by the desire to have a small, manageable ontology about different genera of fruit to train a human-like robot. Size-wise the final ontology is as extensive as the Vehicle ontology, with 312 concepts names and 568 TBox axioms. In contrast to the two previously presented topic ontologies, the Fruit ontology contains no individuals.

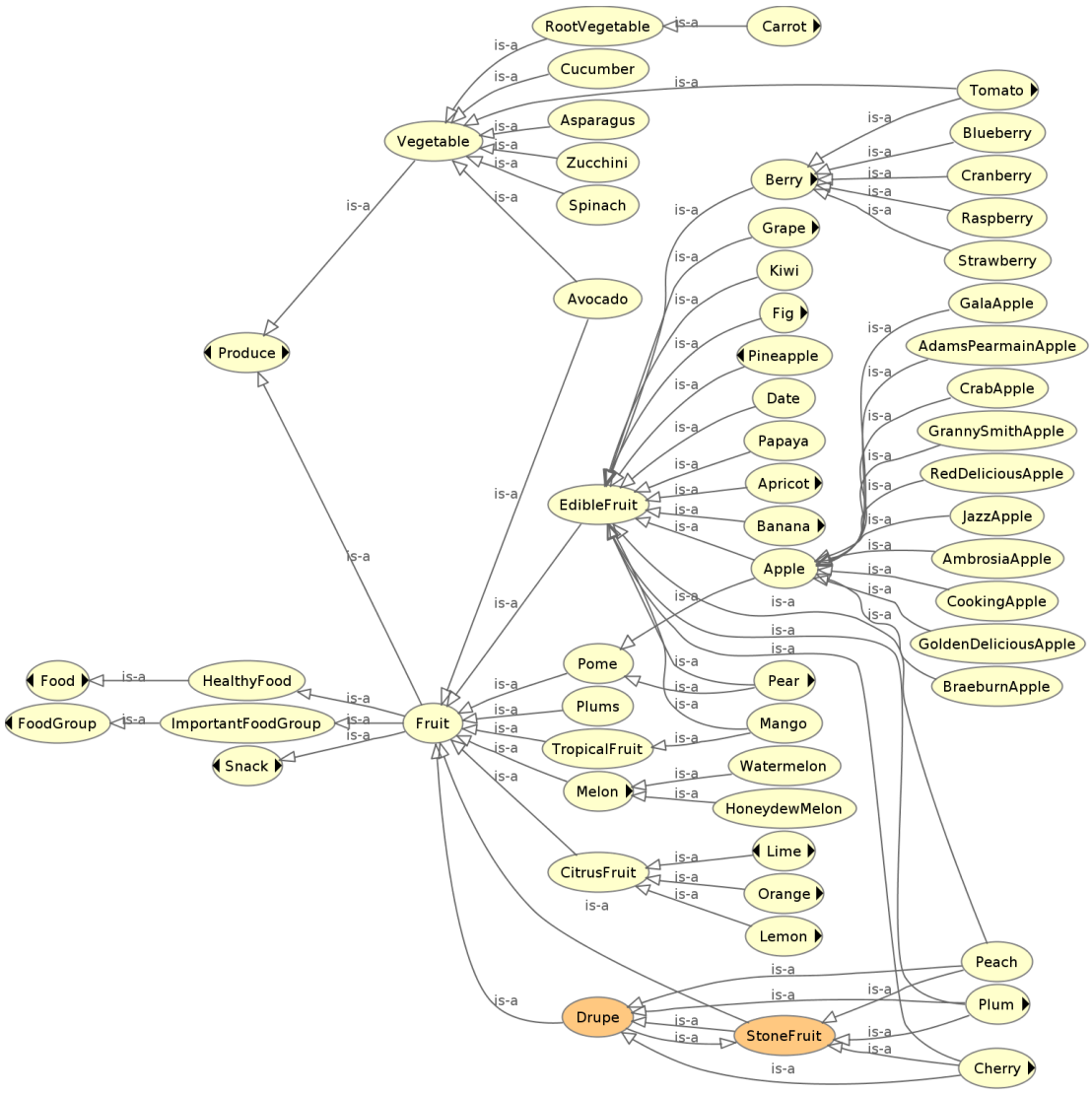


Figure 7.5: $\mathcal{CT}$ of the Fruit ontology. Subclasses of Apple and Berry are displayed with respective subclasses for exemplary purposes. Further subclasses for other fruits, such as Grape or Banana exist in the complete version but were omitted for space reasons.

CitrusFruit

CitrusFruit ⊑ Fruit, Lemon ⊑ CitrusFruit, Lime ⊑ CitrusFruit, Orange ⊑ CitrusFruit,

CitrusFruit ⊑ ∃ HasTaste.Sour,
CitrusFruit ⊑ ∃ HighIn.CitricAcid,
CitrusFruit ⊑ ∃ TypGrownInLoc.SouthAfrica

Lime

Lime ⊑ CitrusFruit, Lime ⊑ SolidFood,
KeyLime ⊑ Lime,

Lime ⊑ ∃ HasColor.(Yellow ⊔ Green),
Lime ⊑ ∃ HasColor.YellowGreen,

Caipirinha ⊑ ∃ Contain.Lime,
Daiquiri ⊑ ∃ Contain.Lime,
Gimlet ⊑ ∃ Contain.Lime

Strawberry

Strawberry ⊑ Berry,

Strawberry ⊑ ∃ HasColor.Red,
Strawberry ⊑ ∃ HasTaste.(Sour ⊔ Sweet),
Strawberry ⊑ ∃ HighIn.VitaminC

Guacamole

Guacamole ⊑ Food,
Guacamole ⊑ Dip,

Guacamole ⊑ ∃ MadeFrom.Avocado,
Guacamole ⊑ ∃ TypicallyContain.Tomato,

Apple

Apple ⊑ EdibleFruit, Apple ⊑ Pome,
AdamsPearmainApple ⊑ Apple,
AmbrosiaApple ⊑ Apple, CrabApple ⊑ Apple,
BraeburnApple ⊑ Apple, CookingApple ⊑ Apple,
GalaApple ⊑ Apple, JazzApple ⊑ Apple,
GoldenDeliciousApple ⊑ Apple,
GrannySmithApple ⊑ Apple,
RedDeliciousApple ⊑ Apple,

Apple ⊑ ∃ HasPart.Peel,
Apple ⊑ ∃ HasPart.AppleCore,
Apple ⊑ ∃ HasPart.Stem,
Apple ⊑ ∃ GrowOnTree.AppleTree
Apple ⊑ ∃ HasShape.Round
Apple ⊑ ∃ TypicallyHasColor.Red
Apple ⊑ ∃ HasColor.(Green ⊔ Red ⊔ Yellow),

AppleOrchard ⊑ ∃ PlaceToGrow.Apple,
ApplePie ⊑ ∃ Contain.Apple,
ToffeeApple ⊑ ∃ MadeFrom.Apple,

PineappleCake

PineappleCake ⊑ Cake,
PineappleCake ⊑ Food,
PineappleCake ⊑ FruitBasedFood,
PineappleCake ⊑ SweetFood,

PineappleCake ⊑ ∃ Contain.Pineapple,

Figure 7.6: Selected axioms from the Fruit ontology.

Contents:
different types of fruit, some vegetables, spices and herbs, basic food (bread, egg, meat etc.), fruit- and vegetable-based foods (guacamole, pineapple cake etc.), drinks (alcoholic drinks, juices), storage containers (bowl, can, jar etc.), food components (sugars, vitamins, fiber etc.), food states (fresh, preserved), locations (market, supermarket, etc.), geographical locations, tastes (salty, sour, spicy etc.), shapes, colors, states (ripeness, physical states, healthiness),

List of role names:
BoughtAt, ConsideredBestWhen, Contain, DependOn, FoundAtSection, GrowOnTree, HasColor, HasFoodState, HasPart, HasPhysicalState, HasShape, HasSize, HasStateOfHealth, HasStateOfRipeness, HasSubdivision, HasTaste, HasTexture, HasTopping, HighIn, Induce, KnownToEat, MadeFrom, MayBeContainedIn, MayContain, NotContentOf, PartOf, PlaceToGrow, Sell, Stores, TypicalFoodOfAnimal, TypicallyConsidered, TypicallyContain, TypicallyGrownInLocation, TypicallyHasColor, TypicallyStoredIn

Comments:
The Fruit ontology grew very rapidly. It was easy to interactively add other foods and engaging to explore further suggestions. Interesting contents were the different types of drinks and (mostly sweet) foods that were suggested because they have some fruit as a key ingredient.
For some things, such as Avocado and Strawberry, the categorization was not distinct, due to the fact that it is different from a biological and a layperson perspective. An avocado is sometimes considered a fruit and other times a vegetable. Similarly tomatoes can be classified as fruits or as vegetables.
There are two synonymous concept names, induced by the axiom StoneFruit $\equiv$ Drupe. Optionally one of them could be chosen to represent the corresponding concept.
Funny concepts that were suggested are Monkey for Banana and Popeye for Spinach. It might be entertaining to explore other iconic people or animals that are associated with certain foods.

7.4 Natural Disaster

The Natural Disaster ontology is a rather small topic ontology whose main purpose was to test how well our knowledge extraction method works for limited topic concepts. It has 89 concept names and 150 TBox axioms.

Contents:
events that are categorized as natural disasters, meteorological phenomena, measurement instruments and scales (seismograph, Beaufort scale, Richter scale), consequences (death, destruction etc.), geographic locations,

Figure 7.7: $\mathcal{CT}$ of the Natural Disaster ontology. The graph displays the complete central taxonomy.

Hurricane

Hurricane ⊑ Cyclone,

Hurricane ⊑ ∃ OccurOverOcean.(AtlanticOcean ⊔ PacificOcean),

GulfOfMexico ⊑ ∃ TypGeogLocOf.Hurricane
Mexico ⊑ ∃ TypGeoLocOf.Hurricane
UnitedStates ⊑ ∃ TypGeoLocOf.Hurricane
Utah ⊑ ∃ TypGeoLocOf.Hurricane
WestVirginia ⊑ ∃ TypGeoLocOf.Hurricane

Storm

Storm ⊑ MeteorologicalEvent,
Blizzard ⊑ Storm, Cyclone ⊑ Storm,
Hailstorm ⊑ Storm, Rainstorm ⊑ Storm,

Storm ⊑ ∃ HasPart.StormCenter
Storm ⊑ ∃ Cause.CoolAir
Storm ⊑ ∃ CoOccurWith.Rain
Storm ⊑ ∃ CoOccurWith.Thunder
Storm ⊑ ∃ CoOccurWith.Lightning

BeaufortScale ⊑ ∃ UsedToMeasure.Storm

Famine

Famine ⊑ Disaster,

Famine ⊑ ∃ HasSpreadRate.SpreadRate,

Starvation ⊑ ∃ CoOccurWith.Famine,
Malnutrition ⊑ ∃ CoOccurWith.Famine

Earthquake

Earthquake ⊑ GeographicalPhenomenon,
Earthquake ⊑ NaturalDisaster,
DeepFocusEarthquake ⊑ Earthquake,
InterplateEarthquake ⊑ Earthquake,
IntraplateEarthquake ⊑ Earthquake,
MajorEarthquake ⊑ Earthquake,
StrongEarthquake ⊑ Earthquake,
Seaquake ⊑ Earthquake, Tremor ⊑ Earthquake,

Earthquake ⊑ ∃ Cause.Aftershock,
Earthquake ⊑ ∃ Cause.EarthTremor,
Earthquake ⊑ ∃ Cause.MainShock,
Earthquake ⊑ ∃ Affect.GeographicalLocation,

RichterScale ⊑ ∃ UsedToMeasure.Earthquake
Seismograph ⊑ ∃ UsedToMeasure.Earthquake

Disaster

ChemicalDisaster ⊑ Disaster,
Ecodisaster ⊑ Disaster, Fire ⊑ Disaster,
Famine ⊑ Disaster, TrainWreck ⊑ Disaster,
ManMadeDisaster ⊑ Disaster,
NaturalDisaster ⊑ Disaster,
NuclearDisaster ⊑ Disaster,

Disaster ⊑ ∃ Cause.Distruction,
Disaster ⊑ ∃ HasDeathToll.Number,
Disaster ⊑ ∃ HasVictim.Person,

Figure 7.8: Selected axioms from the Natural Disaster ontology.

List of role names:
Affect, Cause, CausedBy, CoOccurWith, ComprisedOf, HasDeathToll, HasPart, HasPhase, HasSpreadRate, HasTypicalGeographicLocation, HasVictim, OccurOverOcean, PartOf, ResultOf, TypicalGeographicLocationOf, UsedToMeasure

Comments:
The suggestions were a lot better and more extensive than expected. Although it is arguable what constitutes a natural disaster the suggested concepts were fitting and not too far fetched.
The measurement instruments and scales that were suggested are interesting, but a lot of them are missing to form an exhaustive list.
The four individuals contained in the ontology are katrina (Hurricane), chernobyl (NuclearDisaster), fukushima (NuclearDisaster), and tess (Typhoon). It would be nice to introduce more individuals of well-known earthquakes, hurricanes or tsunamis that occurred in the past decades to further extend the list. You could then define, for example, grave events by using the number of victims and count the amount of people injured or killed during each natural disasters.

CHAPTER 8

Conclusion

In this thesis we have explored a new method that exploits existing knowledge graphs to semi-automatically generate manageable $\mathcal{ALCIO}$ topic ontologies. In the following we summarize our findings and discuss extensions of our method and our implementation that we consider interesting for future work.

- We have developed a stepwise procedure that successively expands the contents of an ontology starting from a single topic concept. To achieve this we have, in Chapter 3, defined the taxonomy as a core part of a topic ontology and introduced the notions of central taxonomy and central concepts as the most relevant atomic concept inclusions and atomic concepts associated with a topic.
  In Chapter 4, we have formalized a universal method to extract appropriate knowledge from a chosen knowledge graph, leveraging connection weights, to produce suggestions for concept names corresponding to the topic. Through dynamic extensions we have presented a technique to immediately introduce further relevant axioms to the TBox in an intuitive way, whenever a concept name is added. The IOQ algorithm for traversing the taxonomy that we developed in Chapter 3 is very simple, but provides a structured way to navigate through the taxonomy that is tailored to the command-line setting we chose for our implementation. We further adapted our construction process to make it possible to interactively discover knowledge about a domain and indefinitely extend the contents of a created ontology.
  A major decision that went into our development process, was the selection of axiom types that we wanted our system to support. We are aware that the current ones only cover a fragment of the expressiveness of $\mathcal{ALCIO}$ and an even smaller part of the capabilities of OWL. We especially opted to include disjointness axioms as they can be used to make intuitive domain knowledge explicit, which is often forgotten for common-sense, and role range restrictions to allow for restrictions on role names that are used to express concept characteristics, such a colors or tastes.

- Before we chose ConceptNet for our implementation of *CN2TopicOnto*, we conducted a lot of preliminary research to explore the contained knowledge as well as possible drawbacks of the methodology used to built it. We have discussed our findings in detail in Chapter 5. With *parts, locations* and *capabilities*, we have subsequently defined sets of ConceptNet relation types that we deemed appropriate to represent an initial subset of the domain knowledge that we were interested in.
  Our results have shown that ConceptNet contains interesting general and commonsense knowledge that, when transformed into suggestions, extends past what we expect classical methods of knowledge elicitation to produce in a similar time frame. This finding is easily explained by the years-long crowdsourcing efforts that went into the construction of ConceptNet. While building sample ontologies, some fun and interesting results showed up, such as "Popeye" when querying for spinach, "pirates" for boat, or "wiggletail" for pig. We consider such knowledge to be especially interesting, as it is universally prevalent among humans, but not generally valid.

  Unfortunately, we discovered that the ConceptNet weight function leaves a lot to be desired. The distribution process is rather obscure and its additive nature skews weights in favor of popular concepts. In consequence, this resulted in a high variety of suggestions for some concepts, which reduced their manageability and made it hard to keep track of selected concept names. With our method to adjust information over hierarchy levels and the support of *Numberbatch* vector embeddings, we have explored two remedies to improve the suggestions. While we consider the outcomes in our few tests an overall improvement, there was still a large difference in the individual results and we recognize that further adaption is needed, especially when it comes to the assessment of universal threshold values.

- To illustrate the usefulness of *CN2TopicOnto*, we have constructed six topic ontologies on the topics *animal, fruit, vehicle, natural disaster, fast food* and *university* that are available for download. In Chapter 7, we have presented selected parts for some of those ontologies, and have shown that it is possible to construct ontologies with 100-600 concept names and 150-1200 TBox axioms using our tool within a couple of hours. The ontology on *natural disaster* shows that our method also works for topic concepts that cover only small domains.
  In consideration of our example ontologies, we conclude that the axioms that we were able to create are suitable to answer the competency questions that we posed in the introduction. We hope that our constructed ontologies are an adequate demonstration of the types of topic ontologies that can be created with *CN2TopicOnto*, and of use to other researchers. In Chapter 6 of our work we have taken the issues that occurred during the construction of our sample ontologies as a basis to discuss frequent ontology engineering problems, and we have provided some guidelines that we hope will assist possible users with the optimal application of our tool. Modeling commonsense knowledge is hard, especially when it comes to modalities that distinguish between necessary, possible and typical knowledge. In our examples we show how modalities can be pushed into the role names, through roles, such as TypicallyFoundAt

and HasPreferredFood. Exploring lesser known DL extensions that introduce typicality operators that can properly handle modalities, is definitely an interesting task to consider for future work in this area.

We hope that our results are a motivation to explore further techniques to help users build ontologies on a chosen topic domain, utilizing the vast amount of unstructured knowledge that is available nowadays. We argue that manageable and understandable ontologies are rare and that a method to quickly construct such ontologies is valuable to produce customized, and customizable ontologies that can be leveraged for further research purposes, such as testing proof-of-concept prototypes or employing query answering techniques.

There are still many challenges to be faced when it comes to finding good models and representations of common-sense everyday knowledge. Exploring the boundaries and capabilities of DLs for modeling common-sense knowledge is an important task, which has been considered in many research efforts. We hope that our method and small implementation can be useful as means to detect common issues and illustrate frequent mistakes.

Future Work There are multiple directions from a research and an implementation perspective that are interesting to explore for future work.

- For the practical implementation a next step would be to extend the supported axioms in *CN2TopicOnto* to a more expressive DL, for example $\mathcal{SHOIQ}$. For this we would need to find a way to support negation and establish a method to isolate respective suggestions. An easier extension would be to allow for more axiom forms that contain all-quantification, as our current version only supports $\forall$ in a very limited setting with HasPart-axioms. Furthermore the axioms contained in our example ontologies have shown that number restrictions and proper role hierarchies would be well suited for our modeling purposes.
- An extension of our algorithm to more expressive DLs raises the question whether a graphical user interface is necessary to maintain good user experience as decisions and axiom configurations grow. We suggest a Protégé plug-in, because Protégé it is a popular ontology engineering tool and its system for user input could be easily adapted and extended for our method. Central concepts could still be proposed one after another using the iteration order queue, but concepts would also be directly selectable through the Protégé class hierarchy. Suggestions could be easily selected from lists by clicking, and restrictions on axioms and selection of axiom types could be realized through ticking boxes and drop-down menus. A GUI would in general make the topic ontology creation process easier and the tool more attractive to use for small experiments or tests.
- From a contextual aspect our method needs to be tested with other knowledge graphs, in order to receive diversified results. From the mentioned knowledge graphs, WebChild

seems especially interesting, because it was recently published and has a lot of fine grained properties, such as hasShape, hasSize and hasTaste, as sub-relations of the hasProperty relation type. Furthermore it claims to be able to distinguish between different word senses, which, if properly applied, gives WebChild a huge advantage over ConceptNet when it comes to generating suggestions.

- As a last point for future work we want to mention the development of possible improvement algorithms for topic ontologies that can be applied to a "finished" ontology. We propose two different approaches:

  1. Research axiom patters that can be (semi-)automatically applied to add new axioms based on existing ones.
     The *property closure pattern*

     $$A \sqsubseteq \exists R.B_1, A \sqsubseteq \exists R.B_2 \ldots A \sqsubseteq \exists R.B_n \Rightarrow A \sqsubseteq \forall R.(B_1 \sqcup B_2 \sqcup \ldots \sqcup B_n)$$

     and the *covering axiom pattern*

     $$A_1 \sqsubseteq B, A_2 \sqsubseteq B, \ldots, A_n \sqsubseteq B \Rightarrow B \sqsubseteq (A_1 \sqcup A_2 \sqcup \ldots \sqcup A_n)$$

     are examples of such patterns for $A_i, B_i \in N_C$ and $R \in N_R$. Together with user feedback both of those patterns could be used to raise awareness of missing subconcepts that should be included in the ontology.
  2. Exploit prevalent pre- and suffixes in (compound) concept names to generate generalizing and specializing axioms with the help of user feedback. Rules could roughly look as follows:

     If $A_1 \sqsubseteq A_2$ and $A_1 \sqsubseteq \exists R.B$ and A_1 and A_2 share a suffix, then suggest a new axiom $A_2 \sqsubseteq \exists R.C$, where C is entered by the user, or based on the prefix of A_1.

     To illustrate with an example, if Bookshelf $\sqsubseteq$ Shelf and Bookshelf $\sqsubseteq \exists$ Store.Book holds, then suggest to introduce a more general axiom Shelf $\sqsubseteq \exists$ Store.C, where C is entered by the user, for instance as $C =$ Item. This is then a generalization from Bookshelf to Shelf. Another similar axiom would be Kitchenshelf $\sqsubseteq \exists$ Store.Kitchen(items). Detecting such axiom patterns could be especially useful, if multiple semantically similar concept names exist (Container, Box, Can, Bowl), but only few of them are used within a certain axiom form.
     Another rule to generate new axioms could be:

     If $A_1 \sqsubseteq A_2$ and $A_1 \sqsubseteq \exists R.B_1$ and $A_2 \sqsubseteq \exists R.B_2$, then suggest a new axiom $B_1 \sqsubseteq B_2$.

     As an example, if Automobile $\sqsubseteq \exists$ HasEngine.Engine, Toyota $\sqsubseteq$ Automobile and Toyota $\sqsubseteq \exists$ HasEngine.InternalCombustionEngine, then InternalCombustionEngine $\sqsubseteq$ Engine. This added axiom is again encouraged by the fact that InternalCombustionEngine and Engine share the same suffix.
     In general we consider the research area of improving small ontologies on commonsense domains to be very interesting and want to keep it in mind it for future work.

List of Figures

List of Tables

List of Algorithms

Bibliography

[BDRR11] Christian Brel, Anne-Marie Dery-Pinna, Philippe Renevier-Gonin, and Michel Riveill. Ontocompo: A tool to enhance application composition. In *Human-Computer Interaction - INTERACT 2011 - 13th IFIP TC 13 International Conference, Lisbon, Portugal, September 5-9, 2011, Proceedings, Part IV*, pages 588–591, 2011.

[BHLS17] Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

[BN03] Franz Baader and Werner Nutt. Basic description logics. In *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95. 2003.

[CFG06] Óscar Corcho, Mariano Fernández-López, and Asunción Gómez-Pérez. Ontological engineering: Principles, methods, tools and languages. In *Ontologies for Software Engineering and Software Technology*, pages 1–48. 2006.

[CRCB15] Luigi Di Caro, Alice Ruggeri, Loredana Cupi, and Guido Boella. Commonsense knowledge for natural language understanding: Experiments in unsupervised and supervised settings. In *AI*IA 2015, Advances in Artificial Intelligence - XIVth International Conference of the Italian Association for Artificial Intelligence, Ferrara, Italy, September 23-25, 2015, Proceedings*, pages 233–245, 2015.

[CV05] Philipp Cimiano and Johanna Völker. Text2onto. In *Natural Language Processing and Information Systems, 10th International Conference on Applications of Natural Language to Information Systems, NLDB 2005, Alicante, Spain, June 15-17, 2005, Proceedings*, pages 227–238, 2005.

[Dom98] J. Domingue. Tadzebao and webonto: Discussing, browsing, and editing ontologies on the web. volume 4 of *KM*, Banff, Canada, 1998. 11th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'98).

[Don06] Kevin Donnelly. Snomed-ct: The advanced terminology and coding system for ehealth. *Studies in health technology and informatics*, 121:279, 2006.

[EW16] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. In *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16) co-located with the 12th International Conference on Semantic Systems (SEMANTiCS 2016), Leipzig, Germany, September 12-15, 2016.*, 2016.

[FBMR18] Michael Färber, Frederic Bartscherer, Carsten Menne, and Achim Rettinger. Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago. *Semantic Web*, 9(1):77–129, 2018.

[FdE08] Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. DL-FOIL concept learning in description logics. In *Inductive Logic Programming, 18th International Conference, ILP 2008, Prague, Czech Republic, September 10-12, 2008, Proceedings*, pages 107–121, 2008.

[FEMR15] Michael Färber, Basil Ell, Carsten Menne, and Achim Rettinger. A comparative survey of dbpedia, freebase, opencyc, wikidata, and yago. *Semantic Web Journal*, 1:1–5, 2015.

[FGM07] Blaz Fortuna, Marko Grobelnik, and Dunja Mladenic. Ontogen: Semi-automatic ontology editor. In *Human Interface and the Management of Information. Interacting in Information Environments, Symposium on Human Interface 2007, Held as Part of HCI International 2007, Beijing, China, July 22-27, 2007, Proceedings, Part II*, pages 309–318, 2007.

[FL99] Mariano Fernández-López. Overview of methodologies for building ontologies. 1999.

[FLGPJ97] Mariano Fernández-López, Asunción Gómez-Pérez, and Natalia Juristo. Methontology: from ontological art towards ontological engineering. 1997.

[FV11] Daniel Fleischhacker and Johanna Völker. Inductive learning of disjointness axioms. In *On the Move to Meaningful Internet Systems: OTM 2011 - Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2011, Hersonissos, Crete, Greece, October 17-21, 2011, Proceedings, Part II*, pages 680–697, 2011.

[GF95] Michael Grüninger and Mark S Fox. Methodology for the design and evaluation of ontologies. 1995.

[GPH05] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: a benchmark for owl knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.

[Gro12] W3C OWL Working Group. Owl 2 web ontology language: Document overview. W3C Recommendation, December 11 2012.

[HD03] Ann M. Hickey and Alan M. Davis. Elicitation technique selection: How do experts do it? In *11th IEEE International Conference on Requirements Engineering (RE 2003), 8-12 September 2003, Monterey Bay, CA, USA.*, page 169, 2003.

[HSA+10] Catherine Havasi, Robert Speer, Kenneth C. Arnold, Henry Lieberman, Jason B. Alonso, and Jesse Moeller. Open mind common sense: Crowdsourcing for common sense. In *Collaboratively-Built Knowledge Sources and Artificial Intelligence, Papers from the 2010 AAAI Workshop, Atlanta, Georgia, USA, July 11, 2010*, 2010.

[HSG15] Robert Hoehndorf, Paul N. Schofield, and Georgios V. Gkoutos. The role of ontologies in biological and biomedical research: a functional perspective. *Briefings in Bioinformatics*, 16(6):1069–1080, 2015.

[JG11] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. Logmap: Logic-based and scalable ontology matching. In *The Semantic Web - ISWC 2011 - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I*, pages 273–288, 2011.

[KLOW17] Boris Konev, Carsten Lutz, Ana Ozaki, and Frank Wolter. Exact learning of lightweight description logic ontologies. *Journal of Machine Learning Research*, 18:201:1–201:63, 2017.

[Lam17] Jean-Baptiste Lamy. Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies. *Artificial Intelligence in Medicine*, 80:11–28, 2017.

[LH10] Jens Lehmann and Pascal Hitzler. Concept learning in description logics using refinement operators. *Machine Learning*, 78(1-2):203–250, 2010.

[MCWD06] Cynthia Matuszek, John Cabral, Michael J. Witbrock, and John DeOliveira. An introduction to the syntax and content of cyc. In *Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering, Papers from the 2006 AAAI Spring Symposium, Technical Report SS-06-05, Stanford, California, USA, March 27-29, 2006*, pages 44–49, 2006.

[Mus15] M.A. Musen. The protégé project: A look back and a look forward. AI Matters. Association of Computing Machinery Specific Interest Group in Artificial Intelligence, 1(4), June 2015. DOI: 10.1145/2557001.25757003.

[NM01] Natalya F. Noy and Deborah L. Mcguinness. Ontology development 101: A guide to creating your first ontology. Technical report, 2001.

[OOP] Oops! ontology pitfall scanner! catalogue of common pitfalls. `http://oops.linkeddata.es/catalogue.jsp`. Accessed:2018-08-16.

[Pau17] Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, 8(3):489–508, 2017.

[PGS14] María Poveda-Villalón, Asunción Gómez-Pérez, and Mari Carmen Suárez-Figueroa. Oops! (ontology pitfall scanner!): An on-line tool for ontology evaluation. *Int. J. Semantic Web Inf. Syst.*, 10(2):7–34, 2014.

[PR16] Alina Petrova and Sebastian Rudolph. Web-mining defeasible knowledge from concessional statements. In *Graph-Based Representation and Reasoning - 22nd International Conference on Conceptual Structures, ICCS 2016, Annecy, France, July 5-7, 2016, Proceedings*, pages 191–203, 2016.

[PSG12] María Poveda-Villalón, Mari Carmen Suárez-Figueroa, and Asunción Gómez-Pérez. Validating ontologies with oops! In *Knowledge Engineering and Knowledge Management - 18th International Conference, EKAW 2012, Galway City, Ireland, October 8-12, 2012. Proceedings*, pages 267–281, 2012.

[Rec] Alan L. Rector. Foundations of the semantic web: Ontology engineering (lecture slides). `http://www.cs.man.ac.uk/~rector/modules/CS646/Lecture-Handouts/Lect-3-problems-and-patterns-2007.ppt.pdf`. Accessed: 2018-10-02.

[Rec03] Alan L. Rector. Modularisation of domain ontologies implemented in description logics and related formalisms including owl. In *Proceedings of the 2nd International Conference on Knowledge Capture (K-CAP 2003), October 23-25, 2003, Sanibel Island, FL, USA*, pages 121–128, 2003.

[RGROB08] Lila Rao-Graham, Han Reichgelt, and Kweku-Muata Osei-Bryson. Knowledge elicitation techniques for deriving competency questions for ontologies. In *ICEIS 2008 - Proceedings of the 10th International Conference on Enterprise Information Systems*, volume 2, pages 105–110, 01 2008.

[RPM+14] Yuan Ren, Artemis Parvizi, Chris Mellish, Jeff Z. Pan, Kees van Deemter, and Robert Stevens. Towards competency question-driven ontology authoring. In *The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings*, pages 752–767, 2014.

[Sat] Uli Sattler. Role modelling (lecture slides). `http://studentnet.cs.manchester.ac.uk/pgt/2015/COMP62342/slides/Week4-RoleModelling.pdf`. Accessed: 2018-16-08.

[SB89] Nigel Shadbolt and A Mike Burton. The empirical study of knowledge elicitation techniques. *ACM SIGART Bulletin*, (108):15–18, 1989.

[Sch05] Stefan Schlobach. Debugging and semantic clarification by pinpointing. In *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings*, pages 226–240, 2005.

[SCH17] Robert Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 4444–4451, 2017.

[SEA+02] York Sure, Michael Erdmann, Jürgen Angele, Steffen Staab, Rudi Studer, and Dirk Wenke. Ontoedit: Collaborative ontology development for the semantic web. In *The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, pages 221–235, 2002.

[SH12] Robert Speer and Catherine Havasi. Representing general relational knowledge in conceptnet 5. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012*, pages 3679–3686, 2012.

[SHS10] Robert Speer, Catherine Havasi, and Harshit Surana. Using verbosity: Common sense data from games with a purpose. In *Proceedings of the Twenty-Third International Florida Artificial Intelligence Research Society Conference, May 19-21, 2010, Daytona Beach, Florida*, 2010.

[SKW07] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 697–706, 2007.

[SL17] Robert Speer and Joanna Lowry-Duda. Conceptnet at semeval-2017 task 2: Extending word embeddings with multilingual relational knowledge. In *Proceedings of the 11th International Workshop on Semantic Evaluation, SemEval@ACL 2017, Vancouver, Canada, August 3-4, 2017*, pages 85–89, 2017.

[TdMW17] Niket Tandon, Gerard de Melo, and Gerhard Weikum. Webchild 2.0 : Fine-grained commonsense knowledge distillation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, System Demonstrations*, pages 115–120, 2017.

[UG96] Mike Uschold and Michael Gruninger. Ontologies: principles, methods and applications. *Knowledge Eng. Review*, 11(2):93–136, 1996.

[WCH87] Morton E. Winston, Roger Chaffin, and Douglas Herrmann. A taxonomy of part-whole relations. *Cognitive Science*, 11(4):417–444, 1987.

[weba] Conceptnet documentation. `https://github.com/commonsense/conceptnet5/wiki`. Accessed: 2016-10.

[webb] The web graph database: What are the differences between a vocabulary, a taxonomy, a thesaurus, an ontology, and a meta-model? `http://infogrid.org/trac/wiki/Reference/PidcockArticle`. Accessed: 2018-06-11.

[WSSR06] Yimin Wang, York Sure, Robert Stevens, and Alan L. Rector. Knowledge elicitation plug-in for protégé: Card sorting and laddering. In *The Semantic Web - ASWC 2006, First Asian Semantic Web Conference, Beijing, China, September 3-7, 2006, Proceedings*, pages 552–565, 2006.