Fachbereich 3: Mathematik/Informatik

Master Thesis

Ontology Partitioning using $\mathcal{E}$-Connections: Revisited

Sascha Jongebloed

Erstgutachter: Prof. Dr. Thomas Schneider
Zweitgutachter: Dr. Serge Autexier

Wednesday 10th June, 2020

Revision	Date	Description
1.0	25.05.20	Final version for submission
1.1	09.06.20	Fixed small errors in indexing, style and spelling
1.2	10.06.20	Added acknowledgements

Abstract.

Modularity of ontologies has received increased attention in the past. For ontologies, methods for extracting a module and also methods for decomposing an ontology into several parts were developed. So far, many and also successful module extraction methods have been investigated, whereas only a few and often less successful decomposition methods have been investigated. In this thesis, we investigate the decomposition method "Partitioning using $\mathcal{E}$-connections", which was first introduced by Cuenca Grau et al. (2005, 2006). During our investigation of the existing notation, algorithm, and implementation, we developed a simplified notation and extended it to the full OWL 2, with the exception of the universal role. In this thesis, we develop a new, conceptually much simpler algorithm that runs in a linear rather than the original quadratic runtime. The algorithm is deterministic and allows simplified proofs of correctness and maximality. To test partitioning using $\mathcal{E}$-connections, we implement the algorithm in Java and evaluate it on a large and diverse established corpus of biomedical ontologies. It turns out that the algorithm is fast in practice, but for "usable" decompositions, one needs heuristics in most cases, which we develop and investigate in this thesis. Furthermore, we investigate the transfer of partitioning using $\mathcal{E}$-connections to other fragments of First-Order Logic.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my main supervisor Thomas Schneider. His academic and personal support during the writing of this thesis was outstanding. Thank you for the excellent lectures I attended over the course of my studies.

I also want to thank Mihai Pomarlan for his ideas and suggestions. Our discussions together with Thomas and Mihai Codescu helped me a lot in my work. I gratefully acknowledge Sergej Autexier for reviewing this thesis and Yongsheng Gao for taking the time for an interview and for providing me feedback afterwards. Furthermore, thanks to Uli Sattler and Haorou Zhao for their feedback and comment on $\mathcal{E}$-connections and modularity.

Great appreciation goes to my parents and brothers for their support in many ways. Their continuous understanding, encouragement and love were essential for the success of my studies and this thesis.

Sincere thanks to all my friends, especially Rafael Miranda and Lukas Samel for their continuous kindness and moral support during my studies. Among my friends, I would also like to particularly thank Robin Nolte for the academic dialogue and discussions about our theses. I cannot overstate how thankful I am for all the friends I met during my studies in Bremen; for their friendship and memories. I could not have finished my studies without you.

Table of Contents

List of Figures

List of Tables

Chapter 1

Motivation

Modularity has become an essential paradigm in software development over the past few decades. This paradigm has received increased attention for the development of ontologies in the past as witnessed by the WoMO[1]/WoMoCoE[2] workshop series and a monograph on modular ontologies (Stuckenschmidt et al., 2009).

There are several advantages to modularity in the development of ontologies: modular ontologies are easier to maintain, to comprehend, and to reason over. The extraction of modules can also allow the reuse of a specific part of an ontology if only a specific part of the ontology is of interest.

Decomposition is the task of splitting ontologies into several parts and therefore making them modular. Decomposition can also be used to help to extract a single module (Klinov et al., 2012). Several decomposition methods have been developed: Structure-based partitioning (Stuckenschmidt and Schlicht, 2009), partitionings based on $\mathcal{E}$-connections (Cuenca Grau et al., 2005, 2006) and atomic decomposition (Del Vescovo et al., 2011). Only the latter two provide strong logical guarantees important for the scenarios mentioned above.

In this thesis, we will cover partitionings based on $\mathcal{E}$-connections (from now on: $\mathcal{E}$-partitions). This procedure was developed by Cuenca Grau et al. (2005, 2006). An $\mathcal{E}$-partition is a partitioning of the axioms that complies with the $\mathcal{E}$-connections framework. The axioms are partitioned into several subsets that are connected by edges that represent "semantic links".

The usage of the $\mathcal{E}$-connections framework gives us strong logical guarantees, e.g. (certain combinations of) the components are encapsulating modules of the input ontology (Cuenca Grau et al., 2006). $\mathcal{E}$-connections were first defined for *abstract description systems (ADSs)* (Kutz et al., 2004). ADSs generalize DLs, modal logics, and further formalisms (Baader et al., 2002).

$\mathcal{E}$-connections for AD are a combination of heterogeneous logical theories via semantic links established by a designated set of relations, called *link relations*. Their models contain sets of pairwise disjoint components which are each locally interpreted. Link relations are interpreted as relations between the different components.

Hitherto $\mathcal{E}$-partitions have been only defined for $\mathcal{SHOIQ}(\mathcal{D})$. The procedure takes an ontology and tries to find the finest possible $\mathcal{E}$-connection, which is equivalent to the

[1] `https://iaoa.org/womo/history.html`
[2] `https://womocoe19.fbk.eu/`

input ontology (under the $\mathcal{E}$-connection semantics), by finding link relations among the roles in the ontology according to the rules of $\mathcal{E}$-connection.

Cuenca Grau (2005) introduced an efficient algorithm for finding the finest possible $\mathcal{E}$-connection, which is equivalent to the input ontology (under the $\mathcal{E}$-connection semantics). The algorithm builds a directed graph in which each vertex represents a component of the partitioning. The algorithm goes through all subconcepts and assigns either an existing or "fresh" component. It assigns an existing component whenever necessary according to the definitions of $\mathcal{E}$-connection. A "fresh" component is assigned if some freshly identified link relation witnesses this. The assignment of concepts then induces a partition of the ontology's axioms.

An important notion for this procedure is *safety* (Cuenca Grau, 2005; Cuenca Grau et al., 2005, 2006), which is needed for an ontology to be equivalent to its $\mathcal{E}$-partition under $\mathcal{E}$-connection semantics. It coincides with *domain-independence (DI)* known from first-order logic and database theory (Abiteboul et al., 1995). Although DI is undecidable for FOL (Vardi, 1981), DI for $\mathcal{SHOIQ}(\mathcal{D})$ is decidable via locality (Cuenca Grau et al., 2006).

Initial experiments, using an implementation for the discontinued ontology editor Swoop [3], showed some useful $\mathcal{E}$-partitions, while some of those $\mathcal{E}$-partitions revealed modeling deficiencies (Cuenca Grau et al., 2006). The results of their experiment suggest several modeling patterns that are problematic for the use of $\mathcal{E}$-partitions. One of them is the use of a top-level concept. Another problematic modeling pattern is the extensive use of disjoint concepts and equivalent or distinct individuals. These patterns, together with the definition of $\mathcal{E}$-connection, can result in a coarse $\mathcal{E}$-partition, by forcing many concepts or individuals into one component. These problems could be one reason for the limited success of $\mathcal{E}$-partitions. Other reasons could be the incomplete and incorrect implementation and the non-deterministic behavior of the algorithm.

In this thesis, we want to evaluate and examine partitioning using $\mathcal{E}$-connections. The goal is to find out whether the problems mentioned above are due to $\mathcal{E}$-partitions or the weaknesses of the existing algorithm and its implementation. Another goal was to extend the algorithm to full OWL 2 and implement the algorithm. We want to test and evaluate our new implementation to examine the strengths and weaknesses of $\mathcal{E}$-partitions. The last goal was to show that $\mathcal{E}$-partition can be transferred to other fragments of FOL.

For our evaluation we use among others the important ontology repository BioPortal (Noy et al., 2009). BioPortal is a corpus of many, lifelike and diverse ontologies. In this thesis we use a snapshot from March 2017 (Matentzoglu and Parsia, 2017).

While evaluating the existing method by Cuenca Grau et al. (2005, 2006), we found a simpler way to compute $\mathcal{E}$-partitions using undirected graphs and connected components. Instead of quadratic time like the previous algorithm, our proposed algorithm only needs linear time. We transferred $\mathcal{E}$-partitions to OWL 2, with the exception of the universal role. If an ontology contains the universal role u, locality does no longer characterize domain-independence. Therefore equivalence could not be guaranteed given a universal

[3] `https://github.com/ronwalf/swoop`

role Fortunately the universal role seems to play a minor role in modeling [4].

Our proposed algorithm has the following advantages: full support of OWL 2 (with some restrictions ensuring equivalence), a simplified notation of theory, a simpler algorithm, that is deterministic and in linear time. Furthermore, we provide simple and rigorous proofs.

We implemented the proposed algorithm in Java. The implementation and extensive documentation of the implementation can be found on our Github repository[5]. It takes an ontology in the owl file format as input and calculates the partitioning of the ontology. The implementation outputs the partitioning represented as a set of ontologies in OWL format. The implementation can visualize the components connected via link relations in the form of a graph. We implemented a large number of unit tests for all supported logical axioms and special cases. Additionally, we evaluated our implementation over a representative set of ontologies and discussed the results.

Finally, we examined the transfer of $\mathcal{E}$-partition to another fragment of FOL. We transferred the notations, results and their proofs to tuple-generating dependencies (TGDs), a fragment of FOL used in database theory. We have chosen TGDs because of the simple transfer of our theorems and results. The notations are kept general for FOL instead of TGD, so that they can be used to transfer $\mathcal{E}$-partition to other fragments of FOL.

In chapter 2, we will introduce standard description logic notions. Chapter 3 will introduce the necessary notions for $\mathcal{E}$-partition and discuss the difference of our notions to the notions by (Cuenca Grau et al., 2005, 2006). Our new algorithm is described in chapter 4. We give a proof for the correctness and maximality of the results of the algorithm. The implementation and its tests will be described in chapter 6. In chapter 5, we discuss several heuristics that we implemented to improve the results of our algorithm. An evaluation of our implemented algorithm over a representative set of ontologies will be discussed in chapter 7. In chapter 8 we will discuss the transfer of $\mathcal{E}$-partition to other fragments of FOL. As an example, we will transfer the important theorems and the algorithm to TGD's.

Chapters 2, 3 and 4 are an extended and revised version of our joint paper (Jongebloed and Schneider, 2018). Chapter 8 is partially based on unpublished joint work by the same authors.

1.1 Related Work

Numerous module extraction and modularisation approaches are known (Konev et al., 2008; Suntisrivaraporn, 2008a; Cuenca Grau et al., 2009a,b; Del Vescovo et al., 2011; Stuckenschmidt and Schlicht, 2009; Gatens et al., 2013), as well as ontology languages supporting modular development (Bao et al., 2009; Serafini and Tamilin, 2009). Several decomposition methods have been developed.

[4] In the BioPortal corpus from 2017 (Matentzoglu and Parsia, 2017), u occurs in only 322 of all 16.3 million axioms.

[5] Repository of EPartitioner: `https://github.com/sasjonge/epartition`

We can differentiate between two kinds of modularization: *A-priori* and *a-posteriori*. *A-priori* modularity tries to give the user a tool to compose modular ontologies

One example for a-priori modularization is the package-based description logic $\mathcal{SHOIQP}$ by Bao et al. (2009). $\mathcal{SHOIQP}$ is an extension of $\mathcal{SHOIQ}$ with A $\mathcal{SHOIQP}$ ontology consists of multiply modules, so called packages, which can be viewed as $\mathcal{SHOIQ}$ ontologies. An important feature of $\mathcal{SHOIQP}$ is, that it supports interpretations from the point of view of a specific package, unlike OWL imports, which only allow for interpretations from a global point of view, which can lead to unintended inferences if the contextual nature of the assertions is not cared for.

Another example for an a-priori modularization is the Distributed Description Logic framework by Borgida and Serafini (2003); Serafini and Tamilin (2009), which use bridge rules to link between concepts of different ontologies. In contrast to $\mathcal{SHOIQP}$, bridge rules can only express subsumption and equivalence and do not allow concept construction across different modules.

A-posteriori methods provide tools to extract a subset of a given ontology or split a given ontology into several smaller parts.

Parikh introduced *signature decomposition* in the context of belief revision (Parikh, 1999) motivated by the work of Alchourrn et al. (1985). *Update and belief revision* deals with the question that, given a theory and an axiom inconsistent to the theory, if we add the axiom to the theory how does the ontology need to be changed to stay consistent. Naturally, the question of which part of the ontology needs to be changed follows. The *signature decomposition* proposed by Parikh (1999) (which he called T-splitting with T being a theory) is a decomposition of the signature of a logical theory. It can easily be transfered to first order logic without equality by using the countable infinite model property of consistent first order theories (without equality) and induces a partition of the axioms into signature-disjoint parts.

Konev et al. (2010) later refined *signature decomposition* for Description Logics. They especially observed that some roles (e.g. *hasPart*) can behave like a logical symbol and create unwanted dependencies in the signature. Therefore they introduced *Δ-signature decomposition* to allow the exclusion of a set Δ of these symbols for the decomposition.

Several other decomposition approaches have been developed, some of which are solely based on statistical parameters, like structure-based partitioning. An important assumption for structure-based partitioning is, that the dependency between concepts can be derived by the structure of the ontology (Stuckenschmidt and Schlicht, 2009; Amato et al., 2015; Seidenberg and Rector, 2006).

Stuckenschmidt and Schlicht (2009), for example, create a weighted dependency graph by the structure of the ontology. They extract modules by splitting the ontology into parts that are stronger internally connected than externally.

In the context of this thesis our main interest lies in module extraction and decomposition methods which yields strong logical guarantees.

For this purpose Ghilardi et al. (2006) introduced the notion of conserative extension: The union of two ontologies is a conservative extension of one of the parts (w.r.t. a given signature) if all consequences of the union are already consequences of the part.

In this context a module can be defined as a part of an ontology s.t. the ontology is an conservative extension of the module. This would give the logical guarantee, that all consequences of the ontology are consequences of the module.

Unfortunately Lutz et al. (2007) showed that deciding if an union of two ontologies is a conservative extension of one of its parts is undecidable for expressive description logics, like $\mathcal{SROIQ}$ which forms the basis of OWL. Additionally Lutz et al. (2007) showed that deciding conservative extensions defined in model-theoretic ways is undecidable even for less expressive logics.

Several approximations have been introduced to circumvent this negative results. In their paper Grau et al. (2008) discussed the property they call *safety*. Safety defines conditions so that a set symbols can be reused without changing their meaning. Unfortunely Grau et al. (2008) also find that determining safety for the description logic $\mathcal{ALCIQO}$ is undecidable. To find a approximation of safety that is decidable Grau et al. (2008) define `safety classes`, which describe sufficient conditions for safety.

Grau et al. (2008) introduced the safety class *locality*, which is a sufficient condition for safety and can be calculated efficiently. The modules archieved with this method are called *Locality-based modules (LBM)*. Several other approximations were developed: *Reachability-based modules* by Suntisrivaraporn (2008b), *MEX modules* by Konev et al. (2008) and *Datalog-based modules* by Romero et al. (2016).

Instead of extracting one module, decomposition, i.e. extracting all modules could be of interest. A naive approach for a decomposition method is, to create it from LBM's by using every possible seed. Unfortunately there are exponential many seeds in the the size of the ontology (Parsia and Schneider, 2010). Del Vescovo et. al. define in Del Vescovo et al. (2011) *fake* and `genuine` modules. Genuine modules can be seen as modules that cannot be decomposed into smaller modules. Del Vescovo et al. (2011) show that there are only linear many genuine modules that they can be calculated in polynomial time. In contrast to the parts of $\mathcal{E}$-connections, genuine modules can overlap.

Vescovo et al. (2019) define atoms as the maximal subset of axioms that are never separated by any genuine modules. $\mathfrak{A}(\mathcal{O})$ is the set of all atoms of $\mathcal{O}$. An atom $\mathfrak{a}$ is dependent on an atom $\mathfrak{b}$, written $\mathfrak{a} \succeq \mathfrak{b}$ if, if for each module it holds that if $\mathfrak{a}$ is in the module, $\mathfrak{b}$ is also in this module. We call the poset $(\mathfrak{A}(\mathcal{O}), \succ)$ the *atomic decomposition (AD)* of $\mathcal{O}$.

Chapter 2

Preliminaries

We denote concept names with $A, B, \ldots$, complex concepts with $C, D, \ldots$, abstract role names with $r, s, \ldots$, complex abstract roles (role names r or inverses r^-) with $R, S, \ldots$, concrete roles with $P, P', \ldots$ individual names with $a, b, \ldots$, datatypes with $d, d', \ldots$ and axioms with $\alpha, \beta, \ldots$. Concepts and axioms are built according to table 2.1 and table 2.2, where $m \in \mathbb{N}$ and the remaining letters are as explained above.

Additionally, to the axioms of $\mathcal{SROIQ}(\mathcal{D})$, we also handle keys, which are part of OWL. Syntax and semantics of keys are defined in Parsia et al.'s (2008) conference paper. They are expressed using the following syntax:

Definition 1.

$$C \ \mathsf{HasKey}(R_1, \ldots, R_n, P_1, \ldots, P_m)$$

where the R_i are abstract roles (possibly inverse) and the P_i are concrete roles. We use NAMED to denote the set $\{a^{\mathcal{I}} \mid a \in \Sigma_{\mathsf{I}}\}$. The semantic of keys is:

$\mathcal{I} \models C \ \mathsf{HasKey}(R_1, \ldots, R_n, P_1, \ldots, P_m)$

if for all $d, d', e_1, \ldots, e_n \in \Delta^{\mathcal{I}} \cap \mathsf{NAMED}$ and all $x_1, \ldots, x_m \in \Delta^{\mathcal{D}}$ the following holds: if

- $d, d' \in C^{\mathcal{I}}$ and
- $(d, e_i), (d', e_i) \in R_i^{\mathcal{I}}$ for all $i \leq n$ and
- $(d, x_i), (d', x_i) \in P_i^{\mathcal{D}}$ for all $i \leq m$,

then $d = d'$.

The remaining operators $\top, \bot, \sqcup, \exists R, \forall R, \leqslant n\, R$ and axiom types (role transitivity, symmetry, etc.) are "syntactic sugar"; hence we do not treat them explicitly, thus simplifying the presentation without losing generality. We furthermore ignore OWL's global restrictions (regularity, restricted use of non-simple roles) as $\mathcal{E}$-connections and partitionings do not rely on them. An *ontology* is a set of axioms.

Table 2.1: Syntax and semantic of $\mathcal{SROIQ}(\mathcal{D})$ constructors (Krötzsch et al., 2012; Horrocks and Sattler, 2001)

Name	Syntax	Semantics
Individuals:		
individual	a	$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
Roles:		
atomic abstract role	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
inverse abstract role	r^{-}	$\{(x, y) \mid (y, x) \in r^{\mathcal{I}}\}$
atomic concrete role	P	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{D}}$
universal role	u	$u^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Datatypes:		
datatype	d	$d^{\mathcal{I}} \in \Delta^{\mathcal{D}}$
Concepts:		
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
complement	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
datatype complement	$\neg d$	$\Delta^{\mathcal{D}} \setminus d^{\mathcal{I}}$
top concept	$\top$	$\Delta^{\mathcal{I}}$
bottom concept	$\bot$	$\emptyset$
existential restriction	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y.(x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
universal restriction	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}}.(x, y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
at-least restriction.	$\geqslant m\, R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq m\}$
at-most restriction.	$\leqslant m\, R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq m\}$
local reflexivity	$\exists R.\mathsf{Self}$	$\{d \in \Delta^{\mathcal{I}} \mid (d, d) \in r^{\mathcal{I}}\}$
concrete existential restriction	$\exists P.d$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y.(x, y) \in P^{\mathcal{I}} \wedge y \in d^{\mathcal{I}}\}$
concrete universal restriction	$\forall P.d$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}}.(x, y) \in P^{\mathcal{I}} \rightarrow y \in d^{\mathcal{I}}\}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\} \in \Delta^{\mathcal{I}}$

Table 2.2: Syntax and semantic of $\mathcal{SROIQ}(\mathcal{D})$ axioms (Krötzsch et al., 2012; Horrocks and Sattler, 2001)

Name	Syntax	Semantics
ABox:		
concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
abstract role assertion	$R(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
concrete role assertion	$P(a, d)$	$(a^{\mathcal{I}}, d^{\mathcal{I}}) \in P^{\mathcal{I}}$
individual equality	$a \approx b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
individual inequaltity	$a \not\approx b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$
TBox:		
concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
concept equivalence	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
RBox:		
abstract role inclusion	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
abstract role equivalence	$R \equiv S$	$R^{\mathcal{I}} = S^{\mathcal{I}}$
abstract role disjointness	$\mathsf{Disjoint}(R, S)$	$R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$
complex abstr. role inclusion	$R_1 \circ R_2 \sqsubseteq S$	$R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
concrete role inclusion	$P \sqsubseteq P'$	$P^{\mathcal{I}} \subseteq P'^{\mathcal{I}}$

Chapter 3

Foundations

The definitions in this section largely follow those by Cuenca Grau (2005); Cuenca Grau et al. (2005); however, we simplify notation to allow, as we believe, a more concise presentation. The differences and their motivation are explained at the end of this section.

3.1 Syntax and semantic of $\mathcal{E}$-connection

In the following, we work with a fixed *signature*, which is a finite set Σ of concept names, abstract role names, concrete role names and individual names (together: *terms*), i.e., $\Sigma = \Sigma_{\mathsf{C}} \uplus \Sigma_{\mathsf{R}_a} \uplus \Sigma_{\mathsf{R}_c} \uplus \Sigma_{\mathsf{I}}$. Here and in the following, $\uplus$ denotes disjoint union.

An *I-indexing of Σ* is a function ν that assigns to each concept, individual name and concrete role name an index $\nu(A), \nu(a), \nu(P) \in I$ and to each abstract role name a pair $\nu(r) \in I \times I$ of indices.

Indexings are extended to arbitrary concepts and axioms inductively:

Definition 1. Let ν be an I-indexing of Σ. ν is extended to arbitrary roles R and concepts C as follows.

1. If $R = r$, then $\nu(R) = \nu(r)$.
2. If $R = r^-$, and $\nu(r) = (i, j)$, then $\nu(R) = (j, i)$.
3. If $C = \neg D$, then $\nu(C) = \nu(D)$.
4. If $C = D \sqcap E$ and $\nu(D) = \nu(E)$, then $\nu(C) = \nu(D)$.
5. If $C = \geqslant m\, R.D$ and $\nu(R) = (i, j)$ and $\nu(D) = j$, then $\nu(C) = i$.
6. If $C = \geqslant m\, P.dr$, then $\nu(C) = \nu(P)$.
7. If $C = \exists R.\mathsf{Self}$ and $\nu(R) = (i, i)$, then $\nu(C) = i$.
8. If $C = \{a\}$, then $\nu(C) = \nu(a)$.

$\nu(C)$ is defined, if the indexing of C is valid according to 1-8, e.g. $\nu(C)$ with $C = \exists r.D$ is undefined if $\nu(r) = (i, j)$ and $\nu(D) \neq j$. A concept C is called an *i-concept* if $\nu(C)$ is defined and $\nu(C) = i$.

Definition 2. ν is further extended to axioms as follows. An *i-axiom* is of the form

9. $C \sqsubseteq D$ or $C \equiv D$ where $\nu(C) = \nu(D) = i$;
10. $R \sqsubseteq S$, $R \equiv S$, or $\mathsf{Disjoint}(R, S)$ where $\nu(R) = \nu(S) = (i, j)$;
11. $P_1 \sqsubseteq P_2$, $P_1 \equiv P_2$ or $\mathsf{Disjoint}(P_1, P_2)$ where $\nu(P_1) = \nu(P_2)$
12. $R_1 \circ R_2 \sqsubseteq S$ where $\nu(R_1) = (i, j)$, $\nu(R_2) = (j, k)$, and $\nu(S) = (i, k)$;
13. $C(a)$ where $\nu(C) = \nu(a) = i$;
14. $R(a, b)$ where $\nu(a) = i$ and $\nu(R) = (\nu(a), \nu(b))$;
15. $P(a, x)$ where $\nu(P) = \nu(a)$
16. $a \approx b$ or $a \not\approx b$ where $\nu(a) = \nu(b) = i$.
17. $C\ \mathsf{HasKey}(R_1, \ldots, R_n, P_1, \ldots, P_m)$ where $\nu(C) = \nu(P_1) = \cdots = \nu(P_m) = i$ and, for all $j \leq n$, $\nu(R_j) = (i, k_j)$ for some k_j.

The choice of i for the item 12 and 14 are arbitary, e.g. for 14 it could also be "$\nu(b) = i$".

The central notion of an ontology consisting of several parts depends on ν:

Definition 3. Given an I-indexing ν with $\mid I \mid = n$, a *ν-ontology* is a set $\mathbb{O} = \{\mathcal{O}_i \mid i \in I\}$, each of whose *components* $\mathcal{O}_i$ is a nonempty set of i-axioms.

To distinguish (monolithic) ontologies $\mathcal{O}$ from (combined) ν-ontologies $\mathbb{O}$, we sometimes call the former *simple ontologies*. We sometimes say that a concept C, individual a or concrete role P *"lives"* in the component $\mathcal{O}_i$ if $\nu(C) = i$, $\nu(a) = i$ or $\nu(P) = i$. A abstract role R *"lives"* in a component $\mathcal{O}_i$ if $\nu(R) = (i, j)$.

The semantics of ν-ontologies is given by ν-interpretations, which are partitioned according to the indexing of the signature given by ν:

Definition 4. A *ν-interpretation* is an interpretation $(\Delta^{\mathcal{I}}, \Delta^{\mathcal{D}}, \cdot^{\mathcal{I},\nu})$ such that

- $\Delta^{\mathcal{I}} = \biguplus_{i \leq n} \Delta_i^{\mathcal{I}}$ with $\Delta_i \neq \emptyset$ for each *component* Δ_i
- $\Delta^{\mathcal{D}}$ is a nonempty set of the concrete domain, s.t. $\Delta^{\mathcal{I}} \cap \Delta^{\mathcal{D}} \neq \emptyset$
- $A^{\mathcal{I},\nu} \subseteq \Delta_i^{\mathcal{I}}$ for all $A \in \Sigma_{\mathsf{C}}$ with $\nu(A) = i$
- $r^{\mathcal{I},\nu} \subseteq \Delta_i^{\mathcal{I}} \times \Delta_j^{\mathcal{I}}$ for all $r \in \Sigma_{\mathsf{R}_a}$ with $\nu(r) = (i, j)$
- $P^{\mathcal{I}} \subseteq \Delta_i^{\mathcal{I}} \times \Delta^{\mathcal{D}}$ for all $P \in \Sigma_{\mathsf{R}_c}$ with $\nu(P) = i$
- $a^{\mathcal{I},\nu} \in \Delta_i^{\mathcal{I}}$ for all $a \in \Sigma_{\mathsf{I}}$ with $\nu(a) = i$

ν-interpretations are used for two purposes: (a) for defining the semantics of simple ontologies given ν, and (b) for defining the semantics of ν-ontologies, which is the usual semantics of $\mathcal{E}$-connections (Kutz et al., 2004). In case (a), we use the standard DL semantics and the usual satisfaction symbol $\models$. We say that $\mathcal{O}$ has a *ν-model* if there is a ν-interpretation $\mathcal{I} \models \mathcal{O}$. The class of ν-models of $\mathcal{O}$ is contained in the class of its (unrestricted) models. For example, if $\mathcal{O}$ contains $\top \sqsubseteq A$, then all ν-models have a single component; thus, if ν is an I-indexing with $|\ I\ | \geq 2$, then $\mathcal{O}$ has no ν-models but may still be consistent (i.e., have unrestricted models). For purpose (b), the interpretation function is extended ensuring that the extension of arbitrary i-concepts is a subset of $\Delta_i^{\mathcal{I}}$. For this purpose, (only) the case of negation has to differ from the standard semantics.

Definition 5. The interpretation function of a ν-interpretation $\mathcal{I}$ is extended to arbitrary ν-concepts as follows.

- $(\neg C)^{\mathcal{I},\nu} = \Delta_i^{\mathcal{I}} \setminus C^{\mathcal{I},\nu}$ for i-concepts C
- $(C \sqcap D)^{\mathcal{I},\nu} = C^{\mathcal{I},\nu} \cap D^{\mathcal{I},\nu}$
- $(\geqslant m\, R.C)^{\mathcal{I},\nu} = \{d \in \Delta_i^{\mathcal{I}} \mid \#\{e \in \Delta_j^{\mathcal{I}} \mid (d,e) \in R^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I},\nu}\} \geq m\}$ for (i,j)-abstract roles R and j-concepts C
- $(\geqslant m\, P.dr)^{\mathcal{I},\nu} = \{d \in \Delta_i^{\mathcal{I}} \mid \#\{e \in \Delta^{\mathcal{D}} \mid (d,e) \in P^{\mathcal{I}}\ \&\ e \in dr^{\mathcal{D}}\} \geq m\}$ for i-concrete roles P
- $(\exists R.\mathsf{Self})^{\mathcal{I},\nu} = \{d \in \Delta_i^{\mathcal{I}} \mid (d,d) \in R^{\mathcal{I}}\}$ for (i,i)-roles R
- $\{a\}^{\mathcal{I},\nu} = a^{\mathcal{I},\nu}$

Satisfaction of i-axioms in ν-interpretations is defined as follows.

- $\mathcal{I} \models^{\nu} C \sqsubseteq D$ if $C^{\mathcal{I},\nu} \subseteq D^{\mathcal{I},\nu}$
- $\mathcal{I} \models^{\nu} C \equiv D$ if $C^{\mathcal{I},\nu} = D^{\mathcal{I},\nu}$
- $\mathcal{I} \models^{\nu} C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I},\nu}$
- For i-axioms α of all other types (role inclusion, equivalence, disjointness, assertion, individual (in)equality) or HasKey, $\mathcal{I} \models^{\nu} \alpha$ if $\mathcal{I} \models \alpha$.

$\mathcal{I}$ is a *model* of a ν-ontology $\mathbb{O}$, written $\mathcal{I} \models^{\nu} \mathbb{O}$, if $\mathcal{I} \models^{\nu} \alpha$ for all axioms α in $\mathbb{O}$.
$\mathbb{O}$ is *consistent* if it has a model.

3.2 Compatibility and equivalence

The correspondence between simple and ν-ontologies is captured by compatibility and equivalence. The former notion is syntactic and only requires that the components of $\mathbb{O}$ partition $\mathcal{O}$. The latter is semantic and relative to ν-interpretations.

Definition 6. Let $\mathcal{O}$ be an ontology, ν an I-indexing, and $\mathbb{O} = (\mathcal{O}_1, \ldots, \mathcal{O}_n)$ a ν-ontology.

1. $\mathcal{O}$ and $\mathbb{O}$ are *compatible*, written $\mathcal{O} \sim \mathbb{O}$, if $\mathcal{O} = \bigcup_{i \leq n} \mathcal{O}_i$.
2. $\mathcal{O}$ and $\mathbb{O}$ are *equivalent*, written $\mathcal{O} \approx \mathbb{O}$ if, for all ν-interpretations $\mathcal{I}$, it holds that $\mathcal{I} \models \mathcal{O}$ iff $\mathcal{I} \models^{\nu} \mathbb{O}$.

Unsurprisingly, compatibility does not imply equivalence, and neither does the converse hold. The latter is trivial:

Example 7. $\mathcal{O} = \{A \sqsubseteq B\}$ and $\mathbb{O} = \{\neg B \sqsubseteq \neg A\}$, then $\mathcal{O} \approx \mathbb{O}$ but $\mathcal{O} \nsim \mathbb{O}$.

For the other direction, take

Example 8. $\mathcal{O} = \{\neg A \sqsubseteq A',\ B \sqsubseteq B'\}$ and $\mathbb{O} = (\{\neg A \sqsubseteq A'\},\ \{B \sqsubseteq B'\})$. Then $\mathbb{O}$ is well-formed according to Definitions 1, 2 and 4 and $\mathcal{O} \sim \mathbb{O}$, but $\mathcal{O} \not\approx \mathbb{O}$ because $\mathbb{O}$ has ν-models $\mathcal{I}$ with 2 components but no such $\mathcal{I}$ is a model of $\mathcal{O}$. The reason is that the axiom $\neg A \sqsubseteq A'$ cannot be satisfied if the extension of both A and A' is restricted to only one component, as required by Definition 4.

As a consequence of this observation, an additional assumption has to be made, which is the DL equivalent of domain-independence known from the first-order and database worlds (Abiteboul et al., 1995). For (most of) $\mathcal{SROIQ}$, this assumption admits an efficiently decidable syntactic characterisation. For domain-independent ontologies, compatibility implies equivalence; see Theorem 13.

Definition 9. A concept C (axiom α) is *domain-independent (DI)* if $C^{\mathcal{I}} = C^{\mathcal{J}}$ ($\mathcal{I} \models \alpha$ iff $\mathcal{J} \models \alpha$) for all interpretations $\mathcal{I}, \mathcal{J}$ with $X^{\mathcal{I}} = X^{\mathcal{J}}$ for all terms X. An ontology is DI if so are all its axioms.

The syntactic characterisation by Cuenca Grau (2005); Cuenca Grau et al. (2005, 2006) is the following.

Definition 10. The set of *local* concepts is defined inductively as follows.

- Every concept name is local.
- $\neg C$ is local if C is not.
- $C \sqcap D$ is local if C or D is local.
- $\geqslant m\, R.C$, $\geqslant m\, P.\mathsf{dr}$, $\exists R.\mathsf{Self}$, and $\{a\}$ are local.

An ontology $\mathcal{O}$ is *safe* if the following hold.

- For all axioms $C \sqsubseteq D$, if D is local, then so is C.
- For all axioms $C \equiv D$, C is local iff so is D.

Given an interpretation $\mathcal{I}$ and a set S, we write $\mathcal{J} = \mathcal{I} \uplus S$ if $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}} \uplus S$ and $X^{\mathcal{J}} = X^{\mathcal{I}}$ for all terms X.

All other axiom types are always DI and therefore do not need to be included in the definition of *safe*. HasKey axioms are alway DI because of the restriction to NAMED in Definition 1.

Theorem 11 ((Cuenca Grau, 2005; Cuenca Grau et al., 2005, 2006)). *For all concepts C and ontologies $\mathcal{O}$:*

1. *C is DI iff C is local.*
2. *If C is DI, then $C^{\mathcal{J}} = C^{\mathcal{I}}$ for all $\mathcal{I}$ and $\mathcal{J} = \mathcal{I} \uplus S$.*
3. *If C is not DI, then $C^{\mathcal{J}} = C^{\mathcal{I}} \cup S$ for all $\mathcal{I}$ and $\mathcal{J} = \mathcal{I} \uplus S$.*
4. *$\mathcal{O}$ is DI iff $\mathcal{O}$ is safe.*

Not only does Theorem 11 provide a syntactic characterisation of DI; with Definition 10 it also yields a linear-time decision procedure. Here it becomes clear why the universal role u (with the semantics $u^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$) cannot be accommodated by the framework:

Example 12. the concept $C = \exists u.A$ is not DI but violates point 3 in Theorem 11: there are interpretations $\mathcal{I}$ and $\mathcal{J} = \mathcal{I} \uplus S$ with $C^{\mathcal{J}} = C^{\mathcal{I}} \cup S$ (e.g., $\Delta^{\mathcal{I}} = A^{\mathcal{I}} = \{d\}$ and $S = \{e\}$); but if $A^{\mathcal{I}} = \emptyset$ then $C^{\mathcal{J}} = C^{\mathcal{I}} = \emptyset$.

However, point 3 is an essential ingredient not just in the syntactic characterisation witnessing decidability of DI, but also in the following theorem linking compatibility and equivalence. Its proof completes the proof of Theorem 7.2 by Cuenca Grau (2005).

Theorem 13. *Let $\mathcal{O}$ be an ontology, ν an I-indexing, $\mathbb{O} = (\mathcal{O}_1, \ldots, \mathcal{O}_n)$ a ν-ontology.*

1. *(a) If $\mathcal{O}$ is DI and $\mathcal{O} \sim \mathbb{O}$, then $\mathcal{O} \approx \mathbb{O}$.*
   (b) If additionally $\mathcal{O}$ is consistent, then so is $\mathbb{O}$.
2. *If $\mathcal{O}$ is consistent and not DI, and $\mathcal{O} \sim \mathbb{O}$ and $\mathcal{O} \approx \mathbb{O}$, then $n = 1$, i.e., $\mathbb{O} = \mathcal{O}$.*

Proof.

Ad 1. (a) Assume that $\mathcal{O}$ is DI and $\mathcal{O} \sim \mathbb{O}$. Let $\mathcal{I}$ be a ν-interpretation. We need to show: $\mathcal{I} \models \mathcal{O}$ iff $\mathcal{I} \models^{\nu} \mathbb{O}$. For this purpose, we first establish the following property of i-concepts.

Claim 1. *For all i-concepts C:*

(i) If C is DI, then $C^{\mathcal{I}} = C^{\mathcal{I},\nu}$.

(ii) If C is not DI, then $C^{\mathcal{I}} = C^{\mathcal{I},\nu} \cup \bigcup_{j \neq i} \Delta^{\mathcal{I}}_j$.

Proof of Claim 1. By induction on C.

- $C = A$ (concept name). Then C is DI by Definition 9, and (i) follows from Definition 4.

- $C = \{a\}$. Dito, but (i) follows from Definition 4 and 5

- $C = \geqslant m\, P.dr$. Dito.

- $C = \exists R.\mathsf{Self}$. Dito.

- $C = \neg D$. Then D too is an i-concept by Definition 1.

  (i) If C is DI, then by Theorem 11 and Definition 10 D is not DI and, by induction hypothesis, $D^{\mathcal{I}} = D^{\mathcal{I},\nu} \cup \bigcup_{j \neq i} \Delta_j^{\mathcal{I}}$. Hence $(\neg D)^{\mathcal{I}} = \Delta_i^{\mathcal{I}} \setminus D^{\mathcal{I},\nu}$, which by Definition 5 equals $(\neg D)^{\mathcal{I},\nu}$.

  (ii) If C is not DI, then D is DI and, by induction hypothesis, $D^{\mathcal{I}} = D^{\mathcal{I},\nu}$. Hence $(\neg D)^{\mathcal{I}} = (\Delta_i^{\mathcal{I}} \setminus D^{\mathcal{I},\nu}) \cup \bigcup_{j \neq i} \Delta_j^{\mathcal{I}}$, which by Definition 5 equals $(\neg D)^{\mathcal{I},\nu} \cup \bigcup_{j \neq i} \Delta_j^{\mathcal{I}}$.

- $C = D \sqcap E$. Then both D and E are i-concepts, too.

  (i) If C is DI, then at least one of D, E is DI. We assume D is DI and E is not; the other two cases are analogous. By induction hypothesis, we get:

  $$D^{\mathcal{I}} = D^{\mathcal{I},\nu} \qquad\qquad E^{\mathcal{I}} = E^{\mathcal{I},\nu} \cup \bigcup_{j \neq i} \Delta_j^{\mathcal{I}}$$

  Hence $(D \sqcap E)^{\mathcal{I}} = D^{\mathcal{I},\nu} \cap E^{\mathcal{I},\nu}$, which by Definition 5 equals $(D \sqcap E)^{\mathcal{I},\nu}$.

  (ii) If C is not DI, then both D and E are not DI. By induction hypothesis, we get:

  $$D^{\mathcal{I}} = D^{\mathcal{I},\nu} \cup \bigcup_{j \neq i} \Delta_j^{\mathcal{I}} \qquad\qquad E^{\mathcal{I}} = E^{\mathcal{I},\nu} \cup \bigcup_{j \neq i} \Delta_j^{\mathcal{I}}$$

  Hence $(D \sqcap E)^{\mathcal{I}} = (D^{\mathcal{I},\nu} \cap E^{\mathcal{I},\nu}) \cup \bigcup_{j \neq i} \Delta_j^{\mathcal{I}}$, which by Definition 5 equals $(D \sqcap E)^{\mathcal{I},\nu} \cup \bigcup_{j \neq i} \Delta_j^{\mathcal{I}}$.

- $C = \geqslant m\, R.D$. Then by Definition 1 R is an (i, j)-role and D a j-concept, for some j. Furthermore, C is local (Theorem 11 and Definition 10), and we have to show (i). By induction hypothesis, we either have $D^{\mathcal{I}} = D^{\mathcal{I},\nu}$ or $D^{\mathcal{I}} = D^{\mathcal{I},\nu} \cup \bigcup_{k \neq j} \Delta_k^{\mathcal{I}}$ (depending on whether D is DI or not). Together with $R^{\mathcal{I}} \subseteq \Delta_i^{\mathcal{I}} \times \Delta_j^{\mathcal{I}}$, we have

$$\begin{aligned}(\geqslant m\, R.D)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \#\{e \in \Delta^{\mathcal{I}} \mid (d,e) \in R^{\mathcal{I}} \text{ and } e \in D^{\mathcal{I}}\} \geq m\} \\ &= \{d \in \Delta_i^{\mathcal{I}} \mid \#\{e \in \Delta_j^{\mathcal{I}} \mid (d,e) \in R^{\mathcal{I}} \text{ and } e \in D^{\mathcal{I},\nu}\} \geq m\} \\ &= (\geqslant m\, R.D)^{\mathcal{I},\nu}\end{aligned}$$

It now suffices to show $\mathcal{I} \models \alpha$ iff $\mathcal{I} \models^{\nu} \alpha$ for every $\alpha \in \mathcal{O}$, proceeding by case analysis over the possible axiom types and using Claim 1 for the cases of concept inclusions, equivalences, and assertions. The cases for the remaining axiom types follow directly from the last bullet point in Definition 5.

- $\alpha = C \sqsubseteq D$. We have to show:

$$C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \quad \text{iff} \quad C^{\mathcal{I},\nu} \subseteq D^{\mathcal{I},\nu} \tag{$*$}$$

  Since $\mathcal{O}$ is DI, there are 3 possible cases concerning the domain-independence of C and D:

  1. Both C and D are DI. Then $(*)$ follows directly from Claim 1 (i).
  2. C is DI, D is not. Then we have

$$\begin{aligned} C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \;\text{ iff }\; & C^{\mathcal{I},\nu} \subseteq D^{\mathcal{I},\nu} \cup \bigcup_{j \neq i} \Delta_j^{\mathcal{I}} && \text{(Claim 1 (i) and (ii))} \\ \text{iff }\; & C^{\mathcal{I},\nu} \subseteq D^{\mathcal{I},\nu} && \text{(since } C^{\mathcal{I},\nu} \subseteq \Delta_i^{\mathcal{I}}\text{)} \end{aligned}$$

  3. Neither C nor D is DI. Analogous to the previous case.

- $\alpha = C \equiv D$. Analogous to the case $C \sqsubseteq D$, requiring only subcases 1 and 3.

- $\alpha = C(a)$. Analogous to the case $C \sqsubseteq D$, requiring only subcases 1 and 2.

Ad 1. (b) It suffices to show that, whenever $\mathcal{O}$ is consistent (and DI and compatible with the ν-ontology $\mathbb{O}$), $\mathcal{O}$ has a ν-model. By 1. (a) that model then is a model of $\mathbb{O}$ too.

For this purpose we first construct from an arbitrary interpretation $\mathcal{I}$ a ν-interpretation $\mathcal{J}$ and then show that $\mathcal{I} \models \mathcal{O}$ implies $\mathcal{J} \models \mathcal{O}$. The construction is by creating n disjoint copies of the domain (ν is an I-indexing) and distributing the original extensions of the terms over the parts corresponding to their ν-value. Thus, if $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, then $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ with:

$$\begin{aligned} \Delta^{\mathcal{J}} &= \Delta^{\mathcal{I}} \times \{1, \ldots, n\} && \\ A^{\mathcal{J}} &= A^{\mathcal{I}} \times \{i\} && \text{for all } A \in \Sigma_{\mathsf{C}} \text{ with } \nu(A) = i \\ r^{\mathcal{J}} &= \{((d,i),(e,j)) \mid (d,e) \in r^{\mathcal{I}}\} && \text{for all } r \in \Sigma_{\mathsf{R}_a} \text{ with } \nu(r) = (i,j) \\ P^{\mathcal{J}} &= \{((d,i),x) \mid (d,x) \in P^{\mathcal{I}}\} && \text{for all } P \in \Sigma_{\mathsf{R}_c} \text{ with } \nu(P) = (i) \\ a^{\mathcal{J}} &= (a^{\mathcal{I}}, i) && \text{for all } a \in \Sigma_{\mathsf{I}} \text{ with } \nu(a) = i \end{aligned}$$

We first establish an auxiliary property that has the same status as Claim 1 in 1 (a):

Claim 2. *For all i-concepts C:*

(i) If C is DI, then $C^{\mathcal{J}} = \{(d,i) \mid d \in C^{\mathcal{I}}\}$.

(ii) If C is not DI, then $C^{\mathcal{J}} = \{(d,i) \mid d \in C^{\mathcal{I}}\} \cup \{(d,j) \mid d \in \Delta^{\mathcal{I}},\ j \neq i\}$.

The proof of Claim 2 is analogous to that of Claim 1 and thus omitted. We can now use it to show the desired property:

$$\mathcal{I} \models \mathcal{O} \quad \text{implies} \quad \mathcal{J} \models \mathcal{O}$$

As in 1 (a) we treat each axiom $\alpha \in \mathcal{O}$ separately, distinguishing its type and using Claim 2 for the cases of concept inclusions, equivalences, and assertions. Let $\mathcal{I} \models \alpha$.

- $\alpha = C \sqsubseteq D$. We need to distinguish two cases.

  If C is DI (then D can be DI or not), then we have

  $$\begin{aligned} C^{\mathcal{J}} &= \{(d,i) \mid d \in C^{\mathcal{I}}\} && \text{by Claim 2 (i)} \\ &\subseteq \{(d,i) \mid d \in D^{\mathcal{I}}\} && \text{since } \mathcal{I} \models \alpha \\ &\subseteq D^{\mathcal{J}} && \text{by Claim 2 (i)/(ii)} \end{aligned}$$

  If C is not DI, then neither is D because $\mathcal{O}$ is DI, and we have

  $$\begin{aligned} C^{\mathcal{J}} &= \{(d,i) \mid d \in C^{\mathcal{I}}\} \cup \{(d,j) \mid d \in \Delta^{\mathcal{I}},\ j \neq i\} && \text{by Claim 2 (ii)} \\ &\subseteq \{(d,i) \mid d \in D^{\mathcal{I}}\} \cup \{(d,j) \mid d \in \Delta^{\mathcal{I}},\ j \neq i\} && \text{since } \mathcal{I} \models \alpha \\ &\subseteq D^{\mathcal{J}} && \text{by Claim 2 (ii)} \end{aligned}$$

- $\alpha = C\ \mathsf{HasKey}(R_1, \ldots, R_n, P_1, \ldots, P_m)$

  First we show that the following holds: Suppose C is not DI, $\nu(C) = i$ and there is an i-axiom of the form $C\ \mathsf{HasKey}(R_1, \ldots, R_n, P_1, \ldots, P_m)$ then

  $$C^{\mathcal{J}} = \{(d,i) \mid d \in C^{\mathcal{I}}\} \cup \{(d,j) \mid d \in \Delta^{\mathcal{I}},\ j \neq i\} = \{(d,i) \mid d \in C^{\mathcal{I}}\} \qquad (*)$$

  We show this by contradiction: Assume that C is not DI and $C^{\mathcal{J}} = \{(d,i) \mid d \in C^{\mathcal{I}}\} \cup \{(d,j) \mid d \in \Delta^{\mathcal{I}},\ j \neq i\}$ with $\{(d,j) \mid d \in \Delta^{\mathcal{I}},\ j \neq i\} \neq \emptyset$. Then there is a $(d,k) \in C^{\mathcal{J}}$ with $(d,k) \in \{(d,j) \mid d \in \Delta^{\mathcal{I}},\ j \neq i\}$. By definition of HasKey atleast one of the following holds: $n > 1$ or $m > 1$

  - If $n > 1$ then there is a $((d,k),(e,p)) \in R_1^{\mathcal{J}}$. By definition of $R^{\mathcal{J}}$ and the assumption follows that $\nu(R_1) = (k,p)$ with $k \neq \nu(C)$. Therefore $C\ \mathsf{HasKey}(R_1, \ldots, R_n, P_1, \ldots, P_m)$ is not a i-axiom according to 2, which contradicts the assumption.
  - If $m > 1$ then there is a $((d,k),x) \in P_1^{\mathcal{J}}$. By definition of $P^{\mathcal{J}}$ and the assumption follows that $num(P_1) = k$ with $k \neq \nu(C)$. Therefore $C\ \mathsf{HasKey}(R_1, \ldots, R_n, P_1, \ldots, P_m)$ is not a i-axiom according to 2, which contradicts the assumption.

  Assume that all $(d,j), (d',j'), (e_1,j_1), \ldots, (e_n,j_n) \in \Delta^{\mathcal{J}} \cap \mathsf{NAMED}_{\mathcal{J}}$ with $\mathsf{NAMED}_{\mathcal{J}} = \{a^{\mathcal{J}} | a \in \Sigma_{\mathsf{I}}\} = \{(a^{\mathcal{I}}, i) | a \in \Sigma_{\mathsf{I}}\}$, all $x_1, \ldots, x_m \in \Delta^{\mathcal{D}}$ and the following holds:

(a) $(d, j), (d', j') \in C^{\mathcal{J}}$ and

(b) $((d, j), (e_i, j_i)), ((d', j'), (e_i, j_i)) \in R_i^{\mathcal{J}}$ for all $i \leq n$ and

(c) $((d, j), x_i)$, $((d', j'), x_i) \in P_i^{\mathcal{D}}$ for all $i \leq m$

We can show that the following holds:

- $d, d', e_1, \ldots, e_n \in \Delta^{\mathcal{I}} \cap \mathsf{NAMED}$. By definition of $a^{\mathcal{J}}$.
- $d, d' \in C^{\mathcal{I}}$. If C is DI, then by (a) and Claim 2 (i). If C is not DI then by (a) and by $(*)$
- (d, e_i), $(d', e_i) \in R_i^{\mathcal{I}}$. By (b) and the definition of $\mathcal{R}^{\mathcal{J}}$
- (d, x_i), $(d', x_i) \in P_i^{\mathcal{D}}$ for all $i \leq m$. By the (c) and the definition of $P^{\mathcal{J}}$

Thus, by $\mathcal{I} \models \alpha$, we have $d = d'$. It is also easy to see, that $j = j'$ by definition of $A^{\mathcal{J}}$ and $d, d' \in C^{\mathcal{I}}$. Hence $(d, j) = (d', j)$.

- $\alpha = C \equiv D$. Then C is DI if and only if D is, and we can repeat the previous argument, proving equality in the first subcase.

- $\alpha = C(a)$. Analogous to case $C \sqsubseteq D$, first subcase.

- The cases $\alpha = R \sqsubseteq S$ and $\alpha = R \equiv S$ are analogous to case $C \sqsubseteq D$, first subcase, invoking the definition of $R^{\mathcal{J}}$ instead of Claim 2, and replacing set inclusion with equality where necessary.

- The remaining cases are obtained by simple modifications to the previous one.

Ad 2. Assume $\mathcal{O}$ and $\mathbb{O}$ satisfy the four premises of the implication, and let $\mathbb{O} = (\mathcal{O}_1, \ldots, \mathcal{O}_n)$. We need to show that $n = 1$. Since $\mathcal{O}$ is not DI, by Theorem 11 and Definition 10 $\mathcal{O}$ contains an axiom $\alpha = C \sqsubseteq D$ (or $C \equiv D$) such that D is DI and C is not. Assume w.l.o.g. that $\alpha \in \mathcal{O}_1$; hence both C and D are 1-concepts. Since $\mathcal{O}$ is consistent, it has a model $\mathcal{I}$, from which we can construct a ν-model $\mathcal{J} \models \mathcal{O}$ as in (1) b above. Claim 2 now implies

$$C^{\mathcal{J}} = \{(d, 1) \mid d \in C^{\mathcal{I}}\} \cup \{(d, j) \mid d \in \Delta^{\mathcal{I}},\ 1 < j \leq n\} \quad \text{and}$$
$$D^{\mathcal{J}} = \{(d, 1) \mid d \in D^{\mathcal{I}}\}.$$

Since $C^{\mathcal{J}} \subseteq D^{\mathcal{J}}$, this implies $n = 1$. ❑

Our proof of 1 (b) completes the corresponding proof of Lemma 7.4 by Cuenca Grau (2005): Claim 2 provides the required strengthening of Claim ♣ by Cuenca Grau (2005), and we additionally provide the part of the proof required to establish $\mathcal{J} \models \mathcal{O}$, using that claim. In addition, our "direct" definition of $\mathcal{J}$ ensures that $\Delta^{\mathcal{J}} = \biguplus_{i \leq n} \Delta_i^{\mathcal{J}}$ (here with $\Delta_i^{\mathcal{J}} = \Delta^{\mathcal{I}} \times \{i\}$), which was not the case in the original "cumulative" construction, which only ensured $\Delta^{\mathcal{J}} \supseteq \biguplus_{i \leq n} \Delta_i^{\mathcal{J}}$.

3.3 Main differences in notation compared to (Cuenca Grau, 2005; Cuenca Grau et al., 2005, 2006)

I-indexed signatures correspond to "partitioned vocabularies" by Cuenca Grau (2005); Cuenca Grau et al. (2005). We consider it easier to carry along the parameter ν instead of a partitioned vocabulary with its rather long denotation. Indexings also extend more naturally to complex concepts and axioms. Consequently, "combined knowledge bases" (a special case of $\mathcal{E}$-connections) are now called ν-ontologies.

ν-interpretations conflate "partitioned interpretations" and "combined interpretations", making "corresponding interpretations" redundant. Consequently we distinguish two semantics: $\mathcal{I} \models \mathcal{O}$ (the standard semantics for simple ontologies) and $\mathcal{I} \models^{\nu} \mathbb{O}$ (the $\mathcal{E}$-connection semantics for ν-ontologies).

Equivalence replaces "semantic compatibility" because it indeed captures equivalence under the class of ν-interpretations. As per the usual understanding of equivalence, $\mathcal{O}$ is not required to have (ν-)models. The implication "if $\mathcal{O}$ consistent, then also $\mathbb{O}$" (under certain assumptions) is now captured by Theorem 13. As a result we reduced "syntactically compatible" to "compatible".

Finally, domain-independence replaces "invariance under domain expansions". We decided to reflect the relationship with the fundamental first-order notion.

Chapter 4

The Partitioning Algorithm

4.1 Description

We now present the partitioning algorithm, based on the preceding definitions. As in the paper by Cuenca Grau et al. (2005, 2006), our algorithm receives as input an ontology $\mathcal{O}$ and returns a ν-ontology $\mathbb{O} = (\mathcal{O}_1, \ldots, \mathcal{O}_n)$ such that $\mathcal{O} \sim \mathbb{O}$ and n is maximal with this property. If $\mathcal{O}$ is domain-independent, $\mathcal{O} \approx \mathbb{O}$ follows by Theorem 13. The indexing ν is not computed by the algorithm but is implicit in the created data structure.

Our algorithm determines n, the $\mathcal{O}_i$, and the implicit ν by "gathering" constraints between the ν-values of the concepts and roles occurring in $\mathcal{O}$ in an undirected and edge-labelled graph G, called the constraint graph, with $G = (V, E, L)$ and $L : E \to 2^{\mathbb{O}}$. Let $\mathsf{sub}(\mathcal{O})$ be the set of concepts (atomic or complex) occurring in $\mathcal{O}$. The graph G has one vertex for each concept in $\mathsf{sub}(\mathcal{O})$, individual or concrete role occurring in $\mathcal{O}$, and two vertices r_0, r_1 per abstract role in $\mathcal{O}$. L labels the edges with a set of axioms. The edges represent the constraints imposed by Definition 1 and 2. For example:

Example 1. Suppose $\alpha = A \sqsubseteq \exists r.\neg B$ and $\beta = B \sqsubseteq B'$, then G has vertices $A, \exists r.\neg B, r_0, r_1, B, B'$, and α, β induce edges $\{A, \exists r.\neg B\}$, $\{\exists r.\neg B, r_0\}$, $\{r_1, B\}, \{B, B'\}$, representing case 5 of Definition 1 and case 9 of 2. The edges $\{A, \exists r.\neg B\}$ and $\{B, B'\}$ are labelled $\{\alpha\}$ and $\{\beta\}$, respectively. Now G has 2 connected components (CCs): G_1 with vertices $A, \exists r.\neg B, r_0$ and label α; G_2 with r_1, B, B' and β. Hence $\mathbb{O} = (\{\alpha\}, \{\beta\})$. The I-indexing with $I = \{1, 2\}$ follows from membership in the G_i: $A, \exists r.\neg B$ are 1-concepts, r is a 12-role, and B, B' are 2-concepts.

This procedure is given by the main routine `partition(`$\mathcal{O}$`)` of Algorithm 1. It first creates the graph G and then adds all edges induced by the structure of the concepts (subroutine `addSubConceptEdges`) and axioms (`addAxiomEdges`) in $\mathcal{O}$ to G. For each role R, both subroutines use the notation R_i, which equals r_i if $R = r$ and r_{1-i} if $R = r^-$, for $i = 0, 1$. Additionally, `addAxiomEdges` labels, for each axiom α, one of the created edges with α. Next, the CCs of G are determined (e.g., via breadth-first search). Finally, the partitioning is read off the axiom labels in the CCs. Not all CCs need to be labelled with some axiom: e.g., if $B \sqsubseteq B'$ is omitted from the above example, we will get the same G_1, G_2, but G_2 will not contain any axiom label. Therefore, the partitioning of $\mathcal{O}$ is determined using only those CCs with ≥ 1 axiom label. We revisit this case below.

Algorithm 1: Partitioning an ontology $\mathcal{O}$

Function `partition`($\mathcal{O}$)**:**

input : $\mathcal{O}$ with signature Σ

output : ν-ontology $\mathbb{O}$

$V \leftarrow \{C \mid C \in \mathsf{sub}(\mathcal{O})\} \cup \Sigma_{\mathsf{I}} \cup \{r_0, r_1 \mid r \in \Sigma_R\} \cup \{P \mid P \in \Sigma_{\mathsf{R}_c}\}$; $E \leftarrow \emptyset$; $L \leftarrow \emptyset$; $G \leftarrow (V, E, L)$

forall $C \in \mathsf{sub}(\mathcal{O})$ **do** `addSubConceptEdges`(G, C)

forall $\alpha \in \mathcal{O}$ **do** `addAxiomEdges`(G, α)

$\{G_1, \ldots, G_n\} \leftarrow$ all connected components (CCs) of G with ≥ 1 axiom label

forall $i \leq n$ **do** $\mathcal{O}_i \leftarrow \{\alpha \mid \alpha \in L(v, v')$ for some edge (v, v') in $G_i\}$

$\mathbb{O} \leftarrow (\mathcal{O}_1, \ldots, \mathcal{O}_n)$

return ($\mathbb{O}$)

Algorithm 2: addSubConceptEdges

Function `addSubConceptEdges`(G, C)**:**

switch C **do**

- **case** $\neg D$ **do** $E \leftarrow E \cup \{C, D\}$
- **case** $D \sqcap F$ **do** $E \leftarrow E \cup \{\{C, D\}, \{C, F\}\}$
- **case** $\geqslant m\, R.D$ **do** $E \leftarrow E \cup \{\{C, R_0\}, \{R_1, D\}\}$
- **case** $\geqslant m\, P.dr$ **do** $E \leftarrow E \cup \{\{C, P\}\}$
- **case** $\exists R.\mathsf{Self}$ **do** $E \leftarrow E \cup \{\{C, R_0\}, \{C, R_1\}\}$
- **case** $\{a\}$ **do** $E \leftarrow E \cup \{\{C, a\}\}$

Algorithm 3: addAxiomEdges

```
1  Function addAxiomEdges(G, α):
2      switch α do
3          case C ⊑ D or C ≡ D do
4              E ← E ∪ {C, D};
5              updateLabel(C, D, α)
6          case R ⊑ S, R ≡ S or Disjoint(R, S) do
7              E ← E ∪ {{R_0, S_0}, {R_1, S_1}};
8              updateLabel(R_0, S_0, α)
9          case P_1 ⊑ P_2, P_1 ≡ P_2 or Disjoint(P_1, P_2) do
10             E ← E ∪ {{P_1, P_2}};
11             updateLabel(P_1, P_2, α)
12         case R ∘ S ⊑ T do
13             E ← E ∪ {{R_1, S_0}, {R_0, T_0}, {S_1, T_1}};
14             updateLabel(R_0, T_0, α)
15         case C(a) do
16             E ← E ∪ {C, a};
17             updateLabel(C, a, α)
18         case R(a, b) do
19             E ← E ∪ {{a, R_0}, {R_1, b}};
20             updateLabel(a, R_0, α)
21         case P(a, x) do
22             E ← E ∪ {{P, a}};
23             updateLabel(P, a, α)
24         case a ≈ b or a ≉ b do
25             E ← E ∪ {{a, b}};
26             updateLabel(a, b, α)
27         case HasKey(C, R^1, ..., R^n, P_1, ..., P_m) do
28             E ← E ∪ {{C, R_0^1}, ..., {C, R_0^n}, {C, P_1}, ..., {C, P_m}}};
29             if n ≥ 1 then
30                 updateLabel(C, R_0^1, α)
31             else
32                 updateLabel(C, P_1, α)
```

Algorithm 4: updateLabel

```
1 Function updateLabel(C, D, α):
2     if L(C, D) undefined then
3         L(C, D) ← {α}
4     else
5         L(C, D) ← L(C, D) ∪ {α}
```

4.2 Correctness

Now we want to show that the algorithm returns a combined ontology $\mathbb{O}$ that is compatible with the input ontology $\mathcal{O}$. With Theorem 13 we also have that $\mathcal{O}$ and $\mathbb{O}$ are equivalent, provided that $\mathcal{O}$ is domain-independent.

Theorem 2. *If* $\mathbb{O} = \texttt{partition}(\mathcal{O})$*, then* $\mathbb{O}$ *is a* ν*-ontology for some* ν*, and* $\mathcal{O} \sim \mathbb{O}$.

Proof. $\mathcal{O} \sim \mathbb{O}$ follows directly from lines 4–8 of `partition`. For the existence of ν, let $G = (V, E, L)$ be the graph created in lines 1–4 of `partition`, $G_1, \ldots, G_n$ the CCs with ≥ 1 axiom label and $G_{n+1}, \ldots, G_m$ with $m \geq n$ the remaining CCs. Let ν be a function assigning a index $\leq n$ to each vertex in G and each axiom in $\mathcal{O}$:

- for every $i \leq n$: $\nu(x) = i$ for all vertices x in G_i
- for every $i > n$: fix some $j \leq n$ and let $\nu(x) = j$ for all vertices in x in G_i
- for all edges $\{x, y\}$ in G_i with $L(\{x, y\}) = \alpha$: $\nu(\alpha) = i$

ν contains an I-indexing of Σ if we additionally set:

- for all roles $r \in \Sigma_R$: $\nu(r) = (\nu(r_0), \nu(r_1))$

It remains to show that ν respects Definition 1 and 2.

1. ν is a I-indexing of $\Sigma = \mathsf{sig}(\mathcal{O})$.
2. ν on arbitrary concepts is an extension of ν on Σ that respects Definition 1, i.e., every $C \in \mathsf{sub}(\mathcal{O})$ is a $\nu(C)$-concept. This is an easy induction on the structure of C.
   - $C = \neg D$: By line 3 of of the algorithm G contains the edge $\{C, D\}$. Hence C, D are in the same CC. Then $\nu(C) = \nu(D)$.
   - $C = \geqslant m\, R.D$: By line 5, G contains the edges $\{C, R_0\}$, $\{R_1, D\}$. Hence C, R_0 are in the same CC, and so are R_1, D. Then $\nu(C) = \nu(R_0) = i$, $\nu(D) = \nu(R_1) = j$ and $\nu(R) = (i, j)$.
   - All other cases are analogous.

3. $\nu(\alpha)$ respects the second part of Definition 2, i.e., every $\alpha \in \mathcal{O}$ is a $\nu(\alpha)$-axiom. This is an easy case distinction on the type of α.
   - $\alpha = C \sqsubseteq D$: By line 5, G contains the edge $\{C, D\}$ with $L(\{C, D\}) = \alpha$. Hence C, D and $\{C, D\}$ are in the same CC and $\nu(C) = \nu(D) = \nu(\alpha)$.
   - All other cases are anologous. ❏

4.3 Maximality

The result of the algorithm is the maximal $\mathbb{O}$ with $\mathcal{O} \sim \mathbb{O}$ in the following sense. Let $\mathbb{O} = (\mathcal{O}_1, \ldots, \mathcal{O}_n)$, $\mathbb{O}' = (\mathcal{O}'_1, \ldots, \mathcal{O}'_m)$ be a ν- and a ν'-ontology with $\bigcup_{i \leq n} \mathcal{O}_i = \bigcup_{i \leq m} \mathcal{O}'_i$. We write $\mathbb{O} \preceq \mathbb{O}'$ ("$\mathbb{O}$ is at least as coarse as $\mathbb{O}'$") if for every $\mathcal{O}_i$ there is an $\mathcal{O}'_i$ s.t. $\mathcal{O}_i \subseteq \mathcal{O}'_i$. In other words: if $\mathbb{O} \preceq \mathbb{O}'$ then every $\mathcal{O}'_i$ is the union of one or several $\mathcal{O}_i$.

Theorem 3. *If* $\mathbb{O} = \texttt{partition}(\mathcal{O})$ *then, for every* $\mathbb{O}'$ *with* $\mathcal{O} \sim \mathbb{O}'$*, we have* $\mathbb{O} \preceq \mathbb{O}'$.

Proof. Let $G = (V, E, L)$ the graph created in lines 1–4 of `partition` and $\mathbb{O} = (\mathcal{O}_1, \ldots, \mathcal{O}_n)$; furthermore let $\mathbb{O}' = (\mathcal{O}'_1, \ldots, \mathcal{O}'_m)$ be a ν'-ontology. It suffices to show:

$$\text{For all } i \leq n \text{ and all } \alpha, \beta \in \mathcal{O}_i\text{: } \nu'(\alpha) = \nu'(\beta) \qquad (*)$$

To prove $(*)$ we will first show an auxiliary property, which involves the obvious extension of ν' to all vertices in G; in particular, if r is a role name with $\nu'(r) = (i, j)$, then $\nu'(r_0) := i$ and $\nu'(r_1) := j$.

Claim 1. *For every* $x, y \in V$*: if* x, y *are in the same CC of* G*, then for all indexings* ν' *of* Σ *we have* $\nu'(x) = \nu'(y)$.

Claim 1 is proven via induction on the distance between x, y in their CC. The base case $x = y$ is trivial. For the induction step, suppose that the claim holds for x, x' (by IH) and there is an edge between x' and y. Now $\nu'(x') = \nu'(y)$ can be shown via a straightforward case distinction on the creation of the edge in the functions `addSubConceptEdges` and `addAxiomEdges` together with Definition 1 and 2. Hence $\nu'(x) = \nu'(y)$.

We can now prove $(*)$ proceeding by axiom types and analysing the respective cases of `addAxiomEdges`. We just show one case; the remaining ones are very similar.

- $\alpha = C_1 \sqsubseteq D_1$ and $\beta = C_2 \sqsubseteq D_2$: Since α, β are in the same CC, then so are C_1, C_2 (by construction of ν and G). Claim 1 implies $\nu'(C_1) = \nu'(C_2)$ and by Definition 2: $\nu'(\alpha) = \nu'(\beta)$. ❏

4.4 Complexity

Let $\mathcal{O}$ be the input ontology, and $G = (V, E, L)$ the constraint graph created in lines 1–4 of `partition`. Set $k = |\mathcal{O}|$ (representing the number of axioms), $\ell = |\mathsf{sub}(\mathcal{O})|$

(representing the number of subconcepts), $m = |\Sigma|$ (representing the length of vocabulary) and $p = max\{max\{n \mid C\ \mathsf{HasKey}(R_1, \ldots, R_n, P_1, \ldots, P_m) \in \mathcal{O}, c\}, max\{m \mid C\ \mathsf{HasKey}(R_1, \ldots, R_n, P_1, \ldots, P_m) \in \mathcal{O}\}\}$ (representing the max number of abstract or concrete roles in the HasKey axioms).

We first consider the size of the created constraint graph. The number of vertices is restricted by $\ell + 2m$ (2 vertices per role). It is easy to see that `addAxiomEdges` is called k-times and each call adds at most h edges with $h = max\{3, p\}$. `addSubConceptEdges` adds at most 2 edges and is called ℓ times. Hence the number of edges is limited by $hk + 2\ell$. Each call to `addAxiomEdges` and `addSubConceptEdges` is in at most linear time if HasKey axioms occur in $\mathcal{O}$. The calls are in constant time if no HasKey axiom occurs in $\mathcal{O}$.

The connected components of G can be calculated using breadth-first search in $O(|V| + |E|) = O(hk + \ell + m)$ (Hopcroft and Tarjan, 1973). The components with ≥ 1 axiom label can be found going through all edges in $O(|E|) = O(hk + \ell)$. To collect the axioms for each component in line 5–6 every edge needs to be checked once. Therefore, this step is also in $O(|E|) = O(hk + \ell + m)$.

Altogether the `partition` algorithm takes linear time without HasKey axioms in $\mathcal{O}$, limited by $O(3k + \ell + m)$, in contrast to Theorem 7.2 by Cuenca Grau (2005), which is quadratic. The algorithm takes quadratic time with HasKey axioms, limited by $O(hk + \ell + m)$. We consider the linear time complexity as the key result, because we observed that HasKey axioms play a minor role in modelling, occuring only once in all 16.3 million axioms of the 2017 BioPortal Snapshot by Matentzoglu and Parsia (2017).

4.5 Discussion

We conclude this section with remarks concerning the implicit labelling of terms in the input ontology, the treatment of $\top$, and the deterministic character of our algorithm.

`partition`($\mathcal{O}$) creates a ν-ontology $\mathbb{O}$ with $\mathcal{O} \sim \mathbb{O}$. The corresponding indexing ν is induced by the CCs of the created constraint graph G. As we have already pointed out in the description of the algorithm, some CCs may not contain any axiom label. In this case the indexing of the "unlabelled" CCs is arbitrary, as seen in the proof of Theorem 2. If a unique indexing is required, e.g., in order to assign "home components" in $\mathbb{O}$ to the terms in $\mathcal{O}$, then user intervention is required:

Example 4. Suppose $\mathcal{O} = \{A \sqsubseteq \exists r.\neg B,\ A' \sqsubseteq \exists r'.B\}$. Then there are 3 CCs: G_1 with vertices $A, r_0, \exists r.\neg B$; G_2 with $A', r'_0, \exists r'.B$; G_3 with r_1, r'_1, B. Now G_3 is not axiom-labelled, and so r_1, r'_1, B could be assigned to either G_1 or G_2. Only the user can make that decision, and they need to know the respective subdomains: if the axioms are rewritten into $\mathtt{HappyChild} \sqsubseteq \exists \mathtt{hasPet}.\mathtt{Puppy}$ and $\mathtt{HappyDog} \sqsubseteq \exists \mathtt{hasChild}.\mathtt{Puppy}$, then G_3 should be merged with G_1.

Algorithm 1 does not treat $\top$ explicitly. The required additions are straightforward but, from a theoretical point of view, unnecessary since $\top$ can be rewritten as $A \sqcup \neg A$, as usual. In that case, a *fresh* concept name A should be used for each occurrence of $\top$; Otherwise

vertices and the corresponding subdomains might be spuriously connected, e.g., $\texttt{hasToy}_1$ and $\texttt{hasFood}_1$ if $\mathcal{O}$ contains $\texttt{HappyChild} \sqsubseteq \exists \texttt{hasToy}.\top$ and $\texttt{HappyDog} \sqsubseteq \exists \texttt{hasFood}.\top$

Compared with the previous algorithms (Cuenca Grau, 2005; Cuenca Grau et al., 2005, 2006), neither `partition` nor its subroutines make any nondeterministic choices. In particular, the computed partition is unique up to permutation of the components because the generated graph G does not depend on the order in which concepts and axioms are traversed. This last property is much less obvious in the previous algorithms, which contain a nondeterministic choice without a rigorous argument that the result is still deterministic (and we did observe a nondeterministic behaviour of the prototype implementation of the previous algorithm (Cuenca Grau, 2005; Cuenca Grau et al., 2005, 2006)).

4.6 Run-Through Example for Algorithm 1

Let us consider the ontology $\mathcal{O} = \{\alpha, \beta, \gamma, \delta\}$ with $\alpha = A \sqsubseteq \exists r.\neg B$, $\beta = C \sqsubseteq \forall s.B$, $\gamma = r \sqsubseteq s$ and $\delta = B \sqsubseteq B' \sqcap B''$. In the Algorithm 1 we have no case for the universal quantification $\forall s.B$. To handle them we will use the equivalent concept $\neg\exists s.\neg B$. The steps of the algorithm are as follows:

Step 1 In line 2 of Algorithm 1: V is set to $\{A, C, \exists r.\neg B, B, \neg\exists s.\neg B, \exists s.\neg B, \neg B, B' \sqcap B'', B', B'', r_0, r_1, s_0, s_1\}$. E, L are empty.

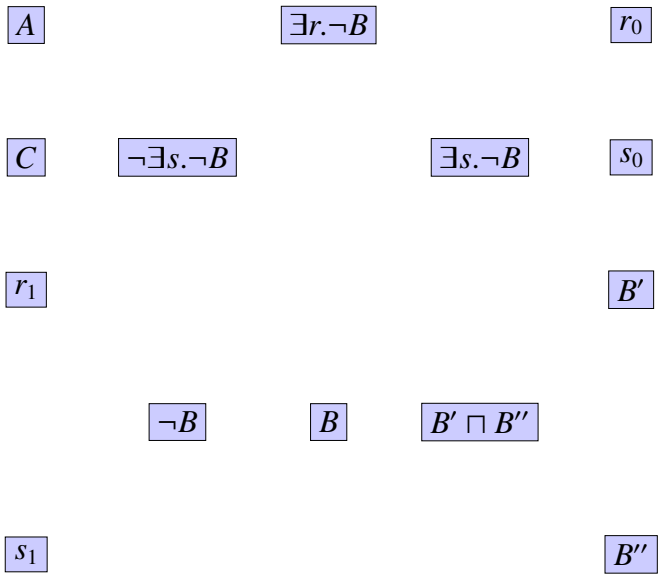

Fig. 4.1: Step 1 of example: Add vertices

Step 2 `addSubConceptEdges` is called in line 3 of Algorithm 1. This calls line 2 to 5 of Algorithm 2: The edges $(\exists r.\neg B, r_0)$, (r_1, B), $(\neg\exists s.\neg B, \exists s.\neg B)$, $(\exists s.\neg B, s_0)$, (s_1, B), $(B' \sqcap B'', B')$ and $(B' \sqcap B'', B'')$ are added to E.

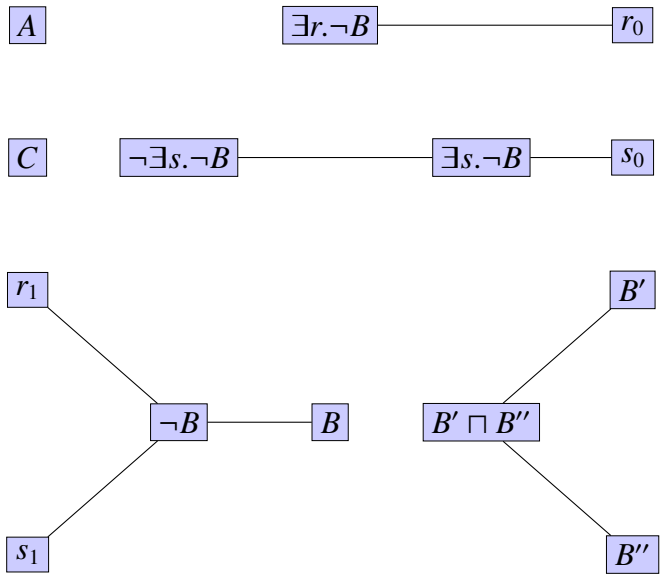

Fig. 4.2: Step 2 of example: Calls to addSubConceptEdges

Step 3 `addAxiomEdges` is called in line 4 of Algorithm 1. This calls line 2 to 8 of Algorithm 3: The edges $(A, \exists r.\neg B)$, $(C, \neg\exists s.\neg B)$, (r_0, s_0), (r_1, s_1) and $(B, B' \sqcap B'')$ are added to E and `updateLabel` is called. The calls to `updateLabel` set L to $\{((A, \exists r.\neg B), \alpha), ((C, \neg\exists s.\neg B), \beta),$
$((r_0, s_0), \gamma), ((B, B' \sqcap B''), \delta)\}$.

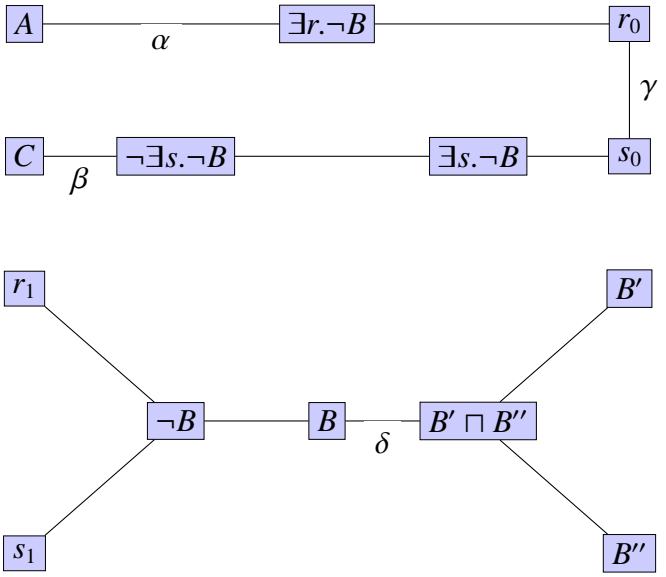

Fig. 4.3: Step 3 of example: Calls to addAxiomEdges

Step 4 In line 6 of Algorithm 1, G has two CCs: G_1 consists of vertices $A, C, \exists r.\neg B, \neg\exists s.\neg B, \exists s.\neg B, r_0, s_0$ and G_2 consists of $r_1, s_1, B, B' \sqcap B'', B', B''$.

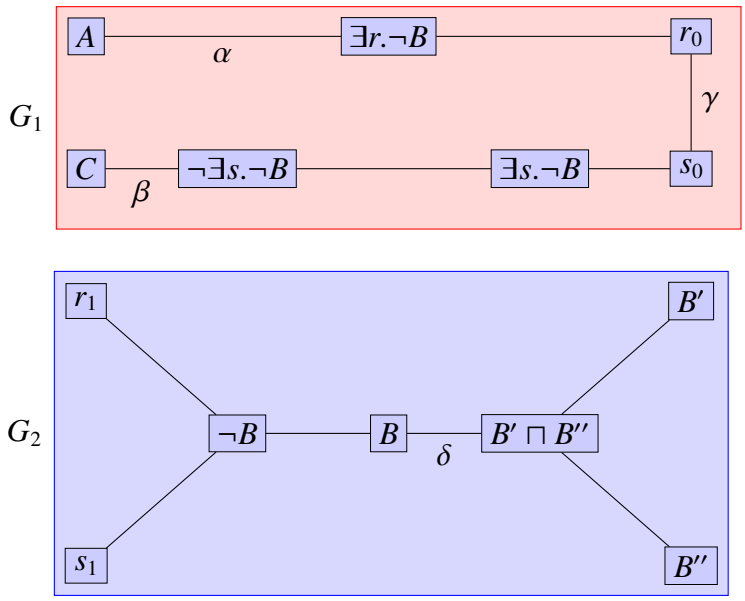

Fig. 4.4: Step 4 of example: Connected components

Step 5 In lines 6 and 7 of Algorithm 1, we get $\mathbb{O} = (\mathcal{O}_1, \mathcal{O}_2)$ with $\mathcal{O}_1 = \{\alpha, \beta, \gamma\}$ and $\mathcal{O}_2 = \{\delta\}$.

If we remove δ from the input ontology, then the run is analogous, but G_2 will not have an axiom label (see also the remark in Chapter 4.1). Consequently $\mathbb{O}$ is the singleton $\mathcal{E}$-connection $(\mathcal{O})$, as expected.

It is easy to see that in the implementation you can save the conversion to the equivalent formula by handling $\forall s.B$ exactly like $\exists s.B$ in the algorithm. In the graph 4.5 we specify the resulting constraint graph of an accordingly extended algorithm.

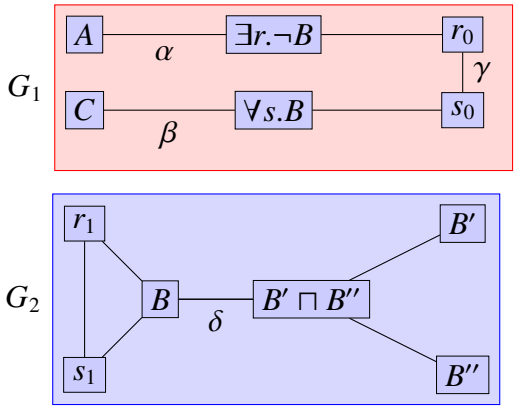

Fig. 4.5: Step 4 of of the extended algorithm on the example

Chapter 5

Heuristics

We have learned in our experiments that the partitioning of many ontologies results in a single partition containing the entire ontology. Another significant portion of the partitions has a part that contains almost all (i.e., >95%) logical axioms of the original ontology. Further observations have shown that this ”collapse” of the axioms into one component is due to a small number of vertices and edges connecting two subgraphs (and thus possibly ”topics” of the ontology). Finding and removing these vertices and edge would result in a finer partitioning. In this chapter, we give a description of the heuristics and discuss them. The heuristics presented here are roughly sorted according to how much input from the notzer is required. An evaluation of the heuristics can be found in chapter 7.

An important term we will use in this chapter is the term ”connected vertex”. A ”connecting vertex” of a graph is a vertex whose removal results in the graph containing more connected components. Accordingly, a ”connecting edge” of a graph is an edge whose removal results in the graph containing more connected components.

It is important to note that after removing axioms from the ontology according to a heuristic, the result of partitioning is no longer equivalent to the original ontology. It is, however, equivalent to the adjusted ontology that is created by the heuristic, if the heuristic only adjusts the input ontology before the run of the algorithm. The user must therefore decide whether the loss of knowledge is acceptable. This question is not investigated in the scope of this thesis. In addition, a non-equivalent partitioning is also interesting to see the general structuring of the ontology.

5.1 Ontology Level Reducer Heuristic (OLH)

We have observed in our tests that the connecting vertices often represent top-level classes. An example would be an ontology about food, that uses the class Food as a top-level concept. Any other class in the ontology is, therefore, a direct or indirect subclass of Food. According to the Definitions 1 and 2 this would result in all classes being pulled into one component. This will result in a partition with only one component.

We implemented a heuristic that removes top-level classes level-wise. For the first level, we remove all classes that are direct subclasses of $\top$ except for classes that do not have subclasses. In the following level, we remove the subclasses of the classes we

removed in the previous step. The number of levels of this step can be adjusted by the user.

5.1.1 OLH After

We have also implemented an alternative or complementary heuristic that, after removing the first level, removes for each subsequent level only the concepts of the largest component (if they have more than a certain percentage of the logical axioms of the original ontology). This is to avoid unnecessary removal of concepts of small components.

5.2 Biconnectivity Heuristic (BH)

While studying the results of the OLH, we have observed that sometimes concepts are removed even though their removal has not reduced the connectivity of the graph. This is because the vertex representing the concept is not a connecting vertex. Mihai Pomarlan from the University of Bremen provided the idea to locate the corresponding vertices and edges, and thus their concepts, roles, and axioms precisely and automatically. A promising approach seemed to be the finding of biconnected components.

A graph is called biconnected if the removal of an arbitrary vertex will still result in a connected graph. A biconnected component of a graph is a maximal subgraph that is biconnected. For this heuristic, we try to find biconnected subgraphs (biconnected components). This is done by using a linear time depth-first algorithm by Hopcroft and Tarjan (1973). These components are connected by shared edges called bridges. This heuristic removes these edges to reduce the connectivity of the graph. When removing these edges, we also have to remove their creating axioms and all vertices and edges created by these axioms. This is necessary to maintain the consistency of the represented knowledge. We allow the user to set a limit to how many axioms may be removed by removing an edge. To limit the loss of axioms, we also allow users to set a limit to how many axioms may be removed by removing an edge.

5.3 Community Detection Heuristic (CD)

We have also investigated an alternative way to locate these connecting vertices and edges. In this heuristic, we try to identify the dense groups of vertices instead of the connecting edges and vertices. The finding of such groups is called Community Detection. Community Detection is a well-researched topic in the field of big data research (Zhang et al., 2018).

One of the most used algorithms for community detection is the Louvain algorithm (Blondel et al., 2008). In their article, Traag et al. (2018) introduced the Leiden algorithm for community detection, which they claim is faster and results in better partitioning than the Louvain algorithm. To demonstrate this claim, Traag et al. (2018)

implemented both algorithms in Java and made it available on Github[1]. The ease of integration into our software was the main reason to choose these two algorithms for our experiments.

After the chosen algorithm has detected the communities, we remove all bridges between these communities. In doing so, as with the Biconnectivity Heuristic, we have to remove their creating axioms and all vertices and edges created by these axioms.

5.4 Upper-Level Ontology Heuristic (ULH)

Hoehndorf (2010) describes upper-level ontologies as "an ontology that defines and axiomatizes these most general categories". They are used to facilitate the development of ontologies by given a basic structure for the top-level concepts. Known upper-level ontologies are the Basic Formal Ontology (BFO), the Business Objects Reference Ontology (BORO), and the Common Semantic Model (COMO).

Our implemented heuristic allows us to store several upper-level ontologies in a single file. This file can then be used to check another ontology for the occurrence of one of the stored upper-level ontologies. If the occurrence is detected (i.e., a given threshold of the concepts are contained in the ontology), we remove the corresponding upper-level ontology.

In contrast to the heuristics mentioned so far, this heuristic can be seen as the least automatic heuristic. The user has to select the upper-level ontologies to be tested and add them to the upper-level file. This would, therefore, also allow a specific selection of the concepts to be removed. It becomes apparent that a very specific selection is usually necessary for the individual ontologies. For this reason a further evaluation of this heuristic is difficult.

5.5 Ignore Properties Heuristic (IPH)

The idea for our last heuristic was born in the expert interview while evaluating our algorithm with the ontology SNOMED CT (see Chapter 7.1.3). As a goal of heuristics one can see to divide ontology into different topics, e.g. diseases and organisms. We found that some properties are used in several different topics of the ontology, although these topics are not otherwise related. An example would be the property "Associated With", which can describe the association of diseases with each other, but also of organisms with each other. This use will (according to definitions 1 and 2) result in the diseases and organism topics being combined into one component.

From this observation, we have decided that it can be useful to ignore these discussed properties when creating the graph. We have divided these properties into three groups. Rangeglobal properties are properties that occur only in their range in different topics of the ontology. Domainglobal properties occur only in their domain in different "topics" of the ontology. Global properties are properties that are rangeglobal and domainglobal.

[1] CWTSLeiden Network Analysis: `https://github.com/CWTSLeiden/networkanalysis`

If this heuristic is switched on, the algorithm is adjusted as follows: For range-global properties, we will no longer draw edges to their "R_1" vertices. For domain-global properties, we will no longer draw edges to their "R_0" vertices. For global properties, both these restrictions apply.

For this heuristic, the user has to define the global, rangeglobal, and domainglobal properties in advance. According to our observation, this is only possible for experts of the ontology, because only they can decide which parts of the ontology correspond to different topics.

In contrast to the other heuristics discussed, this heuristic changes the behavior of the algorithm. For this reason, unfortunately, no statement can be made about the equivalence of the partition and the original ontology. Nevertheless, we believe that the resulting partition might be interesting for an ontology developer to better illustrate the structure of the ontology.

Chapter 6

Implementation

We have implemented the proposed algorithm in Java. We call the implemented tool EPartitioner. The implementation and documentation can be found on Github[1]. EPartitioner takes ontologies as input and returns either a graph representing the partitioning or the partitioning in the form of individual ontologies.

6.1 High-Level View of the System Structure

6.1.1 Core Implementation

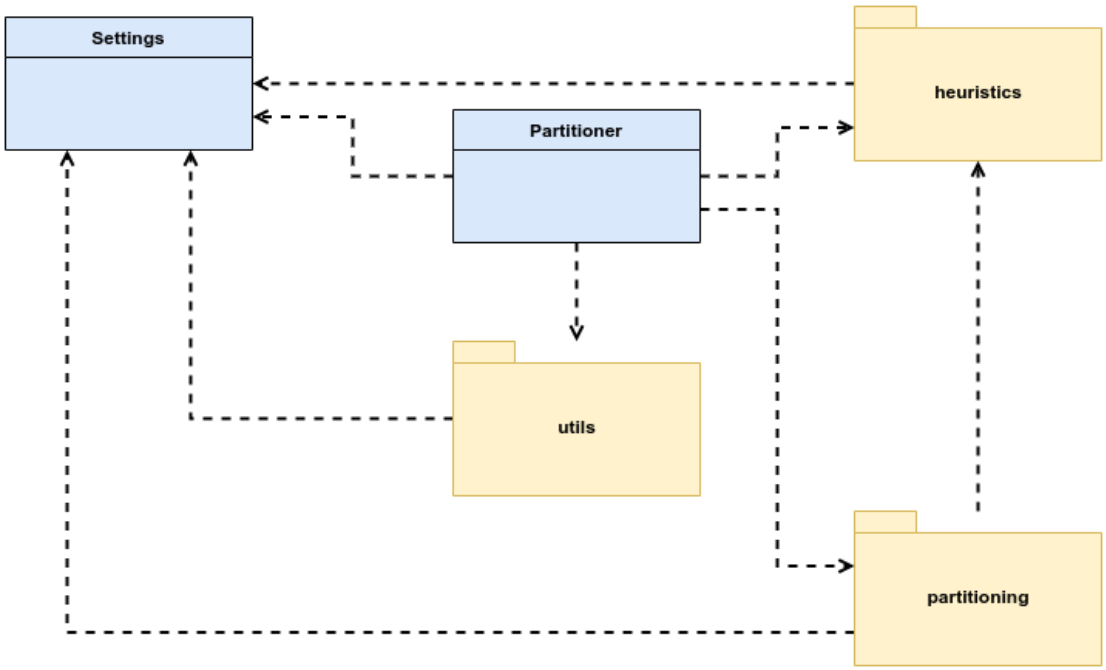

Fig. 6.1: Structure of Core Implementation

First, we want to describe the structure of the core of our implementation. This description of the rest of the software is done in the next sections.

[1] https://github.com/sasjonge/epartition/

Fig. 6.1. is a diagram showing the structure of the implementation. The main class of the tool is the ”Partitioner” class. This class determines the program flow.

The package ”Partitioner” is the logical core of the implementation. The associated class ”PartitioningCore” implements the algorithm we introduced (see Chapter 4.1). In addition, this package contains the implementation of the safety checker according to the procedures introduced by Cuenca Grau (2005).

In the class ”Settings”, a large number of setting options are offered. These can be adjusted by the user during use. A description of these options can be found in chapter 6.3.

In particular, ”Settings” allows you to switch on different heuristics. These heuristics are called either by the ”Partitioner” class or directly by the ”PartitioningCore”. We offer four types of heuristics: the ”Ontology Level Reducer Heuristicc”, the ”Community Detection Heuristic”, the ”Biconnectivity Heuristic”, and the ”Upper Level Heuristic”. A detailed description of these heuristics can be found in chapter 5. The ”Ignore Properties Heuristic” is implemented directly in the algorithm in the ”PartitioningCore”. The package util contains only one abstract class that helps to remove edges created by given axioms. This functionality is used by several heuristics.

The last important implementation for EPartitioner is the class ”GraphExporter” which is in the package utils. This allows to output a graph representing the partitioning, but also the constraint graph. The display of the graph can be customized using the settings in the Settings class. A detailed description of the graph output can be found in chapter 6.2.1. The utils package also contains classes that help to display the graphs, especially to calculate the labels for the different partitions.

6.1.2 Evaluation

The core implementation is extended by the implementation of the evaluation and analysis functions. It is possible to output the runtime of the individual steps. We can run these through a larger set of ontologies. The data can be averaged over several runs. The package ”dataanalyzing” includes an analysis tool to output the number of ontologies of a folder that are ”unsafe”, contain the universal role, or both.

6.1.3 Used Libraries

For the implementation we used several libraries. In this chapter we will introduce and describe the most important of these libraries.

6.1.3.1 OWL API

One of the most important libraries used by EPartitioner is the OWL API. The OWL API is “a high level Application Programming Interface (API) for working with OWL ontologies” (Horridge and Bechhofer, 2011) and the standard API for working with OWL ontologies. We use the API to load the ontologies and handle their subconcepts and axioms. The API is also used for creating the output ontologies.

6.1.3.2 JGraphT

As described in chapter 4.1, our algorithm builds a graph. For this purpose, we use the library JGraphT (Michail et al., 2019). JGraphT offers an easy way to model the graph. Another advantage of the library is the algorithms already available. In the context of this work, the implementation of the finding of connected components offered by JGraphT is particularly important[2]. Finding connected components is solved efficiently via standard traversal algorithms such as BFS or DFS.

JGrapht is also used to build the output graph. For this purpose it is useful that the library has already implemented the possibility to output the graph in the GraphML format. GaphML is a standard format used by many applications, e.g. by the "yEd graph editors"[3] used for this thesis.

6.1.3.3 CWTSLeiden networkanalysis

In Chapter 5.3 we describe, how we use community detection as a heuristic for EPartitioner. Traag et al. (2018) introduced the Leiden algorithm for community detection. To demonstrate the advantages of the Leiden algorithm compared to one of the most popular Community Detection algorithm (the Louvain algorithm), Traag et al. (2018) has implemented both algorithms in Java. We use this implementation for our community detection heuristics.

6.2 Output

EPartitioner provides two kinds of outputs: The parts of the partition as ontologies (in ".owl" format) or as a partition structure graph.

6.2.1 Graphs

Our algorithm provides the option to output two kinds of graphs.

6.2.1.1 Partition Structure Graph

The partition structure graph visualizes the result of EPartitioner. The vertices of the graph represent the components of the partition. They are labeled with information about the represented components. The edges represent the corresponding link relations.

All vertex labels start with some statistics about the represented component. This part of the label has the following form: "Number of logical axioms (Num. of non-logical axioms) / Number of Classes / Number of Properties / Number of Individuals". The rest of the vertex labels represent the classes, properties, and individuals that "live" in this component. For classes, we differentiate between top- and sub-level labels.

[2]JGraphT's Connectivity Inspector: `https://jgrapht.org/javadoc/org/jgrapht/alg/connectivity/ConnectivityInspector.html`

[3]yEd Graph Editor: `https://www.yworks.com/products/yed`

For the top-level, we allow three types of descriptors. The first type of descriptors has the form "[Name]". They represent the class with the given name and all its subclasses. The class has to be a class without super-classes (therefore a top-level class). The second type of labels has the form "{Name1, Name2, Name3}". They represent that the classes Name1, Name2, and Name3 (and all of its subclasses) "live" in this component. The classes are grouped because they have the same super-class, but not all sub-classes of this super-class are in this component. We call the classes in this kind of label the sublevel classes. The last form is simply the name of the class. This form is used when the class, but not all of its subclasses, are in this component and it does not belong to one of the aforementioned groups. The vertex labels for properties are structured in the same way. The labels for individuals are just the names of the individuals. It is also possible to add a certain number of axioms from the component to the label.

The edges of the graph are labeled with a predefined number of corresponding links between the components.

6.2.1.2 Constraint Graph

In addition to the partition structure graph, we provide the option to output the constraint graph that is created in the run of the algorithm (see Chapter 4.1). While this option was mainly useful to debug the implementation, it could also help the user to visualize the relations in the ontology.

6.2.2 Ontologies

EPartitioner allows the output of partitions in the form of ontologies (i.e. OWL files). It is important to note that OWL does not support $\mathcal{E}$-connections. For this reason, the parts will also contain the declarations of classes, properties, and individuals that actually "live" in other parts.

6.3 Usage

In this section, we describe how the *EPartitioner* can be used. In general, the partitioner can be called in the following way:

```
1   > java -jar epartitioner-0.1.jar -t
```

The partitioner is highly customizable. The user can customize the in- and output of the tool, the used heuristics the details of the heuristics.

If the *EPartitioner* is called without command-line arguments, it will try to get the values for the settings from a *settings.xml* file, that is in the same directory as the ".jar". If no such file is provided, that call will fail with an error message. If a settings file is provided, additional command-line arguments will overwrite the values of the file.

We describe the command line arguments for the *EPartitioner* in the following section.

Table 6.1: Settings File Options

Option	Description	Default
--settings_file= <path_to_settings_file>	Sets the path of the settings file	Location of the .jar

Table 6.2: Input Options

Option	Description	Default
--input_directory= <path_to_input_directory>	Sets the path of the input directory, containing one or several ".owl"-files	Location of the .jar

6.3.1 Settings file

We offer the option to give a path to the settings file (see Table 6.1). If the user passes additional command-line arguments, these are used instead of the values in the settings file.

6.3.2 In- and Output Options

Several options to customize the in- and output are available (see Table 6.2). The first set of options belonging to this group are there to customize the input. The user can set the path for the input ontologies.

Another set of options belonging to this group are there to enable and customize the graph output (see Table 6.3). We can customize the output directory for the graphs, the type of graph, and the form of the labels. For understanding, it is important to note, that the class and properties labels are structured by top- and sublevel as described in Chapter 6.2.1.

The last set of options belonging to this group allow adjusting the output of ontologies (see Table 6.4).

6.3.3 Runtime Options

In this chapter, we will introduce the various customization options that affect the runtime behavior of the tool.

In Chapter 4.1 we describe that we add vertices r_0, r_1 per abstract role $r \in \mathcal{O}$ to the constraint graph. Instead of 0 and 1, it can be useful to use other designators for these vertices because 0 and 1 can appear as suffixes, e.g., of classes. This could cause a vertex representing a class with the suffix 0 to be misinterpreted as a r_0 node.

Due to the frequency of using 0 and 1 as suffixes, we have decided to use [0] and [1] as default designators. [0] and [1] do not appear as suffixes for classes, individuals, or roles in any of the ontologies considered in this thesis. For example, none of BioPortal's

Table 6.3: Graph Output Options

Option	Description	Default
--graph_output_path= <path_to_graph_output_direct(	Sets the path of the graph-output directory, to which the ".graphml" files are saved	Location of the .jar
--graph_type=<ID of graph type>	Set the graph type. 0 = Partition structure graph; 1 = Constraint graph.	0
--show_axioms	The vertex labels of the partition structure graph will contain axioms	false
--axiom_count=<Num. of axioms>	Max. number of axioms to print in the vertex labels	3
--number_of_indiv_labels= <Num. of individuals>	Max. number of individuals to print in the vertex labels	5
--number_of_property_labels_edge= <Num. of properties>	Max. number of properties to print in the edge labels	4
--number_of_class_labels_toplevel= <Num. of classes>	Max. number of top-level labels for classes to print in the vertex labels	5
--number_of_class_labels_sublevel= <Num. of classes>	Max. number of sublevel labels for classes to print in the vertex labels	5
--number_of_property_labels_vertex <Num. of properties>	Max. number of top-level labels for properties to print in the vertex labels	4
--number_of_property_labels_vertex <Num. of properties>	Max. number of sublevel labels for properties to print in the vertex labels	3
--use_rdf_label	RDF labels ar used as names for the output graph	false

Table 6.4: Ontology Output Options

Option	Description	Default
--export_ontologies= <Num. of properties>	The tool will create ".owl" files for the parts of the $\mathcal{E}$-partition	false
--ontology_output_path= <Num. of properties>	Sets the path of the ontology output directory, to which the ".owl" files are saved	Location of the .jar

Table 6.5: Role Vertices Designators

Option	Description	Default
--property_0_designator= <Designator>	Set the designator for R_0	[0]
--property_1_designator= <Designator>	Set the designator for R_1	[1]

438 ontologies (Matentzoglu and Parsia, 2017) include [0] and [1] as suffixes. The user can customize these designators if [0] and [1] appear as suffixes in their ontology (see Table 6.5).

Another group of options allows the user to switch the various heuristics on and off. In addition, various adjustments can be made for the different heuristics (see Table 6.6). A more detailed description of the heuristics can be found in Chapter 5. By default, all heuristics are deactivated.

The last group of options allows the user to obtain a partitioning even if the ontology is not safe or contains axioms with the universal role (see Table 6.7). It is important to note that the resulting partitions will not provide the discussed logical guarantees.

6.4 Testing

To ensure the correctness of our implementation, we implemented extensive unit tests. The main goal of the tests is to test if the implementation of the algorithm, given an axiom or subconcept, connects the vertices in the graph correctly.

We implemented 117 unit tests spanning connection tests for all logical axioms of the OWL 2 API [4] and special case tests for Top and Bottom as inputs. All tests are successful for the final version of the implementation. An extensive description and a protocol of the test results can be found in Section A of the appendix.

The unit tests helped us in finding and repairing five incorrectly implemented axiom and subconcept cases in the implementation phase.

The heuristics, the graph exporter, and the general usage of the implementation were tested by simple system testing, by using the implementation and by manually comparing the results with the expected results. Some of the results can be seen in the graphs and evaluation results of chapter 7.

Non-logical and datatype definition axioms are not tested explicitly, because they will not add connections in the constraint graph and the algorithm of chapter 4.1 does not determine the component of the axiom. The only possible test would be to test that the axioms are preserved. These tests were done in manual system tests.

[4] Version 5.1.6; Syntax can be found at `https://www.w3.org/TR/owl2-syntax/`

Table 6.6: Heuristics Options

Option	Description	Default
--use_iph	Use the “Ignore Properties Heuristic (IPH)”	false
--global_properties= <List of properties, e.g. [Prop1,Prop2,Prop3,...]>	Set of global properties	Empty set
--domain_global_properties= <List of properties, e.g. [Prop1,Prop2,Prop3,...]>	Set of domainglobal properties	Empty set
--range_global_properties= <List of properties, e.g. [Prop1,Prop2,Prop3,...]>	Set of rangeglobal properties	Empty set
--use_olh	Use the “Ontology Level Reducer Heuristic (OLH)”	false
--olh_layers_to_remove= <Num. of layers>	Set the number of layers that are removed by the OLH	1
--use_olh_after	Alternative or additional procedure to “Ontology Level Reducer Heuristic (OLH)”. Uses OLH repeatedly on the largest component.	false
--olh_after_repetitions= <Num. of repetitions>	Set the number of repetitions	1
--olh_after_treshhold= <Treshhold>	Treshhold of size of biggest component. If no partition of this size exist, the heuristic is not repeated	0.9
--use_bh	Use the “Biconnectivity Heuristic (BH)”	false
--bh_number_of_axiom_labels= <Number of axioms>	An edge may not be removed by the heuristic if removing would remove more than the given number of axioms.	1
--bh_number_of_repetitions_of_heur <Number of repetitions >	Number of repetitions of the BH	1
--use_cd	Use the “Community Detection Heuristic (CD)”	false
--cd_leiden	Use the Leiden instead of the Louvain Community Detection Heuristic	true
--use_ulh	Use the “Upper level remover heuristic (ULH)”	false
--upper_level_file= <Path to upper level file>	Sets the path of the upper level file	Location of the .jar
--ulh_removal_treshhold= <Treshold>	If the ontology contains more than the treshold of one of the given upper-level ontologies, it is removed	0.9

Table 6.7: Safety and Universal Role Options

Option	Description	Default
--handle_universal_roles	Allow a limited occurrence of axioms which contain the universal role. The affected axioms are removed.	false
--universal_role_treshhold= <Number of axioms>	If "handle_universal_roles" is set: Max. number of axioms that will be allowed, before no further handling is done.	3

Chapter 7

Evaluation

In this chapter we will evaluate the implemented EPartitioner. First, we will discuss the result of our implementation using three specific ontologies.

We will examine the decomposition of two smaller ontologies, namely the Koala[1] and Pizza[2] ontology. We will then evaluate our tool on a larger ontology, namely SNOMED. SNOMED is a medical ontology with over 100.000 logical axioms. We will compare the results of two versions of SNOMED from 2010 and 2020 and discuss the results of an expert interview about the decomposition of SNOMED.

After this, we will evaluate the runtime of EPartitioner over the BioPortal Snapshot by Matentzoglu and Parsia (2017). Wir werden den Korpus anpassen, indem wir die grte Ontologie, namentlich die Proteasix Ontologie [3], entfernen, da diese bisher zu Problemen in unseren GraphExporter fhrt. The resulting decompositions will be compared with the results if EPartitioner uses heuristics.

7.1 Case Analysis

First we want to discuss individual decompositions of ontologies. Due to their simpler structure, we will first partition Koala and Pizza for this. Then we will discuss the partitioning of a larger ontology, namely SNOMED CT. For SNOMED we have also conducted an expert interview.

7.1.1 Koala

The Koala Ontology is a small toy ontology that is often used in tutorials. Due to its size it even allows a representation of the constraints graph. The graph 7.1 represents the partitioning of Koala. On page 118 the constraint graph can be found.

Vescovo et al. (2019) have in their work created a customized version of Koala whose partitioning gives a more interesting partitioning. Our partitioning in the graph 7.2 corresponds to the partitioning expected by Vescovo et al. (2019). Also for this version we specify the constraint graph on page 119.

[1] `https://protege.stanford.edu/ontologies/koala.owl`
[2] `https://protege.stanford.edu/ontologies/pizza/pizza.owl`
[3] `https://bioportal.bioontology.org/ontologies/PXO`

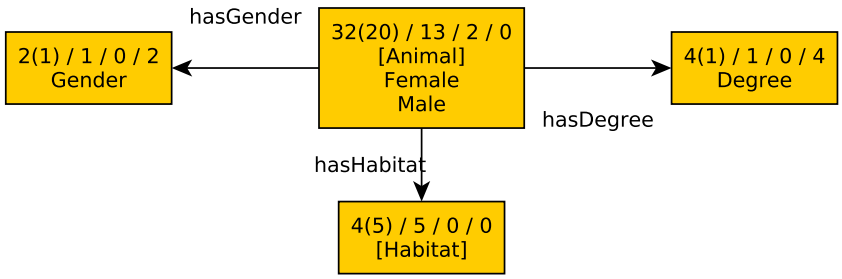

Fig. 7.1: Partitioning of the Koala Ontology

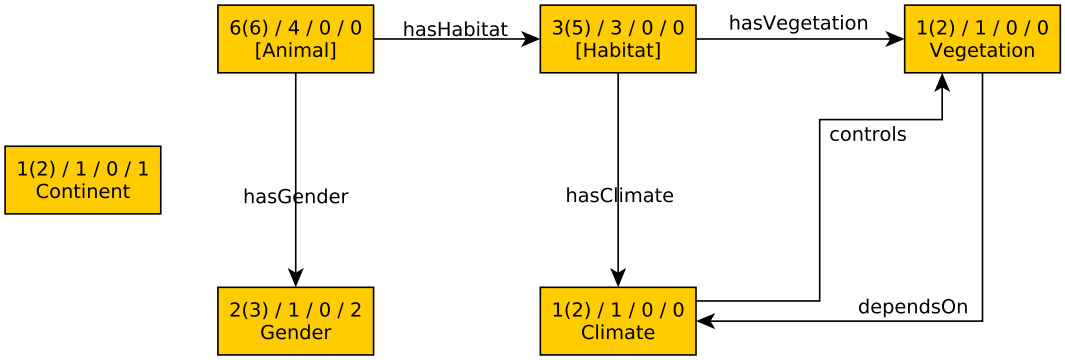

Fig. 7.2: Partitioning of the modified Koala Ontology

7.1.2 Pizza

The Pizza Ontology [4] is a small ontology used for a well-known Protege tutorial [5]. The partitioning of the pizza ontology by EPartitioner without heuristics results in a component that contains the complete ontology.

For this reason we tried partitioning with the ”Ontology Level Reducer Heuristic”, whereby we initially only removed one level. The result is shown in the graph 7.3. The partitioning results in three parts, which represent a reasonable division of content. It is noticeable that one component contains almost all logical axioms. Only 4 of the 322 logical axioms are removed from the heuristics.

We have also tested the ”Ontology Level Reducer Heuristic” with more than one level to be removed. Two removed levels do not change the partitioning. Three removed levels result in a further part in partitioning that only contains the classes ”NonVegetarianPizza” and ”VegetarianPizza” and is not connected to any other component. Additionally, 46 of the 322 logical axioms are removed. We have tested further increasing the levels to be removed, but up to 7 levels the partitioning has not changed any further, although only 170 of the 322 logical axioms have been removed.

[4] https://protege.stanford.edu/ontologies/pizza/pizza.owl
[5] https://protegewiki.stanford.edu/wiki/Protege4Pizzas10Minutes

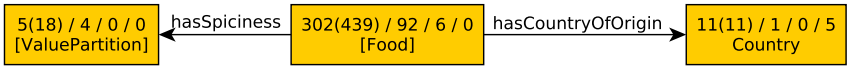

Fig. 7.3: Partitioning of the Pizza Ontolog with 1 Level removed

We have also used the ”Biconnectivity Heuristic” for partitioning the pizza ontology. Although the result was 28 parts, 25 of them contained only one class. The remaining three were structured in the same way as the partitioning in Graph 7.3. The disadvantage is that 87 of the 322 logical axioms were removed from the heuristics.

The ”Community Detection Heuristic” also results in a partitioning similar in content to the graph 7.3 (this time without parts containing only one class), with only 34 axioms removed.

This pattern, that heuristics usually lead to a similar decomposition of ontologies, we have also observed in further experiments. This is because the same nodes are identified and removed by the heuristics. A possible explanation is that especially the top-level concepts usually lead to the connectivity of the subgraphs.

7.1.3 SNOMED

7.1.3.1 2010's Version

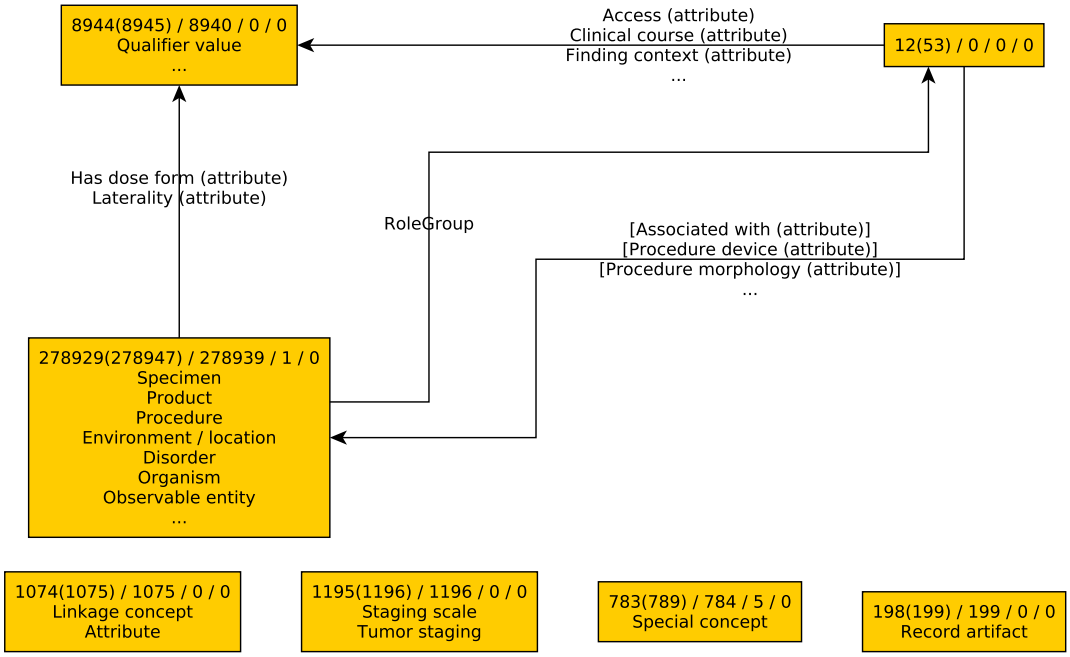

Fig. 7.4: Partitioning of SNOMED from 2010 with Top-Level removed

The partitioning of SNOMED from 2010 with the top levels removed is shown in

the graph 7.4. It shows a rough partitioning with three connected and four unconnected parts. The largest component contains over 95% of all logical axioms of the partitioned ontology. An interesting vertex is located in the upper right corner. It contains only 12 axioms but no classes, properties or individuals. This is achieved by using the RoleGroup property, which we will discuss further in the next chapter.

In contrast to the partitioning of SNOMED in 2020 (see graph 7.5), the largest component is separated from the component of the topic “qualifier value” and the component that is important for RoleGroup on the top right.

7.1.3.2 2020’s Version

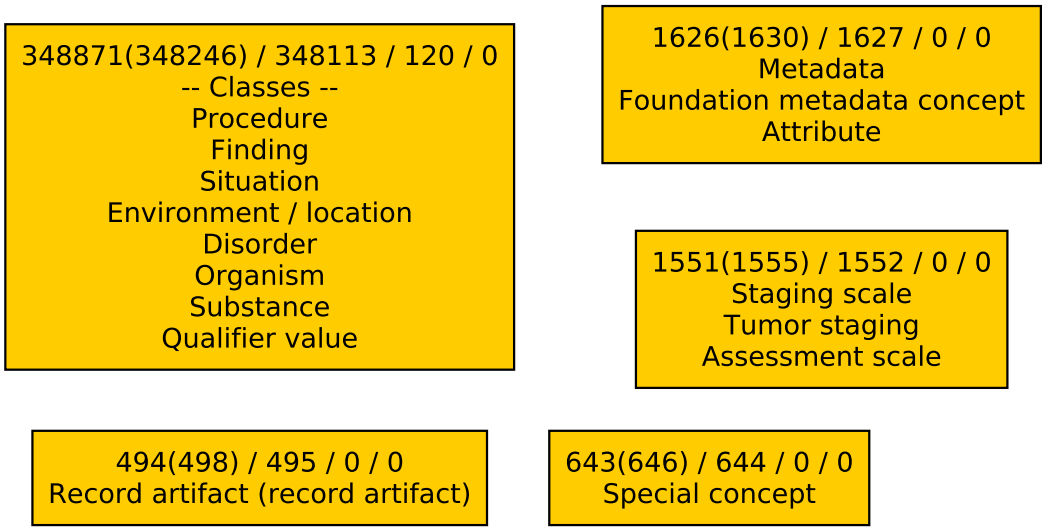

Fig. 7.5: Partitioning of SNOMED from 2020 with Top-Level removed

7.1.3.3 Expert Interview

On February 5, 2020, we interviewed an Expert on SNOMED CT. The expert in question was Yongsheng Gao, a senior terminologist of the International Health Terminology Standards Development Organisation (IHTSDO), which develops and distributes SNOMED CT. We gave him the partitioning discussed in the previous chapters.

As expected, the expert noted the coarse partitioning. We discussed which properties of the ontology could result in this ”coarse” partitioning. The first problematic property we discussed is the usage of the top-level concept with the label ”SNOMED CT Concept (SNOMED RT+CTV3)”. However, this property was not relevant in the case of the existing partitioning, since we already removed this top-level concept.

During the interview, we also identified that properties that are used in several different ”topics” could be another reason for ”coarse” partitioning. For example, the property

”Associated with (attribute)” is used in the axioms with ”Organism (organism)” and ”Disease (disorder)” as a range. Hence our algorithm would create a connection of ”Associated with (attribute)1”, ”Organism (organism),” and ”Disease (disorder)”. The same problem will occur if concepts are used in the domain of these properties.

Widespread use of such properties will result in strong connectivity between the ”topics”. This could be a reason for the coarse partition of SNOMED CT. We differentiate between properties that are global in their domain, their range, or in both areas. We will call these groups of properties ”global”, ”rangeglobal” or ”domainglobal”. We will use ”global” to refer to all properties that belong to one of these groups.

An example of a domainglobal property is the property ”Role Group”. ”Role Group” represents an ”association between a set of attribute value pairs that causes them to be considered together within a concept definition or postcoordinated expression” [6]. The domain of “Role Group” can be almost any topic in the ontology. Therefore “Role Group” is domainglobal.

After this interview, we proposed the heuristic that allows defining sets of global, rangeglobal, and domainglobal properties that will be ignored during the partitioning algorithm. This will result in a “finer” partition but also results in a partition without the known strong-logical guarantees.

In an experiment, we tried to identify global properties for SNOMED CT. This was done by finding the shortest paths between two concepts that seemed to belong to different “topics” and identifying property vertices in this chain that seemed promising. After this, we ran the algorithm on SNOMED CT with the heuristic and the identified property in their corresponding set of global properties. The property will remain in the corresponding set if this run results in a finer partition. If the run did not result in a finer partitioning, we identify a new shortest path, and we will look for further suspicious property vertices on the path. We repeated this step until the partitioning became finer, or there was no property vertex on the shortest path.

The result of this experiment is a partition with 18 components (see the simplified Graph 7.6). The partition is generally finer than the previous partition: The biggest component does only contain 117.628 logical axioms. The topic of this part is disorders, events, and clinical findings. The topics of other bigger parts are Organism (34947 logical axioms), observable entities, procedures and regimes/therapies (67297 logical axioms), body structures (39233 logical axioms), and physical objects and products (38469 logical axioms). Ten parts are connected with several link relations, e.g., the “Substance” is connected to the “Qualifier Value” part via the link relation “Has disposition (attribute)”. “Role Group” creates a central component for attributes. The other eight parts are unconnected, for example, the “organism” part.

We sent the new partition to the expert. His evaluation of the new partition was much more positive than the original partition. The themes of the individual components seem to correspond to real sub-themes of the ontology. Further comments of the expert only referred to basic questions or misunderstandings about partitioning using $\mathcal{E}$-connections.

At the time of publication the discussion with the expert is still ongoing.

[6] https://confluence.ihtsdotools.org/display/DOCGLOSS/attribute%20group

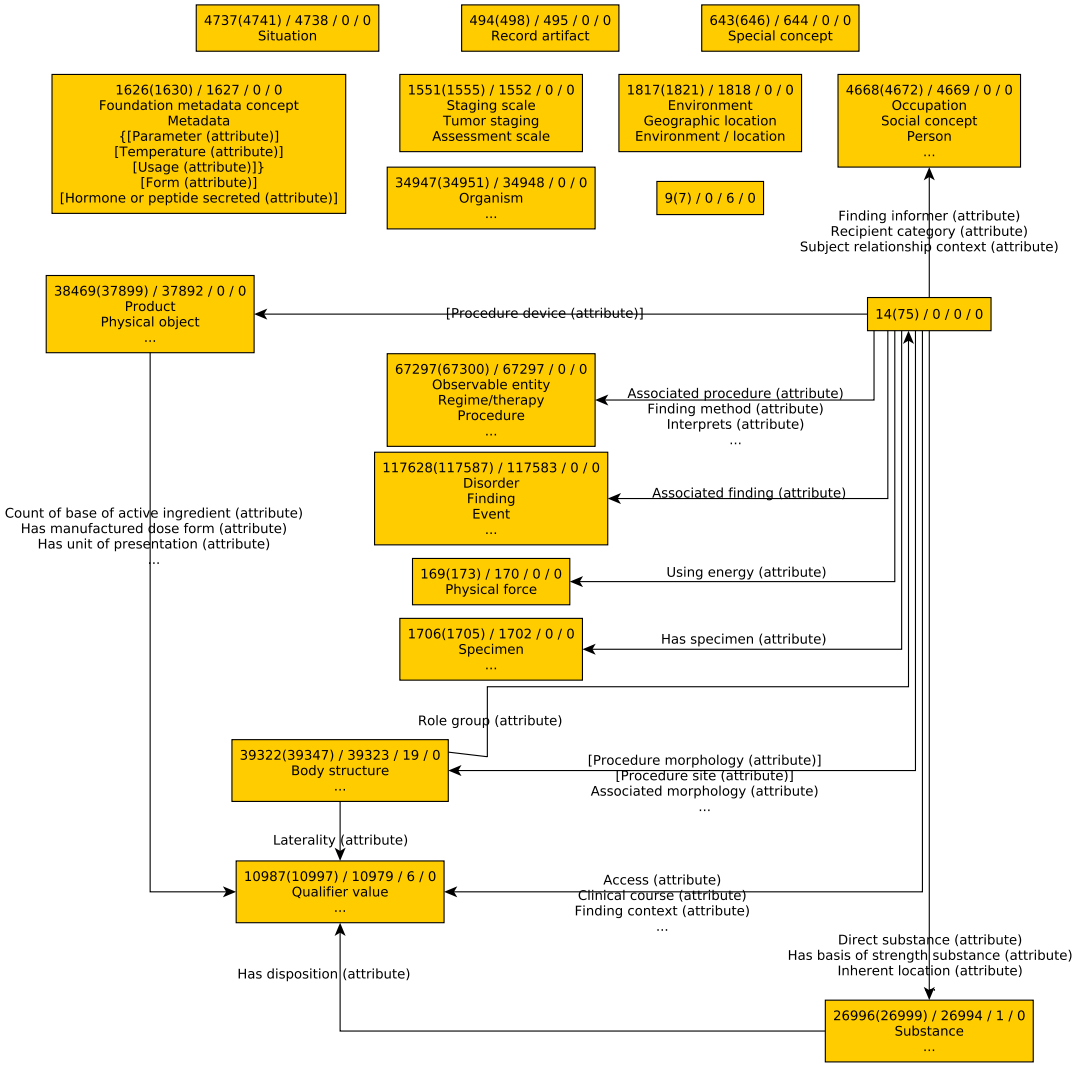

Fig. 7.6: Decomposition of SNOMED after IGRH

7.2 Decomposability

We analyzed how many of the ontologies of our BioPortal Snapshot by Matentzoglu and Parsia (2017) are decomposable.

The result is, that 8 of the 438 ontologies are not safe. Additionally 11 ontologies contain SWRL axioms, which are unsupported in our current implementation. Furthermore, 70 Ontologies contain axioms with the universal role that cannot be handled by E-Connection. Overall 84 of the 438 ontologies are not decomposable because they are not safe or contain the universal role. If we allow ontologies with at most 3 axioms with the universal role (the heuristic will remove these axioms), then the implementation can handle all but 30 ontologies. In the following sections we will evaluate the remaining 408 ontologies.

7.3 Statistical Analysis

7.3.1 BioPortal without Heuristics

First we will discuss the quality of the partitioning of EPartitioner. On average, the ontologies were partitioned into 5.7 parts. On the negative side, the average size of the largest component of the partitioning is 91.3%. The standard deviation is 18.9% and the minimum percentage size of the largest component is 10%. Only in 83 of the 408 partitionings was the largest component less than 90% of the logical axioms. 43 partitions had a major portion that contained less than two-thirds of the logical axioms of the ontologies. 176 ontolgies were not partitioned by EPartitioner, so the result was only one part containing all logical axioms. This indicates that the E partitioner leads to weak partitioning on natural ontologies. To counteract this, we have developed several heuristics that we discuss in chapter 5 and evaluate in chapter 7.3.2.

Now we want to evaluate the runtime of the implementation. We consider the implementation of the core algorithm we discussed in chapter 4 and the implementation including the graph exporter. The result of the evaluation of our runtime is shown in the graph 7.7. The graph also contains a polynomial fit.

The graph and the polynomial fit indicate a linear runtime of the core algorithm. This is particularly indicated by the negligible value of a (assumed $y = ax^2 + bx + c$).The total runtime of EPartitioner (including the runtime of the Graph Exporter) increases faster than that of the core algorithm, but also seems to be linear (with the same argumentation).

7.3.2 BioPortal with Heuristics

In this chapter we want to evaluate some of the implemented heuristics with our BioPortal Snapshot by Matentzoglu and Parsia (2017).

7.3.3 Ontology Level Reducer Heuristics (OLH)

The “Ontology Level Reducer Heuristics” was described in chapter 5.1. In this section we want to evaluate this heuristic. For this we use the BioPortal Snapshot (Matentzoglu and Parsia, 2017). For the evaluation we have removed one to seven levels each. The graph 7.8 represents the ratio of partitions where the largest component contains less than 90% of all axioms (called <90%-Partitions) to the portion of removed logical axioms.

The result shows a big problem of this heuristic: A large portion of all logical axioms are removed. Already when removing a one level, about 9.5% of all logical axioms are removed. Another removed level already results in the loss of 20.2% of all logical axioms. We stopped evaluating after seven removed levels, since 54.2% of all axioms were removed.

It could be argued that the high percentages are due to removing levels in small ontologies, since removing an axiom results in a higher percentage in a small ontology than in a larger ontology. To verify this we have studied the proportion of removed

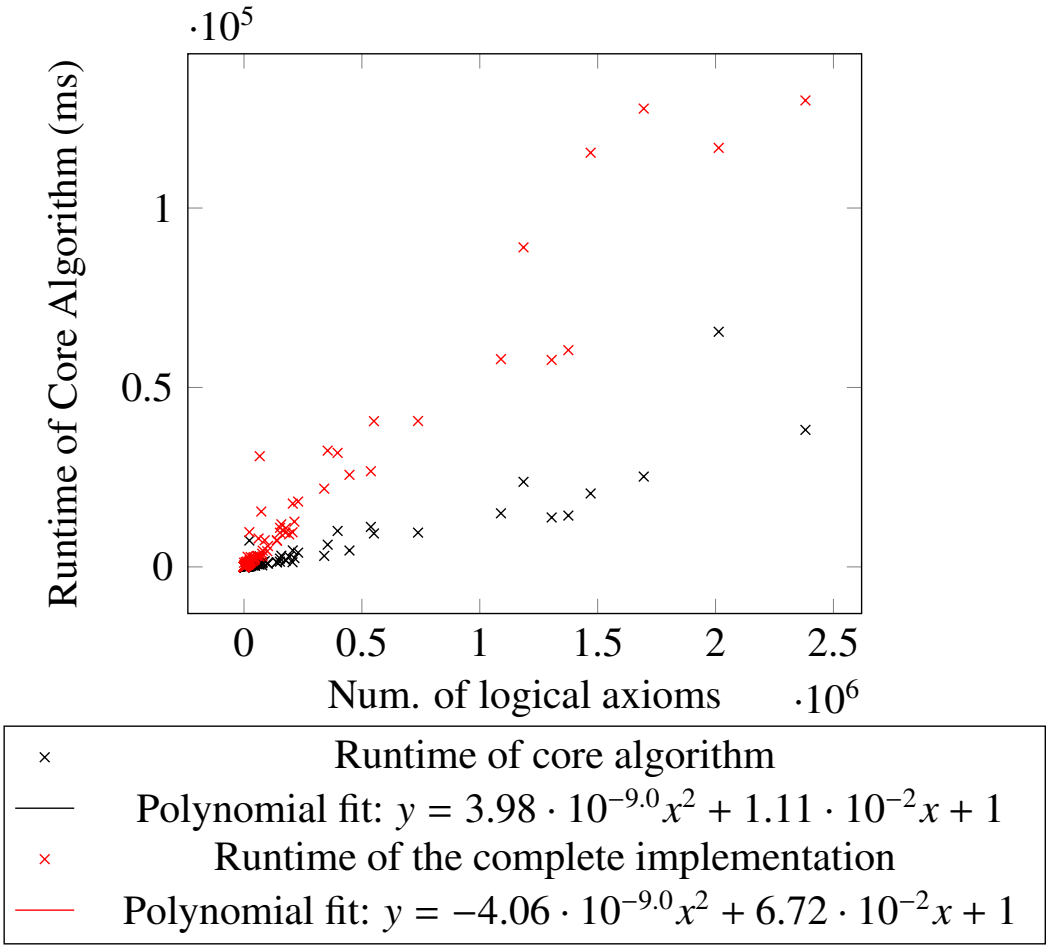

Fig. 7.7: Test Runtime of core algorithm to number of logical axioms

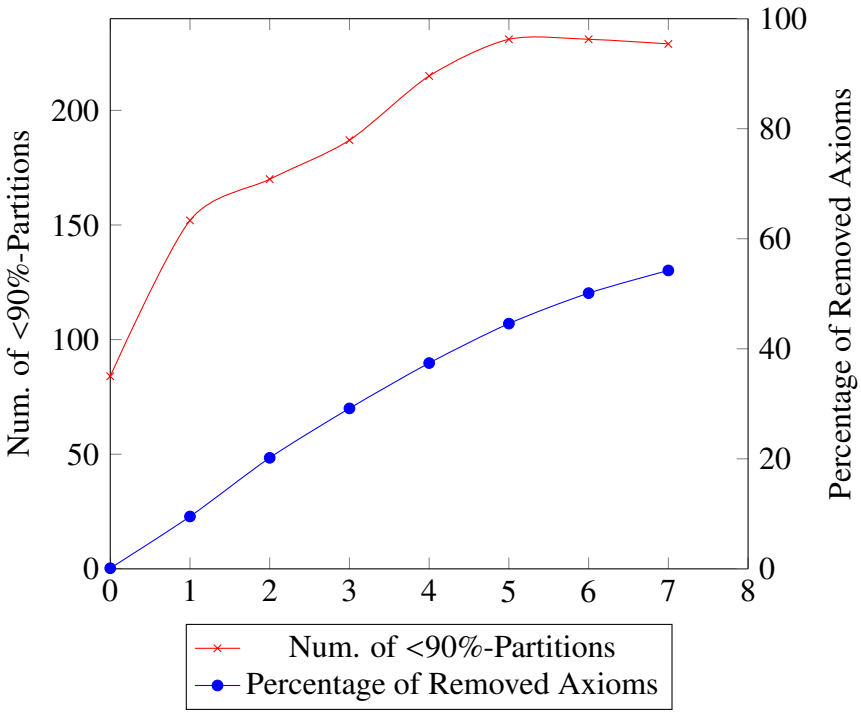

Fig. 7.8: Partitions with biggest Partition containing less than 90% of all axioms to percentage of removed axioms

axioms for all ontologies with more than 1, 100, 10000 and 50000 logical axioms. The results of the investigation are shown in the graph 7.9.

We can see that although the percentages decrease as the size of the ontologies increases, they do not decrease in proportion to the size of the ontology. The percentages for ontologies with more than 100 logical axioms is not one hundredth of the values for ontologies with more than 10000 logical axioms. It follows that the problem also applies to larger ontologies, where thousands of axioms are removed.

One ray of hope is that the number of axioms removed varies greatly. In table 7.1 we present the standard deviation (near the average) of the removed axioms in all ontologies. If we remove a level, 75 of the 354 ontologies with more than 100 logical axioms have only 5 or less logical axioms removed. 24 of the 75 ontologies have a largest component with less than 90% of all logical axioms. This data suggests that heuristics can be useful in some cases, but should be used with caution with regard to loss of knowledge.

7.3.4 OLH Alternative

We have discussed an alternative or supplementary heuristic to “Ontology Level Reducer Heuristic (OLH)” in chapter 5.1.1. The heuristic removes, after removing the first level, for each subsequent level, only the concepts of the largest component (if they have more than a certain percentage of the logical axioms of the original ontology). We call this heuristic “OLH After”.

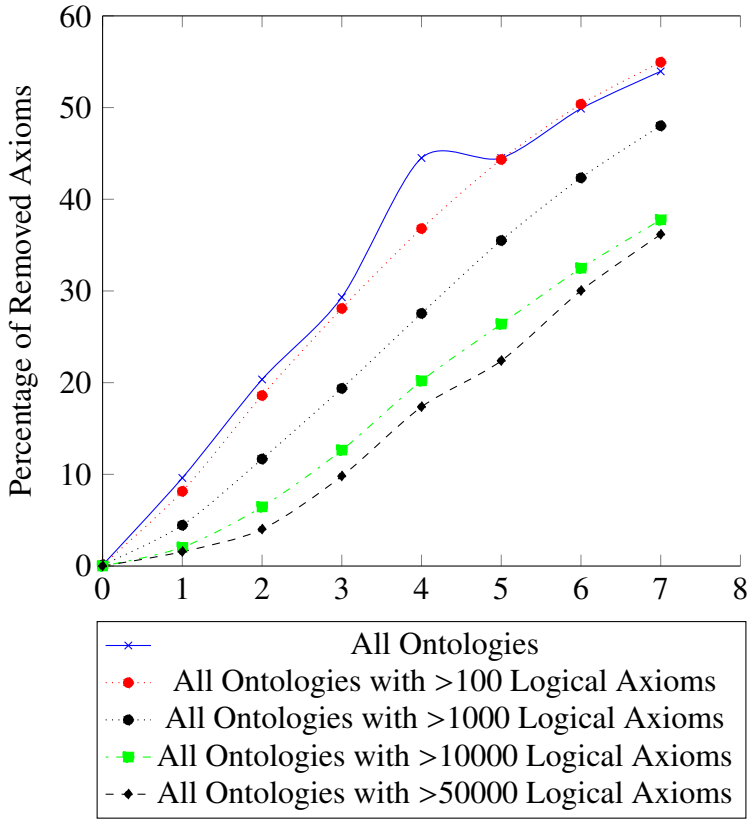

Fig. 7.9: Percentage of Removed Axioms

Furthermore, we will call the heuristic, which removes another level after the removal of the first level by OLH according to the principle of this alternative heuristic, from now on abbreviated OLHAfter1. The heuristic, which removes *two* levels after the first level after OLH according to the principle of this alternative heuristic, we will call OLHAfter2 accordingly, and so on. Accordingly to this notation, OLH1 refers to the removal of a level by the OLH, OLH2 refers to the removal of two levels by the OLH and so on. It is important to note that OLHAfter1 and OLH2, and OLHAfter2 and OLH3 are comparable since both have the same number of "levels" removed.

The result of the evaluation of OLHAfter1 and OLHAfter2, and for comparison, the results for OLH1, OLH2, and OLH3 can be found in table 7.2.

It can be seen that the new heuristic does not perform better than OLH in terms of the number of components and the size of the largest component. OHLAfter1 results in only 28.5 components and the largest component containing an average of 79.3% of all logical axioms. For comparison, OLH2 results in 37.7 components and the largest component containing an average of 79.3% of all logical axioms. OLHAfter1 also removes slightly more logical axioms than OLH2 (10.3% compared to 9.6%).

However, it also shows that the results of OLHAfter1 include more partitions with the largest component that contains an average of less than 90% (and also 66%) of all logical axioms than the results of OLH2. We can also observe that OLHAfter1 has a

Table 7.1: Number of removed axioms: Average and Std. Dev.

Number of removed Layers	Average	Std. Dev.
0	0.11441328	0.86356016
1	9.61155006	17.04151062
2	20.34710726	26.23971333
3	29.31576065	30.28819719
4	37.4556476	32.50159397
5	44.49927078	33.33053132
6	49.88597421	34.37058167
7	53.94784691	34.55769665

Table 7.2: Results of evaluation for OHLAfter1, OHLAfter2, OHL1, OHL2 and OHL3

Heuristic	#Partitions	Avg. % of biggest Part.	<90%	<66%	% of removed Axioms
OLH1	16.52088	81.92243	83	43	0.11441
OLHAfter1	28.52141	79.31848	174	102	10.27262
OLH2	37.66584	79.34570	149	95	9.61155
OLHAfter2	37.22979	78.09416	197	103	10.42340
OLH3	61.58230	76.49270	167	114	20.34710

high standard deviation of 17.1% in the percentage of axioms removed. This is also shown by the fact that less than five logical axioms were removed from 101 ontologies, which is significantly more than the 29 ontologies for OLH2 .

The same observations of the relationship between OLHAfter1 and OLH2 can be seen in the relationship between OLHAfter2 to OLH3, with the exception that the results of OLH3 show a much higher loss of axioms, but the partitioning has more and more components. The results of the alternative heuristics are, therefore, relatively promising. This is particularly evident in the high number of ontologies from which only five axioms were removed. Nevertheless, similar to OLH, a high loss of axioms is generally observed.

7.3.5 Biconnectivity Heuristic (BH)

For the Biconnectivity and Community Detection heuristics, we found that they do not have a linear increase in runtime. The runtime of the heuristic can increase to over one hour for the larger ontologies. The reason for this is a method that both heuristics use. This method is used to remove so-called bridges. To prevent the creation of singleton components (components with only one axiom) the connected components are recalculated after each removal of a bridge. If the connected components contain a new singleton component, the removal is undone. Since the heuristics identify many bridges,

this can lead to a high runtime. For this reason, we will use only the smallest 380 of the 437 ontologies from our BioPortal Snapshot by Matentzoglu and Parsia (2017) for the following evaluations. 347 of these 380 ontologies are safe to contain axioms with the universal role or SWRL axioms.

The results of the evaluation are as follows: On average, the ontologies are divided into partitions of 940 components. Comparable is 14.4 components for OLH1 and 22.1 components for OLH2. This shows us an apparently very fine partitioning.

The largest component contains, on average, 77.1% of all logical axioms of the input ontology. For comparison, the largest component, OLH1, contains 80.6% of all logical axioms. For OLH2, this is 77.8%.

One problem seems to be the loss of logical axioms: On average, 63.5% of all logical axioms have been removed. Even in ontologies with more than 10000 logical axioms, 41.6% of all logical axioms were removed. In comparison, OLH2 removes "only" 21.9% of all logical axioms and even only 3.2% of all logical axioms in the ontologies with more than 10,000 axioms. Similar to OLH, the standard deviation of the average removed axioms is relatively high at 35%. Unfortunately, it does not show, as in OLH, that there is a larger proportion of ontologies where only a few axioms have been removed. Only in 18 ontologies, less than five logical axioms were removed. In comparison, these are 84 ontologies for OLH1 and 22 ontologies for OLH2.

It can be seen that BH leads to fine partitioning. It is questionable whether partitioning with hundreds of components is useful for the user. This could be investigated in future work by user studies. However, compared to these fine partitionings, there is often the problem that the largest component contains the majority of all logical axioms. Moreover, this heuristic removes a high number of logical axioms. OLH1 and OLH2 seem to be equivalent or advantageous in all the metrics discussed, especially when user studies show that partitioning with hundreds of components is not interesting for users.

7.3.6 Community Detection Heuristic (CD)

The results of the evaluation of the Community Detection Heuristic (CD) are very similar to the results of the evaluation of the BH. On average, the partitionings consist of a high number of components: 506.5. The largest component contains, on average, 76.2% of all logical axioms. On average, 47.6% of all axioms are removed (33.9% if we restrict ourselves to ontologies with more than 10000 logical axioms). Slightly better than for BH is that from 30 instead of 18 of all ontologies at most five axioms were removed. Overall, the conclusion for the Community Detection Heuristic is the same as for BH.

7.3.7 Summary

Overall, the algorithm works fast without heuristics but leads to very rough partitioning. In contrast, the heuristics often help to refine the partitions but result in a loss of knowledge. In future work, it would be useful to find out in user studies how fine the partitioning has to be in order to be useful for the user. In addition, the question if the loss of knowledge (and to which extent) is acceptable should be investigated.

Chapter 8

$\mathcal{E}$-connection for other First-Order Logic Fragments

First-order logic (FOL) over relational signatures without equality is defined in the standard way. For every $n \geq 1$, we fix a countably infinite set of *relation symbols* of *arity n*, denoted by $R, S, \ldots$. We also fix a countably infinite set of *variables*, denoted $x, y, \ldots$. *Formulas* are defined inductively by

$$\varphi \ ::= \ R(x_1, \ldots, x_n) \mid \neg\varphi \mid \psi \wedge \varphi \mid \exists x\, \varphi,$$

with R an n-ary relation symbol and $n \geq 1$. The arity of a relation symbol R is denoted $\mathsf{arity}(R)$. The remaining operators $\vee, \rightarrow, \Leftrightarrow, \forall$ are "syntactic sugar", hence they can be expressed using the operators from above. A variable x is *bound* if it occurs in a subformula of the form $\exists x.\psi$, otherwise x is free. We write $\varphi(\mathbf{x})$ with $\mathbf{x} = (x_1, \ldots, x_n)$, $n \geq 0$, if the free relation variables of φ are exactly the x_i. A *sentence* is a formula without free variables. We define $\mathrm{FO}^{\wedge,\exists}$ as the set of all FOL formulas with only atoms, $\wedge$ and $\exists$.

The semantics of FOL formulas is defined in the standard way, too, via satisfaction in *interpretations* $(\mathfrak{A}, \beta)$ that consist of a *structure* $\mathfrak{A} = (A, \cdot^{\mathfrak{A}})$ where A is the *domain* and $\cdot^{\mathfrak{A}}$ an *interpretation function* that assigns to every n-ary relation symbol R a relation $R^{\mathfrak{A}} \subseteq A^n$, and a *valuation* β that maps every variable to a domain element. We write $\beta[x/a]$ to update β by fixing $\beta(x) = a$. For a formula $\varphi(x_1, \ldots, x_n)$ we write $\varphi[a_1, \ldots, a_n]$ or $\varphi(x_1, \ldots, x_n)[a_1, \ldots, a_n]$ to fix $\beta(x_i) = a_i$ for $i \leq n$.

Definition 1. The satisfaction of φ in $(\mathfrak{A}, \beta)$, written $\mathfrak{A}, \beta \models \varphi$, is inductively defined as follows:

- $\mathfrak{A}, \beta \models R(t_1, \ldots, t_n)$ iff $(\beta(t_1), \ldots, \beta(t_n)) \in R^{\mathfrak{A}}$
- $\mathfrak{A}, \beta \models \neg\varphi$ iff $\mathfrak{A}, \beta \not\models \varphi$
- $\mathfrak{A}, \beta \models \varphi \wedge \psi$ iff $\mathfrak{A}, \beta \models \varphi$ and $\mathfrak{A}, \beta \models \psi$
- $\mathfrak{A}, \beta \models \exists x\, \varphi$ iff there is an $a \in A$ with $\mathfrak{A}, \beta[x/a] \models \varphi$

In the previous chapters, we discussed partitionings that are based on $\mathcal{E}$-connections, which in turn are originally based on abstract description systems (ADSs) to abstract

away from a particular DL or modal logic or related formalism. The "point of entry" for establishing an $\mathcal{E}$-connection from a monolithic (DL) ontology is the partitioning of its signature (concepts, abstract roles, concrete roles, individual names) such that each term can be assigned to a specific part of the $\mathcal{E}$-connection. For this purpose, every DL concept, concrete role and individual name is assigned a single number $i \in \mathbb{N}$ that represents that term's "home part"; abstract role names r are assigned pairs (i, j) of numbers – if $i = j$, then r has "home part" i; otherwise r links parts i, j.

In FOL, we only have n-ary relation symbols; there are no explicit distinction of concept, abstract, or concrete role names. Now every n-ary relation symbol must be assigned an n-tuple of indices. We call this an *indexing*. In chapter 3.1, the indexing ν is defined as a mapping of each concept, individual name, and concrete role name to an index i. Unary relation symbols behave like DL concepts in the original approach; their "home part" is given by the assigned singleton tuple. Symbols of arity ≥ 2 generalize the behavior of abstract roles: they have a "home part" if all numbers in the n-tuple are equal or otherwise link n parts. Individual names (in FOL: constants) no longer require separate consideration: they automatically get a "home part" when expressed using unary relation symbols and equality.

In FOL, we have an initial I-indexing ν that assigns every n-ary relation symbol an n-tuple of numbers. This notion is extended by variable numberings for formulas, denoted by α_φ (for a formula φ). Since existential restrictions are more general, the assignment of indices and "permitted indexings" have to be obtained from the occurrences of variables – e.g., if ν assigns (i, j, k) to R and (ℓ, m) to S, then α_φ is a valid variable indexing for $\varphi(x, y) = \exists z\, R(x, y, z) \wedge S(y, z)$ only if $j = \ell$. An axiom is a first-order formula of the form $\forall x\, \psi(x)$, and will be indexed with the index assigned to x. These ideas are captured by the notion of a *variable indexing* of the free variables in a formula:

Definition 2. Let Σ be a signature, I a set of indices and ν an I-indexing for Σ. ν inductively induces a variable indexing α_φ for an arbitrary FOL formula φ:

1. If $\varphi(x_1, \ldots, x_n) = R(x_1, \ldots, x_n)$ with $\nu(R) = (i_1, \ldots, i_n)$, then $\alpha_\varphi(x_j) = i_j$ for all $j \leq n$.
2. If $\varphi(x_1, \ldots, x_n) = \neg\psi(x_1, \ldots, x_n)$, then $\alpha_\varphi(x_i) = \alpha_\psi(x_i)$ for all $i \leq n$.
3. If $\varphi(x_1, \ldots, x_n) = \psi(y_1, \ldots, y_m) \wedge \vartheta(z_1, \ldots, z_k)$, then
   - if $x_i = y_j$ then $\alpha_\varphi(x_i) = \alpha_\psi(y_j)$ for all $i \leq n$, $j \leq m$
   - if $x_i = z_j$ then $\alpha_\varphi(x_i) = \alpha_\psi(z_j)$ for all $i \leq n$, $j \leq k$.
4. If $\varphi(x_1, \ldots, x_n) = \exists x\, \psi(x, x_1, \ldots, x_n)$, then $\alpha_\varphi(x_i) = \alpha_\psi(x_i)$ for all $i \leq n$.

The order of free variables in the first-order formulas of Definition 2 is arbitrary. It is easy to see that there is at most one variable indexing, for each ν and φ. Here it becomes clear why we omit equality in the discussed fragment of FOL: General usage of equality would allow for formulas of the form $\varphi = \forall x, y.x = y$. φ contains no relation variable, and therefore no unambiguous variable indexing α_φ can be defined. It is important to

note that some forms of formulas with equality can still induce a variable indexing. An example would be the formula $\omega = \forall x, y, y'.(R(x, y) \wedge R(x, y') \rightarrow y = y'$ which expresses functionality of R. Each variable in ω has a unique I-indexing, which will induce variable indexing.

Variable indexings are extended to axioms $\varphi = \forall x\, \psi(x)$: we call φ an *i-axiom* given ν if there is an variable indexing α_ψ for ψ given ν such that $\alpha_\psi(x) = i$. The variable indexing α_φ is defined for a formula $\varphi(x_1, \ldots, x_n)$, if the indexing is valid according to 1-4.

Example 3. Fix an I so that $i_1, i_2 \in I$ with $i_1 \neq i_2$. The variable indexing α_φ for the formula $\varphi(x_1, x_2, x_3) = R(x_1, x_2) \wedge R(x_1, x_3)$ is defined for the I-indexing ν with $\nu(R) = (i_1, i_2)$. It is easy to see that the only defined α_φ is $\alpha_\varphi(x_1) = i_1$ and $\alpha_\varphi(x_2) = \alpha_\varphi(x_3) = i_2$. An example for a formula with no possible α_ψ is $\psi(x_1, x_2, x_3) = R(x_1, x_2) \wedge \neg R(x_3, x_1)$. x_1 occurs on both positions of R in ψ, therefore ψ would induce an α_ψ with $\alpha_\psi(x_1) = i_1$ and $\alpha_\psi(x_1) = i_2$, but $i_1 \neq i_2$.

ν-ontologies are defined in the same way as in Chapter 3.1:

Definition 4. Given an I-indexing ν, a *ν-ontology* is a set $\mathbb{O} = \{\mathcal{O}_i \mid i \in I\}$, each of whose *components* $\mathcal{O}_i$ is a nonempty set of i-axioms.

The interpretation of ν-ontologies is defined analogous to the standard FOL-interpretation metioned earlier, with the exception of the $\neg$-case:

Definition 5. A *ν-structure* $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, R_2^{\mathfrak{A}}, \ldots)$ is a structure, s.t. $A = \biguplus_{i \in I} A_i$ and $R_\ell^{\mathfrak{A}} \subseteq A_{i_1} \times \cdots \times A_{i_n}$ for every n-ary relation symbol R_ℓ with $\ell \leq k$ and $i_1, \ldots, i_n \in I$.

$(\mathfrak{A}, \beta)$ is a *ν-interpretation*, if $\mathfrak{A}$ is a ν-structure and β is a *valuation* that maps every variable to a domain element. The *satisfaction* relation between ν-interpretations $(\mathfrak{A}, \beta)$ and ν-indexed-formulas is defined as follows:

1. $\mathfrak{A}, \beta \models^\nu R(x_1, \ldots, x_m)$ if $(\beta(x_1), \ldots, \beta(x_m)) \in R^{\mathfrak{A}}$
2. $\mathfrak{A}, \beta \models^\nu \neg\varphi(x_1, \ldots, x_n)$ and β is such that $\beta(x_j) \in A_{\alpha_\varphi(x_j)}$ for all $j \leq n$.
3. $\mathfrak{A}, \beta \models^\nu \varphi \wedge \psi$ if $\mathfrak{A}, \beta \models^\nu \varphi$ and $\mathfrak{A}, \beta \models^\nu \psi$
4. $\mathfrak{A}, \beta \models^\nu \exists x\varphi$ if there is a $a \in A$ s.t. $\mathfrak{A}, \beta[x/a] \models^\nu \varphi$

Only the $\neg$-case needs to explicitly state $\beta(x_j) \in A_{\alpha_\varphi(x_j)}$ for all $j \leq n$. This property holds for all other cases, which can be proven via straightforward induction. This notion is captured in the following proposition:

Proposition 6. *For all $\mathfrak{A}$, β and $\varphi(x_1, \ldots, x_n)$ if $\mathfrak{A}, \beta \models^\nu \varphi(x_1, \ldots, x_n)$ then $\beta(x_j) \in A_{\alpha_\varphi(x_j)}$ for $j \leq n$.*

A ν-interpretation $(\mathfrak{A}, \beta)$ is a *model* of a ontology $\mathcal{O}_i$ if it satisfies all i-axioms in $\mathcal{O}_i$ and a model for a ν-ontology $\mathbb{O} = \{\mathcal{O}_i \mid i \in I\}$ if $(\mathfrak{A}, \beta) \models \mathcal{O}_i$ for every $i \leq n$. $\mathbb{O}$ is *consistent* if it has a model.

We capture the correspondence between simple and ν-ontologies by compatibility and equivalence as in Section 3.2:

Definition 7. Let $\mathcal{O}$ be an ontology, ν an I-indexing, and $\mathbb{O} = \{\mathcal{O}_i \mid i \in I\}$ a ν-ontology.

1. $\mathcal{O}$ and $\mathbb{O}$ are *compatible*, written $\mathcal{O} \sim \mathbb{O}$, if $\mathcal{O} = \bigcup_{i \leq n} \mathcal{O}_i$.

2. $\mathcal{O}$ and $\mathbb{O}$ are *equivalent*, written $\mathcal{O} \approx \mathbb{O}$ if, for all ν-interpretations $(\mathfrak{A}, \beta)$, it holds that $\mathfrak{A}, \beta \models \mathcal{O}$ iff $\mathfrak{A}, \beta \models^{\nu} \mathbb{O}$.

To establish a link between compatibility and equivalence as for description logic in Chapter 3.2 we first need to define domain-independence for first-order logic:

Definition 8. An FOL formula $\varphi(x_1, ..., x_n)$ is *domain-independent (DI)* if for all structures $\mathfrak{A} = (A, P_1^{\mathfrak{A}}, P_2^{\mathfrak{A}}, \ldots)$ and for all $B \supseteq A$ and for all $(a_1, ..., a_n) \in B^n$:

$$\mathfrak{A} \models \varphi[a_1, ..., a_n] \text{ iff } (B, P_1^A, P_2^A, ...) \models \varphi[a_1, ..., a_n]$$

Domain independence is a well-known notion, particularly important in database theory: the domain-independent fragment of FOL has the same expressivity as relational algebra, on which SQL is based, see (Abiteboul et al., 1995). While domain-independence is undecidable (Vardi, 1981), various sufficient syntactic conditions have been established, see (Kifer, 1988; Van Gelder and Topor, 1991).

In the context of this Thesis, we will limit ourselves to specific fragments of first-order logic. An obvious restriction for these fragments is that it cannot contain negation of relational axioms with arity ≥ 2. They represent a generalization of the discussed problems with the universal role (see page 23).

A fragment of first-order logic that complies with this restriction is a logic limited to tuple generating dependencies.

8.1 Tuple-generating Dependency

A *tuple generating dependency (TGD)* is a sentence in first order logic of the form:

$$\forall X, Y. P(X, Y) \rightarrow \exists Z. Q(X, Z)$$

where P is a possibly empty, X, Y, Z are pairwise disjoint sets of variables, and Q is a non-empty conjunction of relational atoms. In addition, it is easy to see that TGDs are always domain-independent.

TGD's were first introduced by Beeri and Vardi (1984) and provide the foundation for Datalog+/- (Calì et al., 2010). Several systems using TGD's (Bellomarini et al., 2018; Baget et al., 2015) and practical applications in bioinformatics and modeling complex structures of chemical compounds have been developed (Grau et al., 2013; Mungall, 2009; Magka et al., 2012).

8.1.1 Main Theorem

In this chapter, we want to transfer our notations and results for $\mathcal{SHOIQ}(D)$ (with Keys) to *TGDs*. We can write TGD's as FOL-sentences of the form:

$$\varphi = \forall x_1, \ldots, x_n(\psi(x_{j_1}, \ldots, x_{j_\ell}) \rightarrow \omega(x_{k_1}, \ldots, x_{k_1})$$

with ψ being a (possibly empty) conjunction of relational atoms and ω being an existential restriction over a conjunction of relational atoms (as $\exists Z.Q(X, Z)$).

This notation gives us a way to compare the expressivity of $\mathcal{SROIQ}$ and TGD's. It is easy to see, that TGD's can express $\mathcal{SROIQ}$ axioms of the form $C_1 \sqcap \ldots \sqcap C_n \sqsubseteq \exists r_1.D_1 \sqcap \ldots \sqcap \exists r_m.D_m$. The expressivity of $\mathcal{SROIQ}$ and TGD's are different, but none of them both is a subset of the other: On the one hand, TGD's cannot express all axioms in $\mathcal{SROIQ}$, e.g., $\exists r.D \sqsubseteq \neg C$. On the other hand, TGD's allow relational atoms with arity ≥ 2, which cannot be expressed in Description Logic.

Now we can transfer Theorem 13 to TGDs:

Theorem 9. *Let $\mathcal{O}$ be an ontology, ν an I-indexing, $\mathbb{O} = \{\mathcal{O}_i \mid i \in I\}$ a ν-ontology.*

1. *If $\mathcal{O} \sim \mathbb{O}$, then $\mathcal{O} \approx \mathbb{O}$.*
2. *If additionally $\mathcal{O}$ is consistent, then so is $\mathbb{O}$.*

The changes of the Theorem in comparison to Theorem 13 are due to the domain-independence of TGDs.

Proof.

Ad 1. Assume that $\mathcal{O} \sim \mathbb{O}$. Let $\mathfrak{A}, \beta$ be a ν-interpretation. We need to show: $\mathfrak{A}, \beta \models \mathcal{O}$ iff $\mathfrak{A}, \beta \models^\nu \mathbb{O}$. For this purpose, we first establish the following properties:

Claim 1. *$\mathfrak{A}, \beta \models \varphi(x_1, \ldots, x_n)$ iff $\mathfrak{A}, \beta \models^\nu \varphi(x_1, \ldots, x_n)$ for all $\varphi \in FO^{\wedge,\exists}$,.*

Proof of Claim 1.

By induction over the structure of φ. The base case holds by Definition 5 and Definition 1. All other cases can be shown by Definition 5, Definition 1, and the induction hypothesis.

To prove $\mathfrak{A}, \beta \models \mathcal{O}$ iff $\mathfrak{A}, \beta \models^\nu \mathbb{O}$ it suffices to show $\mathfrak{A} \models \varphi$ iff $\mathfrak{A} \models^\nu \varphi$ for every $\varphi \in \mathcal{O}$. To prove this, we establish the following property:

Claim 2. *For all $\mathfrak{A}$, for all TGDs $\varphi = \forall x_1, \ldots, x_n(\psi(x_{j_1}, \ldots, x_{j_\ell}) \rightarrow \omega(x_{k_1}, \ldots, x_{k_m}))$ with $\{j_1, \ldots, j_\ell, k_1, \ldots, k_m\} = \{1, \ldots, n\}$: $\mathfrak{A} \models \varphi$ iff $\mathfrak{A} \models^\nu \varphi$.*

Proof of Claim 2.

We can omit β from the interpretation, because $\mathcal{O}$ and $\mathbb{O}$ are set of sentences. Therefore β is not necessary to interpret the formulas in $\mathcal{O}$ and $\mathbb{O}$ (and consequently $\varphi \in \mathcal{O}$). For the proof we define $\varphi' = \psi(x, x_1, \ldots, x_n)$ for a formula of the form $\varphi(x_1, \ldots, x_n) = \exists x \psi(x, x_1, \ldots, x_n)$.

$$
\begin{aligned}
&\mathfrak{A} \models^{\nu} \varphi \text{ iff } \mathfrak{A} \models^{\nu} \neg\exists x_1, \ldots, x_n(\psi(x_{j_1}, \ldots, x_{j_\ell}) \wedge \neg\omega(x_{k_1}, \ldots, x_{k_m})) && (1)\\
&\quad \text{iff for all } a_1, \ldots, a_n \in A: && (2)\\
&\qquad \mathfrak{A} \not\models^{\nu} (\psi(x_{j_1}, \ldots, x_{j_\ell}) \wedge \neg\omega(x_{k_1}, \ldots, x_{k_m}))[a_1, \ldots, a_n]\\
&\quad \text{iff for all } (a_1, \ldots, a_n) \in A_{\alpha_{\varphi'}(x_1)} \times \ldots \times A_{\alpha_{\varphi'}(x_n)}: && (3)\\
&\qquad \mathfrak{A} \not\models^{\nu} (\psi(x_{j_1}, \ldots, x_{j_\ell}) \wedge \neg\omega(x_{k_1}, \ldots, x_{k_m}))[a_1, \ldots, a_n]\\
&\quad \text{iff for all } a_1, \ldots, a_n \in A_{\alpha_{\varphi'}(x_1)} \times \ldots \times A_{\alpha_{\varphi'}(x_n)}: && (4)\\
&\qquad \mathfrak{A} \not\models^{\nu} \psi[a_{j_1}, \ldots, a_{j_\ell}] \text{ or } \mathfrak{A} \not\models^{\nu} \neg\omega[a_{k_1}, \ldots, a_{k_m}]\\
&\quad \text{iff for all } a_1, \ldots, a_n \in A_{\alpha_{\varphi'}(x_1)} \times \ldots \times A_{\alpha_{\varphi'}(x_n)}: && (5)\\
&\qquad \mathfrak{A} \not\models^{\nu} \psi[a_{j_1}, \ldots, a_{j_\ell}] \text{ or } \mathfrak{A} \models^{\nu} \omega[a_{k_1}, \ldots, a_{k_m}]\\
&\qquad \text{or } a_p \notin A_{\alpha_\omega(x_t)} \text{ with } \beta(x_t) = a_p \text{ and } p, t \leq n\\
&\quad \text{iff for all } a_1, \ldots, a_n \in A_{\alpha_{\varphi'}(x_1)} \times \ldots \times A_{\alpha_{\varphi'}(x_n)}: && (6)\\
&\qquad \mathfrak{A} \not\models^{\nu} \psi[a_{j_1}, \ldots, a_{j_\ell}] \text{ or } \mathfrak{A} \models^{\nu} \omega[a_{k_1}, \ldots, a_{k_m}]\\
&\quad \text{iff for all } a_1, \ldots, a_n \in A: && (7)\\
&\qquad \mathfrak{A} \not\models^{\nu} \psi[a_{j_1}, \ldots, a_{j_\ell}] \text{ or } \mathfrak{A} \models^{\nu} \omega[a_{k_1}, \ldots, a_{k_m}]\\
&\quad \text{iff for all } a_1, \ldots, a_n \in A: && (8)\\
&\qquad \mathfrak{A} \not\models \psi[a_{j_1}, \ldots, a_{j_\ell}] \text{ or } \mathfrak{A} \models \omega[a_{k_1}, \ldots, a_{k_m}]\\
&\quad \text{iff } \mathfrak{A} \models \varphi && (9)
\end{aligned}
$$

(1) by Definition 5, (2) holds by 4. by Definition 5 (no restriction to β are needed, because φ is a sentence) and (3) holds by Proposition 6. (4) holds by 3. of Definition 5 and (5) holds by 2. of Definition 5. (6) holds because $a_p \notin A_{\alpha_\omega(x_t)}$ with $\beta(x_t) = a_p$ and $p, t \leq n$ contradicts $a_1, \ldots, a_n \in A_{\alpha_{\varphi'}(x_1)} \times \ldots \times A_{\alpha_{\varphi'}(x_n)}$. (7) holds by $\alpha_{\varphi'}(x_{j_i}) = \alpha_\psi(x_{j_i})$ and $\alpha_{\varphi'}(x_{k_t}) = \alpha_{\omega'}(x_{k_t})$ for all $i \leq \ell$ and $t \leq m$ and by Proposition 6. (8) holds by Claim 1 and (9) holds by Definition 1.

Ad 2. It suffices to show that, whenever $\mathcal{O}$ is consistent (and DI and compatible with the ν-ontology $\mathbb{O}$), $\mathcal{O}$ has a ν-model. By 1. of Theorem 9, that model then is a model of $\mathbb{O}$ too.

Let $\mathfrak{A}$ be a interpretation with $\mathfrak{A} \models \mathcal{O}$ and $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, R_2^{\mathfrak{A}}, \ldots)$. Then we create the ν-interpretation $\mathfrak{B}$ with $\mathfrak{B} = (B, R_1^{\mathfrak{B}}, \ldots R_k^{\mathfrak{B}})$ as follows:

$$
\begin{aligned}
B &= A \times I\\
R_i^{\mathfrak{B}} &= \{((a_1, i_1), (a_n, i_n)) \mid (a_1, \ldots, a_n) \in R_i^{\mathfrak{A}}\} \qquad \text{for all } R_i \text{ with } i \leq k \text{ and } \nu(R) = (i_1, \ldots, i_n)
\end{aligned}
$$

We can now use $\mathfrak{B}$ to show the desired property:

$$\mathfrak{A} \models \mathcal{O} \quad \text{implies} \quad \mathfrak{B} \models^{\nu} \mathcal{O} \qquad (\star)$$

For this purpose, we introduce the following property:

Claim 3. *For all ν-formulas $\varphi(x_1, \ldots, x_n) \in FO^{\wedge,\exists}$, all $a_1, \ldots, a_n \in A$ with:*

$$\mathcal{A} \models \varphi[a_1, \ldots, a_n] \textit{ iff } \mathfrak{B} \models^{\nu} \varphi[(a_1, \alpha_\varphi(x_1)), \ldots, (a_n, \alpha_\varphi(x_1))]$$

Proof of Claim 3.

We show this property by induction over the structure of φ. The base case with $\varphi[a_1, \ldots, a_n] = R[a_1, \ldots, a_n]$ and the conjunction can be shown in a straightforward way by applying the Definition of $\mathfrak{B}$ and Definition 5. The other case is shown as follows:

- $\varphi(x_1, \ldots, x_n) = \exists x \psi(x, x_1, \ldots, x_n)$

$$\begin{aligned}
\mathcal{A} \models \varphi[a_1, \ldots, a_n] &\text{ iff there is an } a \in \text{ A s.t. } \mathfrak{A} \models \psi[a, a_1, \ldots, a_n] && (1)\\
&\text{iff there is an } (a, i) \in \text{ B s.t. } \mathfrak{A} \models \psi[a, a_1, \ldots, a_n] && (2)\\
&\text{iff there is an } (a, i) \in \text{ B s.t.} && (3)\\
&\quad \mathfrak{B} \models^{\nu} \psi[(a, \alpha_\varphi(x)), (a_1, \alpha_\varphi(x_1)), \ldots, (a_n, \alpha_\varphi(x_n))]\\
&\text{iff } \mathfrak{B} \models^{\nu} \varphi[(a_1, \alpha_\varphi(x_1)), \ldots, (a_n, \alpha_\varphi(x_1))] && (4)
\end{aligned}$$

(1) holds by Definition 1, (4) holds by Definition 5, (2) holds by the Definition of $\mathfrak{B}$ and (3) holds by the Definition of $\mathfrak{B}$, the induction hypothesis and because $\alpha_\varphi(x_i) = \alpha_\psi(x_i)$ for all $i \leq n$.

To pove property ($\star$) it now suffices to show that $\mathfrak{A} \models \varphi$ implies $\mathfrak{A} \models^{\nu} \varphi$ with $\varphi = \forall x_1, \ldots, x_n(\psi(x_{j_1}, \ldots, x_{j_\ell}) \rightarrow \omega(x_{k_1}, \ldots, x_{k_m}))$ and $\{j_1, \ldots, j_\ell, k_1, \ldots, k_m\} = \{1, \ldots, n\}$

$$\begin{aligned}
\mathfrak{A} \models \varphi &\text{ iff for all } a_1, \ldots, a_n \in A : && (1)\\
&\quad \mathfrak{A} \models \neg\psi(x_{j_1}, \ldots, x_{j_\ell}) \vee \omega(x_{k_1}, \ldots, x_{k_m})[a_1, \ldots, a_n]\\
&\text{iff for all } a_1, \ldots, a_n \in A : && (2)\\
&\quad \mathfrak{A} \models \neg\psi[a_{j_1}, \ldots, a_{j_\ell}] \text{ or } \mathfrak{A} \models \omega[a_{k_1}, \ldots, a_{k_m}]\\
&\text{iff for all } a_1, \ldots, a_n \in A_{\alpha_{\varphi'}(x_1)} \times \ldots \times A_{\alpha_{\varphi'}(x_n)} : && (3)\\
&\quad \mathfrak{A} \models \neg\psi[a_{j_1}, \ldots, a_{j_\ell}] \text{ or } \mathfrak{A} \models \omega[a_{k_1}, \ldots, a_{k_m}]\\
&\text{iff for all } a_1, \ldots, a_n \in A_{\alpha_{\varphi'}(x_1)} \times \ldots \times A_{\alpha_{\varphi'}(x_n)} : && (4)\\
&\quad \mathfrak{A} \models^{\nu} \neg\psi[(a_{j_1}, \alpha_\varphi(x_{j_1})), \ldots, (a_{j_\ell}, \alpha_\varphi(x_{j_\ell}))] \text{ or}\\
&\quad \mathfrak{A} \models^{\nu} \omega[(a_{k_1}, \alpha_\varphi(x_{k_1})), \ldots, (a_{k_m}, \alpha_\varphi(x_{k_m}))]\\
&\text{iff } \mathfrak{B} \models^{\nu} \varphi && (5)
\end{aligned}$$

(1), (2) and (5) hold by Definition 1, (3) holds by Proposition 6 and (4) holds by Claim 3.

8.1.2 Algorithm

In this section, we will discuss how an algorithm, like the algorithm in Chapter 4 can be defined for TGDs. We will omit the proofs of correctness and maximality as the sole purpose of this section was to sketch a possible algorithm. The first step of the Algorithm is to define the graph $G = (V, E, L)$, as in Algorithm 1 of Chapter 4.1.

The *signature* τ is a set of relational atoms with a fixed arity. We define $rel(\varphi)$ as the occurences of all relational atoms in φ, e.g. $rel(\forall x, y.R(x, y) \rightarrow \exists z.S(y, z)) = \{R(x, y), S(y, z)\}$.

Algorithm 5: Partitioning an ontology $\mathcal{O}$

1 **Function** `partition(`$\mathcal{O}$`):`
  input : $\mathcal{O}$ with signature τ
  output : ν-ontology $\mathbb{O}$
2 $V \leftarrow \{R_1, \ldots, R_n \mid R \in \tau$ with $\mathsf{arity}(R) = n\}$;
3 $E \leftarrow \{(R_i, S_j) \mid \exists \varphi \in \mathcal{O}$ with $R(x_1, \ldots, x_n) \in ref(\varphi)$ and $S(y_1, \ldots, y_m) \in ref(\varphi)$ and $x_i = y_j\}$
4 $L \leftarrow$ `fixAxiomLabels(`$G, \mathcal{O}$`)`
5 $G \leftarrow (V, E, L)$
6 $\{G_1, \ldots, G_n\} \leftarrow$ all connected components (CCs) of G with ≥ 1 axiom label
7 **forall** $i \leq n$ **do** $\mathcal{O}_i \leftarrow \{\alpha \mid \alpha \in L(v, v')$ for some edge (v, v') in $G_i\}$
8 $\mathbb{O} \leftarrow (\mathcal{O}_1, \ldots, \mathcal{O}_n)$
9 **return** $(\mathbb{O})$

We give the procedure by the routine `partition(`$\mathcal{O}$`)` of Algorithm 5. The routine takes an ontology $\mathcal{O}$ with signature τ as input and returns a ν-ontology $\mathbb{O}$ so that $\mathcal{O}$ and $\mathbb{O}$ are equivalent.

The algorithm will create the vertices $R_1, \ldots, R_n$ for each relational atom $R \in \tau$ with $\mathsf{arity}(R) = n$ (see line 2). Therefore each vertex R_i represent the i-th position of R. After adding the vertices we add the edges (see line 3). Vertices R_i and S_j are connected if $R(x_1, \ldots, x_n)$ and $S(y_1, \ldots, y_m)$ occur in an $\varphi \in \mathcal{O}$ with $x_i = y_j$. In the next step we add the axioms as labels to the edges. We will set $L(e) = \varphi$ for a fixed $e \in E_\varphi$ (represented by `fixAxiomLabels` in line 7).

After this, the procedure is analogous to the Algorithm 1: We will calculate the connected components of G, and we define $\mathbb{O}$ as the connected components and their labels.

The addition of vertices and labels is still in linear time (given we choose a linear time procedure for adding the labels, e.g., labeling the first edge that is created for a $\varphi \in \mathcal{O}$). For each variable x in a formula φ, we add an edge for each pair of occurrences of x in the relational atoms in φ. Therefore the complexity is $O(nph^2)$ with n being the number of $\varphi \in \mathcal{O}$, p the number of variables in φ and $h = rel(\varphi)$. If we set $k =\mid \mathcal{O} \mid$ then we can

argue that the complexity is $O(k^3)$. The complexity of the rest of the algorithm can be calculated analogously to the complexity of Algorithm 1 in Chapter 4.4. Therefore the whole algorithm is in $O(n^3)$.

8.1.3 Summary

We showed that partitioning by $\mathcal{E}$-connection could be transferred to other fragments of FOL than $\mathcal{SROIQ}(\mathcal{D})$. The fragment we have chosen for this demonstration is a logic only containing TGD's. TGD's allow for an easier transfer because of their form: They are always DI and contain no negation of relational atoms with an arity ≥ 2. The form is also similar to the representation of subsumption axioms in FOL.

8.1.4 Outlook

Further investigation could help to find other fragments of first-order logic to which partitioning based on $\mathcal{E}$-connection could be transferred to. Especially fragments without equality (see page 67) and without negation of relational atoms with more than one free variable (see page 68) are interesting, because a simple transfer of theorems and results might be possible. These fragments can contain non-DI sentences, as observed for $\mathcal{SROIQ}(\mathcal{D})$. Therefore an algorithm can only be defined if domain-independence is decidable for this fragment.

A promising candidate is the *unary-negation fragment of first-order logic* (UNFO), first described by ten Cate and Segoufin (2011). It embeds several description logics like $\mathcal{ALC}$.

UNFO is a fragment of FOL that only allows negation on subformulas that has at most one free variable. UNFO formulas are given by the following grammar:

$$\phi ::= R(X) \mid x = y \mid \phi \wedge \phi \mid \phi \vee \phi \mid \exists x.\phi \mid \neg\phi(x)$$

where, in the last clause, ϕ has no other free variables besides (possibly) x. From now on we write $\mathcal{X}(x)$ to say that $\mathcal{X}$ has at most one free variable x. Segoufin and ten Cate (2013) showed that every UNFO sentence is equivalent to a UNFO sentence in UN-normal form that is generated by the following grammar:

$$\mathcal{X}(y) ::= \exists \bar{z}\psi(y, \bar{z}) \mid \neg\mathcal{X}(y) \mid \mathcal{X}_1(y) \vee \mathcal{X}_2(y)$$

where $\psi(y, \bar{z})$ is of the form:

$$\exists \bar{z}(\tau(\bar{z}) \wedge z_i = y \wedge \bigwedge_{j\in\{1\ldots n\}\setminus\{i\}} \phi_j(z_j)) \tag{8.1}$$

or

$$\exists \bar{z}(\tau(\bar{z}) \wedge \bigwedge_{j\in\{1\ldots n\}} \phi_j(z_j)) \tag{8.2}$$

The use of equality atoms is not allowed in our setting, as discussed on Page 67. The discussed normal form allows equality in only one place, namely formula 8.1. The equality atoms can be easily eliminated, as discussed by Jung et al. (2018).

On page 68, we discussed how negation on relational atoms with more than one free variable implies that locality does no longer characterizes domain-independence. The limitation of negations to subformulas with at most one free variable is the main reason why we consider UNFO a promising candidate. We suspect that a major task in transferring the theorems and results to UNFO will be to find an efficient way to characterize DI.

Chapter 9

Conclusion and Future Work

In this thesis, we investigated and evaluated the decomposition method "Partitioning using $\mathcal{E}$-connections". During our investigation, we developed a simplified notation and extended it to the full OWL 2, with the exception of the universal role. We have developed a new, conceptually much simpler algorithm that is deterministic and runs in linear rather than quadratic runtime.

We have implemented the algorithm in Java. We have developed several heuristics, e.g., removing the classes that are highest in heuristics to improve the quality of partitioning. We call the implemented tool EPartitioner[1].

We have implemented extensive testing for the EPartitioner and evaluated the tool on a large and diverse established corpus of biomedical ontologies. The evaluation has shown that the standard algorithm leads to weak results. The resulting partitionings are coarse, and one component often contains most of the logical axioms. The implemented heuristics improve these results but are often accompanied by a high loss of logical axioms.

To examine whether "Partitioning using $\mathcal{E}$-connections" can be transferred to other fragments of first-order logic, we transferred "Partitioning using $\mathcal{E}$-connections" to *tuple generating dependency's (TGD)* in Chapter 8. We also discussed that the *unary-negation fragment of first-order logic* (UNFO) is a promising candidate for the transfer of $\mathcal{E}$-connections. It turns out that the main problem with the transfer is not the $\mathcal{E}$-connections themselves, but *safety*, which is generally undecidable.

9.1 Future Work

For future work, user studies would be useful to answer the question if decompositions of the simple algorithm without heuristics or with heuristics are useful. Questions for the users would be e.g., how fine the decompositions should be and which loss of logical axioms would be tolerable. The authors of this thesis have set the previous limits according to their own opinion. Furthermore, it would be interesting to know if the pure insight into the structure of the ontology could be helpful.

On the application side, the development of a user interface could be part of future work. An interesting question could be how to make the different options offered by

[1] Repository of EPartitioner: `https://github.com/sasjonge/epartition`

the EPartitioner understandable to the user. Furthermore, one could investigate how to display the partitioning so that it is understandable for the user.

Regarding the user questions mentioned above, the question for future work is whether the effort of further theoretical or application-related work is reasonable in relation to the mediocre results.

On the theoretical side, among other things, the logical foundations of the *Ignore Properties Heuristic* (IPH) be examined. It could be investigated how to adapt the notations and the algorithm to include the possible ignoring of global (or domain/rangeglobal) properties. It should also be investigated how this affects the logical guarantees.

Another point for future work on the theoretical side could be the extension of the notations and the algorithm to other fragments of FOL like UNFO. It could also be investigated whether $\mathcal{E}$-connections and partitioning based on $\mathcal{E}$-connections could be transferred to some SO features.

Bibliography

Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.

Alchourrn, C. E., Grdenfors, P., and Makinson, D. (1985). On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50(2):510530.

Amato, F., De Santo, A., Moscato, V., Persia, F., Picariello, A., and Poccia, S. R. (2015). Partitioning of ontologies driven by a structure-based approach. In *Proc. of ICSC-15*, pages 320–323. IEEE Computer Society.

Baader, F., Lutz, C., Sturm, H., and Wolter, F. (2002). Fusions of description logics and abstract description systems. *J. Artif. Intell. Res.*, 16:1–58.

Baget, J.-F., Leclère, M., Mugnier, M.-L., Rocher, S., and Sipieter, C. (2015). Graal: A toolkit for query answering with existential rules. In Bassiliades, N., Gottlob, G., Sadri, F., Paschke, A., and Roman, D., editors, *Rule Technologies: Foundations, Tools, and Applications*, pages 328–344, Cham. Springer International Publishing.

Bao, J., Voutsadakis, G., Slutzki, G., and Honavar, V. (2009). Package-based description logics. In Stuckenschmidt et al. (2009), pages 349–371.

Beeri, C. and Vardi, M. Y. (1984). A proof procedure for data dependencies. *J. ACM*, 31(4):718–741.

Bellomarini, L., Sallinger, E., and Gottlob, G. (2018). The vadalog system: Datalog-based reasoning for knowledge graphs. *Proc. VLDB Endow.*, 11(9):975987.

Blondel, V., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics Theory and Experiment*, 2008.

Borgida, A. and Serafini, L. (2003). *Distributed Description Logics: Assimilating Information from Peer Sources", bookTitle="Journal on Data Semantics I*, pages 153–184. Springer Berlin Heidelberg, Berlin, Heidelberg.

Calì, A., Gottlob, G., and Pieris, A. (2010). Query answering under non-guarded rules in datalog+/-. In Hitzler, P. and Lukasiewicz, T., editors, *Web Reasoning and Rule Systems*, pages 1–17, Berlin, Heidelberg. Springer Berlin Heidelberg.

Cuenca Grau, B. (2005). *Combination and integration of ontologies on the semantic web*. PhD thesis, Universidad de Valencia.

Cuenca Grau, B., Horrocks, I., Kazakov, Y., and Sattler, U. (2009a). Extracting modules from ontologies: A logic-based approach. In Stuckenschmidt et al. (2009), pages 159–186.

Cuenca Grau, B., Parsia, B., and Sirin, E. (2009b). Ontology integration using $\mathcal{E}$-Connections. In Stuckenschmidt et al. (2009), pages 293–320.

Cuenca Grau, B., Parsia, B., Sirin, E., and Kalyanpur, A. (2005). Automatic partitioning of OWL ontologies using $\mathcal{E}$-Connections. In *Proc. of DL-05*, volume 147 of *CEUR-WS.org*.

Cuenca Grau, B., Parsia, B., Sirin, E., and Kalyanpur, A. (2006). Modularity and web ontologies. In *Proc. of KR-06*, pages 198–209. AAAI Press.

Del Vescovo, C., Parsia, B., Sattler, U., and Schneider, T. (2011). The modular structure of an ontology: Atomic decomposition. In *Proc. of IJCAI-11*, pages 2232–2237.

Gatens, W., Konev, B., and Wolter, F. (2013). Module extraction for acyclic ontologies. In *Proc. of WoMO-13*, volume 1081 of *CEUR-WS.org*.

Ghilardi, S., Lutz, C., and Wolter, F. (2006). Did I damage my ontology? a case for conservative extensions in description logics. In Doherty, P., Mylopoulos, J., and Welty, C., editors, *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 187–197. AAAI Press.

Grau, B. C., Horrocks, I., Kazakov, Y., and Sattler, U. (2008). Modular reuse of ontologies: Theory and practice. *J. Artif. Intell. Res.*, 31:273–318.

Grau, B. C., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., and Wang, Z. (2013). Acyclicity notions for existential rules and their application to query answering in ontologies. *J. Artif. Intell. Res. (JAIR)*, 47:741–808.

Hoehndorf, R. (2010). What is an upper level ontology? `http://ontogenesis.knowledgeblog.org/740`.

Hopcroft, J. E. and Tarjan, R. E. (1973). Efficient algorithms for graph manipulation [H] (algorithm 447). *Commun. ACM*, 16(6):372–378.

Horridge, M. and Bechhofer, S. (2011). The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21.

Horrocks, I. and Sattler, U. (2001). Ontology reasoning in the SHOQ(D) description logic. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 199–204.

Jongebloed, S. and Schneider, T. (2018). Ontology partitioning using $\mathcal{E}$-Connections revisited. In Ortiz, M. and Schneider, T., editors, *Proceedings of the 31st International Workshop on Description Logics (DL-18)*, volume 2211 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Jung, J. C., Lutz, C., Martel, M., and Schneider, T. (2018). Querying the unary negation fragment with regular path expressions. In *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, pages 15:1–15:18.

Kifer, M. (1988). On safety, domain independence, and capturability of database queries (preliminary report). In *Proc. of JCDKB-88*, pages 405–415.

Klinov, P., Del Vescovo, C., and Schneider, T. (2012). Incrementally updateable and persistent decomposition of OWL ontologies. In *Proc. of OWLED-12*, volume 849 of *CEUR-WS.org*.

Konev, B., Lutz, C., Ponomaryov, D., and Wolter, F. (2010). Decomposing description logic ontologies. In *Proc. of KR-10*, pages 236–246. AAAI Press.

Konev, B., Lutz, C., Walther, D., and Wolter, F. (2008). Semantic modularity and module extraction in description logics. In *Proc. of ECAI-08*, volume 178 of *FAIA*, pages 55–59. IOS Press.

Krötzsch, M., Simančik, F., and Horrocks, I. (2012). A description logic primer. *CoRR*, abs/1201.4089.

Kutz, O., Lutz, C., Wolter, F., and Zakharyaschev, M. (2004). $\mathcal{E}$-connections of abstract description systems. *Artif. Intell.*, 156(1):1–73.

Lutz, C., Walther, D., and Wolter, F. (2007). Conservative extensions in expressive description logics. In Veloso, M., editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 453–458. AAAI Press.

Magka, D., Motik, B., and Horrocks, I. (2012). Modelling structured domains using description graphs and logic programming. In Simperl, E., Cimiano, P., Polleres, A., Corcho, O., and Presutti, V., editors, *The Semantic Web: Research and Applications*, pages 330–344, Berlin, Heidelberg. Springer Berlin Heidelberg.

Matentzoglu, N. and Parsia, B. (2017). BioPortal Snapshot 30 March 2017 (data set). `http://doi.org/10.5281/zenodo.439510`.

Michail, D., Kinable, J., Naveh, B., and Sichi, J. V. (2019). Jgrapht–a java library for graph data structures and algorithms. *arXiv preprint arXiv:1904.08355*.

Mungall, C. (2009). Experiences using logic programming in bioinformatics. In Hill, P. M. and Warren, D. S., editors, *Logic Programming*, pages 1–21, Berlin, Heidelberg. Springer Berlin Heidelberg.

Noy, N., Shah, N., Whetzel, P., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Rubin, D., Storey, M.-A., Chute, C., and Musen, M. (2009). Bioportal: Ontologies and integrated data resources at the click of a mouse. *Nucleic acids research*, 37:W170–3.

Parikh, R. (1999). Logic, language and computation, vol. 2. chapter Beliefs, Belief Revision, and Splitting Languages, pages 266–278. Center for the Study of Language and Information, Stanford, CA, USA.

Parsia, B., Sattler, U., and Schneider, T. (2008). Easy keys for OWL. In *Proc. of OWLED-08EU*, volume 432 of *CEUR-WS.org*.

Parsia, B. and Schneider, T. (2010). The modular structure of an ontology: an empirical study. In Lin, F., Sattler, U., and Truszczynski, M., editors, *Proc. of KR-2010*, pages 584–586. AAAI Press.

Romero, A. A., Kaminski, M., Grau, B. C., and Horrocks, I. (2016). Module extraction in expressive ontology languages via datalog reasoning. *J. Artif. Int. Res.*, 55(1):499–564.

Segoufin, L. and ten Cate, B. (2013). Unary negation. *Logical Methods in Computer Science*, 9(3).

Seidenberg, J. and Rector, A. (2006). Web ontology segmentation: Analysis, classification and use. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 13–22, New York, NY, USA. ACM.

Serafini, L. and Tamilin, A. (2009). Composing modular ontologies with Distributed Description Logics. In Stuckenschmidt et al. (2009), pages 321–347.

Stuckenschmidt, H., Parent, C., and Spaccapietra, S., editors (2009). *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *LNCS*. Springer.

Stuckenschmidt, H. and Schlicht, A. (2009). Structure-based partitioning of large ontologies. In Stuckenschmidt et al. (2009), pages 187–210.

Suntisrivaraporn, B. (2008a). Module extraction and incremental classification: A pragmatic approach for $\mathcal{EL}$ ontologies. In *Proc. of ESWC-08*, volume 5021 of *LNCS*, pages 230–244. Springer.

Suntisrivaraporn, B. (2008b). Module extraction and incremental classification: A pragmatic approach for ontologies. pages 230–244.

ten Cate, B. and Segoufin, L. (2011). Unary negation. In Schwentick, T. and Dürr, C., editors, *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, March 10-12, 2011, Dortmund, Germany*, volume 9 of *LIPIcs*, pages 344–355. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Traag, V. A., Waltman, L., and van Eck, N. J. (2018). From louvain to leiden: guaranteeing well-connected communities. *CoRR*, abs/1810.08473.

Van Gelder, A. and Topor, R. W. (1991). Safety and translation of relational calculus queries. *ACM Trans. Database Syst.*, 16(2):235–278.

Vardi, M. Y. (1981). The decision problem for database dependencies. *Inf. Process. Lett.*, 12(5):251–254.

Vescovo, C. D., Horridge, M., Parsia, B., Sattler, U., Schneider, T., and Zhao, H. (2019). Modular structures and atomic decomposition in ontologies.

Zhang, J., Ma, Z., Sun, Q., and Yan, J. (2018). Research review on algorithms of community detection in complex networks. *Journal of Physics: Conference Series*, 1069:012124.

Appendices

Appendix A

Test Descriptions and Results

In this section, we describe the implemented Unit-Tests. The goal of our unit tests is to test if our implementation connects the correct vertices for an axiom or subconcept. For the tests (with the exception of the Run-Through-Example-Test) we will at first test an "pre"-ontology $\mathcal{O}_{pre}$, which will build the basis for the tests of the different axioms and subconcepts. Then we will test if the axiom or subconcept connects the correct vertices in $\mathcal{O}_{pre}$ by adding the relevant axiom/subconcept to the pre-ontology.

We will describe the tests for axioms before the tests of subconcepts, because the axioms are necessary to describe and create tests (in- and output are ontologies/a set of axioms). The order of the test (after the Run-Through-Example-Test) is based on Section 8 (ClassExspressions) and Section 9 (Axioms) of the "OWL 2 Web Ontology Language - Structural Specification and Functional-Style Syntax" documentation[1]. To describe the subconcepts and axioms we use the Functional-Style Syntax of OWL [1].

A.1 Test of Run-Through-Example

In this section we describe the unit test with our Run-Through-Example as input. The test is described in Table A.1.

A.2 Test of Class Expression Axioms

In section 9.1 of the "Structural Specification" axioms that allow to express relationships between class expressions are defined. The first axioms described are subclass axioms. In our first test, we want to test if the class expressions A and B are in the same subgraph, if we add the axiom $\gamma = SubClassOf(A, B)$. For this we first create an ontology $\mathcal{O}_{pre}$ containing the axioms $\alpha = SubClassOf(A, B')$ and $\beta = SubClassOf(B, B')$ and test the algorithm with $\mathcal{O}_{pre}$ as input (see Table A.2). We expect that the algorithm returns $\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta\}$.
The test in Table A.3 checks if the addition of γ to $\mathcal{O}_{pre}$ will force α and β into the same subgraph.

We expect that equivalent classes, disjoint classes and disjoint union of class expressions will result in the same connectivity: All their entities will be pairwise connected.

[1] https://www.w3.org/TR/owl2-syntax

Table A.1: Test of run through example

Name	`runThroughExampleTest`
Tested sub-concepts and axioms	ObjectSomeValuesFrom, ObjectAllValuesFrom, ObjectIntersectionOf, SubClassOf, SubObjectPropertyOf
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, ObjectSomeValues.(r, B)),$ $\beta = SubClassOf(C, ObjectAllValues.(s, B)),$ $\gamma = SubObjectPropertyOf(r, s),$ $\delta = SubClassOf(B, ObjectIntersec.Of(B', B''))\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha, \beta, \gamma\}$ and $\mathcal{O}_2 = \{\delta\}$

Table A.2: Test of $\mathcal{O}_{pre}$ for SubClassOf axioms

Name	`subClassOfPreTest`
Tested sub-concepts and axioms	SubClassOf
Input	$\mathcal{O}_{pre} = \{\alpha = SubClassOf(A, B'),$ $\beta = SubClassOf(B, B'')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta\}$

Table A.3: Test of SubClassOf axioms

Name	`subClassOfTest`
Tested sub-concepts and axioms	SubClassOf
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, B'),$ $\beta = SubClassOf(B, B''),$ $\gamma = SubClassOf(A, B)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.4: Test of EquivalentClasses axioms

Name	`equivalentClassesTest`
Tested sub-concepts and axioms	EquivalentClasses
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, B'),$ $\beta = SubClassOf(B, B''),$ $\gamma = EquivalentClasses(A, B)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.5: Test of DisjointClasses axioms

Name	`disjointClassesTest`
Tested sub-concepts and axioms	DisjointClasses
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, B'),$ $\beta = SubClassOf(B, B''),$ $\gamma = DisjointClasses(A, B)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Therefore we test them similar to subclass axioms (see Tables A.4, A.5 and A.7). For equivalent classes axioms and disjoint classes we use the $\mathcal{O}_{pre}$ tested in Table A.2. The test of $\mathcal{O}_{pre}$ for disjoint union axioms is described in Table A.6.

Additionally to the tests above we implemented tests with more than two inputs for equivalent classes and disjoint classes axioms and with more than two classes in the union part of the disjoint union axioms. For this tests we test and create an $\mathcal{O}_{pre}$ that will result in five components (see Table A.8). The tests for equivalent classes, disjoint classes and disjoint union axioms are described in Tables A.9, A.10 and A.11.

Table A.6: Test of $\mathcal{O}_{pre}$ for DisjointUnion axioms

Name	`disjointUnionPreTest`
Tested sub-concepts and axioms	SubClassOf
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, B'),$ $\beta = SubClassOf(B, B''),$ $\gamma = SubClassOf(C, B''')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3\}$ with $\mathcal{O}_1 = \{\alpha\}$, $\mathcal{O}_2 = \{\beta\}$ and $\mathcal{O}_3 = \{\gamma\}$

Table A.7: Test of DisjointUnion axioms

Name	`disjointUnionTest`
Tested sub-concepts and axioms	DisjointUnion
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, B'),$ $\beta = SubClassOf(B, B''),$ $\gamma = SubClassOf(C, B'''),$ $\delta = DisjointUnion(A, B, C)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.8: Test of $\mathcal{O}_{pre}$ for EquivalentClasses axioms with 5 classes

Name	`equivalentClassesManyPreTest`
Tested sub-concepts and axioms	SubClassOf
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, A'),$ $\beta = SubClassOf(B, B'),$ $\gamma = SubClassOf(C, C'),$ $\delta = SubClassOf(D, D'),$ $\epsilon = SubClassOf(E, E')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4, \mathcal{O}_5\}$ with $\mathcal{O}_1 = \{\alpha\}$, $\mathcal{O}_2 = \{\beta\}$, $\mathcal{O}_3 = \{\gamma\}$, $\mathcal{O}_4 = \{\delta\}$ and $\mathcal{O}_5 = \{\epsilon\}$

Table A.9: Test of EquivalentClasses axioms with 5 classes

Name	`equivalentClassesManyTest`
Tested sub-concepts and axioms	EquivalentClasses
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, A'),$ $\beta = SubClassOf(B, B'),$ $\gamma = SubClassOf(C, C'),$ $\delta = SubClassOf(D, D'),$ $\epsilon = SubClassOf(E, E'),$ $\zeta = EquivalentClasses(A, B, C, D, E)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.10: Test of DisjointClasses axioms with 5 classes

Name	`disjointClassesManyTest`
Tested sub-concepts and axioms	DisjointClasses
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, A'),$ $\beta = SubClassOf(B, B'),$ $\gamma = SubClassOf(C, C'),$ $\delta = SubClassOf(D, D'),$ $\epsilon = SubClassOf(E, E'),$ $\zeta = DisjointClasses(A, B, C, D, E)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.11: Test of DisjointUnion axioms with 5 classes

Name	`disjointUnionManyTest`
Tested sub-concepts and axioms	DisjointUnion
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, A'),$ $\beta = SubClassOf(B, B'),$ $\gamma = SubClassOf(C, C'),$ $\delta = SubClassOf(D, D'),$ $\epsilon = SubClassOf(E, E'),$ $\zeta = DisjointUnion(A, B, C, D, E)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.12: Test of $\mathcal{O}_{pre}$ for SubObjectPropertyOf axioms

Name	`subObjectPropertyOfPreTest`
Tested sub-concepts and axioms	SubObjectPropertyOf
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(r, s),$ $\beta = SubObjectPropertyOf(r', s')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta\}$

Table A.13: Test of SubObjectPropertyOf axioms

Name	`SubObjectPropertyOfTest`
Tested sub-concepts and axioms	SubObjectPropertyOf
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(r, s),$ $\beta = SubObjectPropertyOf(r', s'),$ $\gamma = SubObjectPropertyOf(r, r')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

A.3 Test of Object Property Axioms

The subobject property axioms are tested similiarly to the subclass axioms. At first we test a pre-ontology in Table A.12. After this we test the effect of the addition of a subobject property axiom in Table A.13

We will define the same tests (using the same $\mathcal{O}_{pre}$ of Table A.12) for equivalent object properties and disjoint object properties axioms by simply replacing `SubObjectProp.` with the corresponding name, `EquivalentObjectProperties` for equivalent object properties axioms (see Table A.14) and `DisjointObjectProperties` for disjoint object properties axioms (see Table A.15).

Similarly to the tests for e.g. equivalent classes axioms we implemented tests with more than two inputs for equivalent object properties and disjoint object properties axioms. For this tests we test and create an $\mathcal{O}_{pre}$ that will result in five components (see Table A.16). The tests for equivalent classes, disjoint classes and disjoint union axioms

Table A.14: Test of EquivalentObjectProperties axioms

Name	`equivalentObjectPropertiesTest`
Tested sub-concepts and axioms	EquivalentObjectProperties
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(r, s),$ $\beta = SubObjectPropertyOf(r', s'),$ $\gamma = EquivalentObjectProperties(r, r')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.15: Test of DisjointObjectProperties axioms

Name	`disjointObjectPropertiesTest`
Tested sub-concepts and axioms	DisjointObjectProperties
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(r, s),$ $\beta = SubObjectPropertyOf(r', s'),$ $\gamma = DisjointObjectProperties(r, r')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.16: Test of $\mathcal{O}_{pre}$ for EquivalentObjectProperties axioms with 5 roles

Name	`equivalentObjectPropertiesManyPreTest`
Tested sub-concepts and axioms	SubObjectPropertyOf
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(r, r'),$ $\beta = SubObjectPropertyOf(s, s'),$ $\gamma = SubObjectPropertyOf(t, t'),$ $\delta = SubObjectPropertyOf(v, v'),$ $\epsilon = SubObjectPropertyOf(w, w')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4, \mathcal{O}_5\}$ with $\mathcal{O}_1 = \{\alpha\}$, $\mathcal{O}_2 = \{\beta\}$, $\mathcal{O}_3 = \{\gamma\}$, $\mathcal{O}_4 = \{\delta\}$ and $\mathcal{O}_5 = \{\epsilon\}$

are described in Tables A.17 and A.18.

We expect that inverse object properties axioms of the form

$$InverseObjectProperties(r, s)$$

will force the subgraphs of r_0 and s_1 as well as the subgraphs for r_1 and s_0 together. To test this type of axioms we first create a test for $\mathcal{O}_{pre}$ described in Table A.19 and then test the described behaviour in Table A.20.

Complex role inclusions axioms of the form

$$SubObjectProp.(ObjectPropertyChain(s, t, v), r)$$

connect S_0 with r_0, s_1 with t_0, t_1 with v_0 and v_1 with r_1. Therefore the $\mathcal{O}_{pre}$ (described in Table A.21) will have one subgraph for each of this vertices. The test of the addition of ι is described in Table A.22.

Now we will have a look at object property domain axioms. We observed that it is sufficient to connect R_0 with the class expression. Similiarly to our test method for subclass axioms we will first show that an ontology $\mathcal{O}_{pre}$ is subgraphed into two parts (see Table A.24) and that the addition of the object property domain axiom will force this subgraphs into one subgraph. We set $\mathcal{O}_{pre} = \{\alpha = SubObjectPropertyOf(r, s), \beta = SubClassOf(A, B)\}$ and $\gamma = ObjectPropertyDomain(r, A)$. The description of the first test can be found in Table **??** and the test for the addition of γ can be found in Table A.2

Table A.17: Test of EquivalentObjectProperties axioms with 5 roles

Name	`equivalentObjectPropertiesManyTest`
Tested sub-concepts and axioms	SubObjectPropertyOf
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(r, r'),$ $\beta = SubObjectPropertyOf(s, s'),$ $\gamma = SubObjectPropertyOf(t, t'),$ $\delta = SubObjectPropertyOf(v, v'),$ $\epsilon = SubObjectPropertyOf(w, w'),$ $\zeta = EquivalentObjectProperties(r, s, t, v, w)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.18: Test of DisjointObjectProperties axioms with 5 roles

Name	`disjointObjectPropertiesManyTest`
Tested sub-concepts and axioms	DisjointObjectProperties
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(r, r'),$ $\beta = SubObjectPropertyOf(s, s'),$ $\gamma = SubObjectPropertyOf(t, t'),$ $\delta = SubObjectPropertyOf(v, v'),$ $\epsilon = SubObjectPropertyOf(w, w'),$ $\zeta = DisjointObjectProperties(r, s, t, v, w)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.19: Test of $\mathcal{O}_{pre}$ for InverseObjectProperties axioms

Name	`inverseObjectPropertiesPreTest`
Tested sub-concepts and axioms	InverseObjectProperties
Input	$\mathcal{O} = \{\alpha = InverseObjectProperties(r, s),$ $\beta = InverseObjectProperties(r', s')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_1 = \{\beta\}$

Table A.20: Test of InverseObjectProperties axioms

Name	`inverseObjectPropertiesTest`
Tested sub-concepts and axioms	InverseObjectProperties
Input	$\mathcal{O} = \{\alpha = InverseObjectProperties(r, s),$ $\beta = InverseObjectProperties(r', s'),$ $\gamma = InverseObjectProperties(r, r')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.21: Test of $\mathcal{O}_{pre}$ for SubObjectPropertyChain axioms

Name	`subObjectPropertyChainPreTest`
Tested sub-concepts and axioms	SubObjectPropertyOf, InverseObjectProperties
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(r, r'),$ $\beta = SubObjectPropertyOf(s, s'),$ $\gamma = SubObjectPropertyOf(t, t'),$ $\delta = SubObjectPropertyOf(v, v'),$ $\epsilon = InverseObjectProperties(r'', r),$ $\zeta = InverseObjectProperties(s'', s),$ $\eta = InverseObjectProperties(t'', t),$ $\theta = InverseObjectProperties(v'', v')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4, \mathcal{O}_5, \mathcal{O}_6, \mathcal{O}_7, \mathcal{O}_8\}$ with $\mathcal{O}_1 = \{\alpha\}$, $\mathcal{O}_2 = \{\beta\}$, $\mathcal{O}_3 = \{\gamma\}$, $\mathcal{O}_4 = \{\delta\}$, $\mathcal{O}_5 = \{\epsilon\}$, $\mathcal{O}_6 = \{\zeta\}$, $\mathcal{O}_7 = \{\eta\}$ and $\mathcal{O}_8 = \{\theta\}$

Table A.22: Test of SubObjectPropertyChain axioms

Name	`subObjectPropertyChainTest`
Tested sub-concepts and axioms	ObjectPropertyChain
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(r, r'),$ $\beta = SubObjectPropertyOf(s, s'),$ $\gamma = SubObjectPropertyOf(t, t'),$ $\delta = SubObjectPropertyOf(v, v'),$ $\epsilon = InverseObjectProperties(r'', r),$ $\zeta = InverseObjectProperties(s'', s),$ $\eta = InverseObjectProperties(t'', t),$ $\theta = InverseObjectProperties(v'', v'),$ $\iota = SubObjectProp.(ObjectProp.Ch.(s, t, v), r)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4\}$ with $\mathcal{O}_1 = \{\alpha, \beta, \iota\}$, $\mathcal{O}_2 = \{\gamma, \zeta\}$, $\mathcal{O}_3 = \{\delta, \eta\}$ and $\mathcal{O}_4 = \{\epsilon, \theta\}$

Table A.23: Test of $\mathcal{O}_{pre}$ for ObjectPropertyDomain axioms

Name	`objectPropertyDomainPreTest`
Tested sub-concepts and axioms	SubClassOf, SubObjectPropertyof
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(r, s),$ $\beta = SubClassOf(A, B)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta\}$

Table A.24: Test of ObjectPropertyDomain axioms

Name	`objectPropertyDomainTest`
Tested sub-concepts and axioms	ObjectPropertyDomain
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(r, s),$ $\beta = SubClassOf(A, B),$ $\gamma = ObjectPropertyDomain(r, A)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.25: Test of $\mathcal{O}_{pre}$ for ObjectPropertyRange axioms

Name	`objectPropertyRangePreTest`
Tested sub-concepts and axioms	ObjectPropertyRange
Input	$\mathcal{O} = \{\alpha = InverseObjectProperties(r, s),$ $\beta = SubClassOf(A, B),$ $\gamma = ObjectPropertyDomain(s, B)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta, \gamma\}$

Table A.26: Test of ObjectPropertyRange axioms

Name	`objectPropertyRangeTest`
Tested sub-concepts and axioms	ObjectPropertyRange
Input	$\mathcal{O} = \{\alpha = InverseObjectProperties(r, s),$ $\beta = SubClassOf(A, B),$ $\gamma = ObjectPropertyDomain(s, B),$ $\delta = ObjectPropertyDomain(s, B)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

The test for object property range axioms is analogous to the test for object property domain axioms (see Table A.26).

The last axioms of this section are the functional, inverse-functional, reflexive, irreflexive, symmetric, asymmetric and transitive object properties.

Functional and inverse-functional property axioms will not add an edge to G. Therefore we expect the partitioning of $\mathcal{O}_{pre}$ (of Table A.12) and $\mathcal{O}$ ($\mathcal{O}_pre$ with the addition of the corresponding axiom) to be the same. We exemplarly describe the test for functional object property axioms in Table A.27. The test for the inverse-functional property axiom is analogous.

We expect that the reflexive, irreflexive, symmetric, asymmetric and transitive object property axioms force the R_1 and R_0 of their property R into one subgraph.

Exemplary we describe the test for reflexive object property axioms. To test this

Table A.27: Test of FunctionalObjectProperty axioms

Name	`functionalObjectPropertyTest`
Tested sub-concepts and axioms	FunctionalObjectProperty
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(r, s),$ $\beta = SubObjectPropertyOf(r', s'),$ $\gamma = FunctionalObjectProperty(r)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha, \gamma\}$ and $\mathcal{O}_2 = \{\gamma\}$

Table A.28: Test of $\mathcal{O}_{pre}$ for ReflexiveObjectProperty axioms

Name	`reflexiveObjectPropertyPreTest`
Tested sub-concepts and axioms	SubObjectPropertyOf, InverseObjectProperty
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(s,t),$ $\beta = InverseObjectProperty(r,s)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta\}$

Table A.29: Test of ReflexiveObjectProperty axioms

Name	`reflexiveObjectPropertyTest`
Tested sub-concepts and axioms	ReflexiveObjectProperty
Input	$\mathcal{O} = \{\alpha = SubObjectPropertyOf(s,t),$ $\beta = InverseObjectProperty(r,s),$ $\gamma = ReflexiveObjectProperty(s)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

axiom we create

$$\mathcal{O}_{pre} = \{\alpha = InverseObjectProperty(r,s), \beta = SubObjectPropertyOf(s,t)\}$$

which we expect will result in two subgraphs, the test is described in Table A.28. We expect that adding $\gamma = ReflexiveObjectProperty(s)$ will force the subgraphs together. The test for reflexive object property axiom is described in Table A.29.

The tests of irreflexive, symmetric, asymmetric and transitive object property axioms are analogous (using the same $\mathcal{O}_{pre}$).

A.4 Test of Data Property Axioms

We expect that data subproperty, equivalent data properties and disjoint data properties axioms behave similar to subclass, equivalent classes and disjoint classes axioms. Therefore we define the tests for this axioms analougus to their corresponding class expression axioms (see Tables A.30 - A.33).

We additionally implemented tests with more than two inputs for equivalent data properties and disjoint data properties axioms. For this tests we test and create an $\mathcal{O}_{pre}$ that will result in five components (see Table A.34). The tests for equivalent classes, disjoint classes and disjoint union axioms are described in Tables A.35 and A.36.

Data property domain axioms simply connect the vertex of the data property with the vertex of the class expression. We define

$$\mathcal{O}_{pre} = \{\alpha = SubClassOf(A,B), \beta = SubDataPropertyOf(P,P')\}$$

Table A.30: Test of $\mathcal{O}_{pre}$ for SubDataPropertyOf axioms

Name	`subDataPropertyOfPropertiesPreTest`
Tested sub-concepts and axioms	SubDataPropertyOf
Input	$\mathcal{O} = \{\alpha = SubDataPropertyOf(r, r'),$ $\beta = SubDataPropertyOf(s, s')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta\}$

Table A.31: Test of SubDataPropertyOf axioms

Name	`subDataPropertyOfTest`
Tested sub-concepts and axioms	SubDataPropertyOf
Input	$\mathcal{O} = \{\alpha = SubDataPropertyOf(r, r'),$ $\beta = SubDataPropertyOf(s, s'),$ $\gamma = SubDataPropertyOf(r, s)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.32: Test of EquivalentDataProperties axioms

Name	`equivalentDataPropertiesTest`
Tested sub-concepts and axioms	EquivalentDataProperties
Input	$\mathcal{O} = \{\alpha = SubDataPropertyOf(r, r'),$ $\beta = SubDataPropertyOf(s, s'),$ $\gamma = EquivalentDataProperties(r, s)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.33: Test of DisjointDataProperties axioms

Name	`disjointDataPropertiesTest`
Tested sub-concepts and axioms	DisjointDataProperties
Input	$\mathcal{O} = \{\alpha = SubDataPropertyOf(r, r'),$ $\beta = SubDataPropertyOf(s, s'),$ $\gamma = DisjointDataProperties(r, s)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.34: Test of $\mathcal{O}_{pre}$ for EquivalentDataProperties axioms with 5 roles

Name	`equivalentDataPropertiesManyPreTest`
Tested sub-concepts and axioms	SubDataPropertyOf
Input	$\mathcal{O} = \{\alpha = SubDataPropertyOf(r, r'),$ $\beta = SubDataPropertyOf(s, s'),$ $\gamma = SubDataPropertyOf(t, t'),$ $\delta = SubDataPropertyOf(v, v'),$ $\epsilon = SubDataPropertyOf(w, w')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4, \mathcal{O}_5\}$ with $\mathcal{O}_1 = \{\alpha\}$, $\mathcal{O}_2 = \{\beta\}$, $\mathcal{O}_3 = \{\gamma\}$, $\mathcal{O}_4 = \{\delta\}$ and $\mathcal{O}_5 = \{\epsilon\}$

Table A.35: Test of EquivalentDataProperties axioms with 5 roles

Name	`equivalentDataPropertiesManyTest`
Tested sub-concepts and axioms	SubDataPropertyOf
Input	$\mathcal{O} = \{\alpha = SubDataPropertyOf(r, r'),$ $\beta = SubDataPropertyOf(s, s'),$ $\gamma = SubDataPropertyOf(t, t'),$ $\delta = SubDataPropertyOf(v, v'),$ $\epsilon = SubDataPropertyOf(w, w'),$ $\zeta = EquivalentDataProperties(r, s, t, v, w)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.36: Test of DisjointDataProperties axioms with 5 roles

Name	`disjointDataPropertiesManyTest`
Tested sub-concepts and axioms	DisjointDataProperties
Input	$\mathcal{O} = \{\alpha = SubDataPropertyOf(r, r'),$ $\beta = SubDataPropertyOf(s, s'),$ $\gamma = SubDataPropertyOf(t, t'),$ $\delta = SubDataPropertyOf(v, v'),$ $\epsilon = SubDataPropertyOf(w, w'),$ $\zeta = DisjointDataProperties(r, s, t, v, w)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.37: Test of $\mathcal{O}_{pre}$ for DataPropertyDomain axioms

Name	`dataPropertyDomainPreTest`
Tested sub-concepts and axioms	SubClassOf, SubDataPropertyOf
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, B),$ $\beta = SubDataPropertyOf(P, P')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta\}$

Table A.38: Test DataPropertyDomain axioms

Name	`dataPropertyDomainTest`
Tested sub-concepts and axioms	DataPropertyDomain
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, B),$ $\beta = SubDataPropertyOf(P, P'),$ $\gamma = DataPropertyDomain(P, A)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

The test for $\mathcal{O}_{pre}$ is described in Table A.37 and the test for the axiom is described in Table A.38.

We expect that data property range axioms and functional data propertiy axioms will not add additional connections in the constraint graph. The test (with the $\mathcal{O}_{pre}$ of Table A.37) of both axioms is described in Table A.39.

A.5 Test of Key Axioms

Key axioms will add edges between the class expression C and the vertices of the keys, i.e. R_0 for all object property expressions R and P for all data property expressions P in the axiom. To test the implementation for this axioms we create an ontology $\mathcal{O}_{pre} = \{\alpha =$

Table A.39: Test of DataPropertyRange and FunctionalDataProperty axioms

Name	`rangeFunctionalDataTest`
Tested sub-concepts and axioms	DataPropertyRange, FunctionalObjectProperty
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, B),$ $\beta = SubDataPropertyOf(P, P'),$ $\gamma = DataPropertyRange(P, dr),$ $\delta = FunctionalDataProperty(P')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta, \gamma, \delta\}$

Table A.40: Test of $\mathcal{O}_{pre}$ for Key axioms

Name	`keyPreTest`
Tested sub-concepts and axioms	SubClassOf, SubObjectProperty, SubDataPropertyOf
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, B),$ $\beta = SubObjectProperty(r, s),$ $\gamma = SubDataPropertyOf(P, P')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3\}$ with $\mathcal{O}_1 = \{\alpha\}$, $\mathcal{O}_2 = \{\beta\}$ and $\mathcal{O}_3 = \{\gamma\}$

Table A.41: Test of Key axioms

Name	`keyTest`
Tested sub-concepts and axioms	HasKey
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, B),$ $\beta = SubObjectProperty(r, s),$ $\gamma = SubDataPropertyOf(P, P'),$ $\delta = HasKey(A, (r), (P))\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

$SubClassOf(A, B), \beta = SubObjectProperty(r, s), \gamma = SubDataPropertyOf(P, P')\}$ wich will be partitioned into three partitons for each axiom (described in Table A.40). The axiom we test in Table A.41 is $\delta = HasKey(C(r)(P))$.

Additionally we tested key axioms with three object properties and three data properties as keys in Table A.42 and A.43.

A.6 Test of Assertion Axioms

Individual equality and individual inequality axioms will connect the subgraphs containing their entities. We will test it similiarly to `SubClassOf` and `EquivalentClasses`. In the first test (see Table A.44), we show that two individual equality axioms with different entities will create two subgraphs.

Now we will test in Table A.45, if the addition of the axiom $\gamma = SameIndividual(a, b)$ will force the subgraphs into one, as expected (using the $\mathcal{O}_{pre}$ of Table A.44).

Additionally we create a $\mathcal{O}_{pre}$ and a test for five individuals as entities for the same individual tests. The test for the $\mathcal{O}_{pre}$ is described in Table A.46 and the test for the axiom is described in Table A.47.

We will define analougus tests for different individual axioms. Furthermore we will test the axioms with five instead of two entities by replacing the two entities with five.

Table A.42: Test of $\mathcal{O}_{pre}$ for Key axioms with three object properties and three data properties

Name	`keyManyPreTest`
Tested sub-concepts and axioms	SubClassOf, SubObjectProperty, SubDataPropertyOf
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, B),$ $\beta = SubObjectProperty(r, r'),$ $\gamma = SubObjectProperty(s, s'),$ $\delta = SubObjectProperty(t, t'),$ $\epsilon = SubDataPropertyOf(P, P'),$ $\zeta = SubDataPropertyOf(P_2, P'_2),$ $\eta = SubDataPropertyOf(P_3, P'_3)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4, \mathcal{O}_5, \mathcal{O}_6, \mathcal{O}_7\}$ with $\mathcal{O}_1 = \{\alpha\}$, $\mathcal{O}_2 = \{\beta\}$, $\mathcal{O}_3 = \{\gamma\}$, $\mathcal{O}_4 = \{\delta\}$, $\mathcal{O}_5 = \{\epsilon\}$, $\mathcal{O}_6 = \{\zeta\}$, and $\mathcal{O}_7 = \{\eta\}$

Table A.43: Test of Key axioms with three object properties and three data properties

Name	`keyManyTest`
Tested sub-concepts and axioms	HasKey
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, B),$ $\beta = SubObjectProperty(r, r'),$ $\gamma = SubObjectProperty(s, s'),$ $\delta = SubObjectProperty(t, t'),$ $\epsilon = SubDataPropertyOf(P, P'),$ $\zeta = SubDataPropertyOf(P_2, P'_2),$ $\eta = SubDataPropertyOf(P_3, P'_3),$ $\delta = HasKey(A, (r, s, t), (P, P_2, P_3))\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.44: Test of $\mathcal{O}_{pre}$ for SameIndividual axioms

Name	`sameIndividualPreTest`
Tested sub-concepts and axioms	SameIndividual
Input	$\mathcal{O}_{pre} = \{\alpha = SameIndividual(a, a'),$ $\beta = SameIndividual(b, b')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta\}$

Table A.45: Test of SameIndividual axioms

Name	`sameIndividualTest`
Tested sub-concepts and axioms	SameIndividual
Input	$\mathcal{O}_{pre} = \{\alpha = SameIndividual(b, b'),$ $\beta = SameIndividual(a, a'),$ $\gamma = SameIndividual(a, b)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.46: Test of $\mathcal{O}_{pre}$ for SameIndividual axioms with five individuals

Name	`sameIndividualManyPreTest`
Tested sub-concepts and axioms	SameIndividual
Input	$\mathcal{O}_{pre} = \{\alpha = SameIndividual(a, a'),$ $\beta = SameIndividual(b, b'),$ $\gamma = SameIndividual(c, c'),$ $\delta = SameIndividual(d, d'),$ $\epsilon = SameIndividual(e, e')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$, $\mathcal{O}_1 = \{\beta\}$, $\mathcal{O}_1 = \{\gamma\}$, $\mathcal{O}_1 = \{\delta\}$ and $\mathcal{O}_2 = \{\epsilon\}$

Table A.47: Test of SameIndividual axioms with five individuals

Name	`sameIndividualManyTest`
Tested sub-concepts and axioms	SameIndividual
Input	$\mathcal{O}_{pre} = \{\alpha = SameIndividual(a, a'),$ $\beta = SameIndividual(b, b'),$ $\gamma = SameIndividual(c, c'),$ $\delta = SameIndividual(d, d'),$ $\epsilon = SameIndividual(e, e'),$ $\theta = SameIndividual(a, b, c, d, e)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.48: Test of $\mathcal{O}_{pre}$ for ClassAssertion axioms

Name	`classAssertionPreTest`
Tested sub-concepts and axioms	SubClassOf, SameIndividual
Input	$\mathcal{O}_{pre} = \{\alpha = SubClassOf(A, B), \beta = SameIndividual(a, b)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta\}$

Table A.49: Test of ClassAssertion axioms

Name	`classAssertionTest`
Tested sub-concepts and axioms	ClassAssertion
Input	$\mathcal{O}_{pre} = \{\alpha = SubClassOf(A, B), \beta = SameIndividual(a, b), \gamma = ClassAssertion(A, a)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Class assertions axioms create a edge between the class and the individual of the assertion. We expect the subgraphs of the class and the individual to fall together. Therefore we define and test the pre-ontology $\mathcal{O}_{pre} = \{\alpha = SubClassOf(A, B), \beta = SameIndividual(a, b)\}$ and the axiom $\gamma = ClassAssertion(A, a)$ in Table A.48 and A.49.

We expect positive and negative object property assertions with the the property R and the individuals a, b to force the subgraphs of R_0 and a but also R_1 and b together. It should also add an axiom to the partiton of a and r_0. For our test we define the "pre'-ontology in such a way, that all four vertices r_0, r_1, a and b have their own subgraph. The test for the corresponding $\mathcal{O}_{pre}$ is described in Table A.50. We expect, that adding the axiom $\epsilon = ObjectPropertyAssertion(r, a, b)$ will connect the partiton of β and δ but also the subgraphs of γ and ϵ (see Table A.51).

Table A.50: Test of $\mathcal{O}_{pre}$ for ObjectPropertyAssertion axioms

Name	`objectPropertyAssertionPreTest`
Tested sub-concepts and axioms	SubObjectPropertyOf, SameIndividual
Input	$\mathcal{O}_{pre} = \{\alpha = SubObjectPropertyOf(r, s), \beta = OWLInverseObjectProperties(t, s), \gamma = SameIndividual(a, a'), \delta = SameIndividual(b, b')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4\}$ with $\mathcal{O}_1 = \{\alpha\}$, $\mathcal{O}_2 = \{\beta\}$, $\mathcal{O}_3 = \{\gamma\}$ and $\mathcal{O}_4 = \{\delta\}$

Table A.51: Test of ObjectPropertyAssertion axioms

Name	`objectPropertyAssertionTest`
Tested sub-concepts and axioms	ObjectPropertyAssertion
Input	$\mathcal{O}_{pre} = \{\alpha = SubObjectPropertyOf(r, s),$ $\beta = SubObjectPropertyOf(s, t),$ $\gamma = SameIndividual(a, a'),$ $\delta = SameIndividual(b, b'),$ $\epsilon = ObjectPropertyAssertion(r, a, b)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha, \delta, \epsilon\}$ and $\mathcal{O}_2 = \{\beta, \gamma\}$

Table A.52: Test of $\mathcal{O}_{pre}$ for DataPropertyAssertion axioms.

Name	`dataPropertyAssertionPreTest`
Tested sub-concepts and axioms	SameIndividual, SubDataPropertyOf
Input	$\mathcal{O} = \{\alpha = SameIndividual(a, b),$ $\beta = SubDataPropertyOf(P, P')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta\}$

The test for negative object property axioms is analogous (using the same $\mathcal{O}_{pre}$).

Positive data property axioms connect the subgraph of their source individiual with their data property expression. We define and describe the test of $\mathcal{O}_{pre}$ in Table A.52 (v is a arbitary value, like the string “test”) and the test of the data property axiom in Table A.53.

The test of negative data property assertion is implemented analogous (using the same $\mathcal{O}_{pre}$).

Table A.53: Test DataPropertyAssertion axioms

Name	`dataPropertyAssertionTest`
Tested sub-concepts and axioms	DataPropertyAssertion
Input	$\mathcal{O} = \{\alpha = SameIndividual(a, b),$ $\beta = SubDataPropertyOf(P, P'),$ $\gamma = DataPropertyAssertion(P, a, v)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.54: Test of $\mathcal{O}_{pre}$ for ObjectIntersectionOf

Name	`objectInterSectionPreTest`
Tested sub-concepts and axioms	ClassAssertion
Input	$\mathcal{O} = \{\alpha = ClassAssertion(B', a),$ $\beta = ClassAssertion(B'', b)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta\}$

Table A.55: Test of ObjectIntersectionOf

Name	`objectInterSectionTest`
Tested sub-concepts and axioms	ObjectIntersectionOf
Input	$\mathcal{O} = \{\alpha = ClassAssertion(B', a),$ $\beta = ClassAssertion(B'', b),$ $\gamma = SubClassOf(B', ObjectIntersec.(B''))\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

A.7 Test of Propositional Connectives and Enumeration of Individuals

In section 8.1 of the "Structural Specification" propositional connectives and enumeration of individuals are described. This include `ObjectIntersectionOf`, `ObjectUnionOf` and `ObjectComplementOf`. All this subconcepts have in common that they will force their entities to be in the same subgraph. To test this subconcepts, we will create an ontology $\mathcal{O}_{pre}$ containing the axioms $\alpha = ClassAssertion(B', a)$ and $\beta = ClassAssertion(B'', b)$. We expect that the algorithm will create two partitons containing either $ClassAssertion(B, a)$ or $ClassAssertion(C, b)$ with $\mathcal{O}_{pre}$ as input. The test for this $\mathcal{O}_{pre}$ is described in Table A.54 For this test we add an axiom of the form $\gamma = \{SubClassOf(A, C\}$ to the ontology, with A being a new Class Expression and C being the tested subconcept, e.g. $C = ObjectIntersectionOf(B', B'')$. γ has the form $SubClassOf(B', ObjectComplementOf(B''))$ for ObjectComplementOf. The tests are described in Table A.55, A.56 and A.57.

`ObjectOneOf` is the only subconcept in Section 8.1 which entities are Individuals and not class expressions. We use the $\mathcal{O}_{pre}$ of Table A.54 and set

$$\gamma = SubClassOf(A, ObjectOneOf(a, b))$$

.

Additionally to the normal tests for object intersections, union and enumeration of individuals (ObjectOneOf) we add tests with five entities in the tested subconcept. The tests are described in Table A.59 - A.62.

Table A.56: Test of ObjectUnionOf

Name	`objectUnionOfTest`
Tested sub-concepts and axioms	ObjectUnionOf
Input	$\mathcal{O} = \{\alpha = ClassAssertion(B', a),$ $\beta = ClassAssertion(B'', b),$ $\gamma = SubClassOf(A, ObjectUnionOf(B', B''))\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.57: Test of ObjectComplementOf

Name	`objectComplementOfTest`
Tested sub-concepts and axioms	ObjectComplementOf
Input	$\mathcal{O} = \{\alpha = ClassAssertion(B', a),$ $\beta = ClassAssertion(B'', b),$ $\gamma = SubClassOf(A, ObjectCompl.(B', B''))\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.58: Test of ObjectOneOf

Name	`objectOneOfTest`
Tested sub-concepts and axioms	ObjectOneOf
Input	$\mathcal{O} = \{\alpha = ClassAssertion(B', a),$ $\beta = ClassAssertion(B'', b),$ $\gamma = SubClassOf(A, ObjectOneOf(a, b))\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.59: Test of $\mathcal{O}_{pre}$ for ObjectIntersectionOf with five classes

Name	`objectInterSectionManyPreTest`
Tested sub-concepts and axioms	ClassAssertion
Input	$\mathcal{O} = \{\alpha = ClassAssertion(A, a),$ $\beta = ClassAssertion(B, b),$ $\gamma = ClassAssertion(C, c),$ $\delta = ClassAssertion(D, d),$ $\epsilon = ClassAssertion(E, e)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4, \mathcal{O}_5\}$ with $\mathcal{O}_1 = \{\alpha\}$, $\mathcal{O}_2 = \{\beta\}$, $\mathcal{O}_3 = \{\gamma\}$, $\mathcal{O}_4 = \{\delta\}$ and $\mathcal{O}_5 = \{\epsilon\}$

Table A.60: Test of ObjectIntersectionOf with five classes

Name	`objectInterSectionOfManyTest`
Tested sub-concepts and axioms	ClassAssertion
Input	$\mathcal{O} = \{\alpha = ClassAssertion(A, a),$ $\beta = ClassAssertion(B, b),$ $\gamma = ClassAssertion(C, c),$ $\delta = ClassAssertion(D, d),$ $\epsilon = ClassAssertion(E, e),$ $\gamma = SubClassOf(A', ObjectInter.(A, B, C, D, E))\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.61: Test of ObjectUnionOf with five classes

Name	`objectUnionOfManyTest`
Tested sub-concepts and axioms	ClassAssertion
Input	$\mathcal{O} = \{\alpha = ClassAssertion(A, a),$ $\beta = ClassAssertion(B, b),$ $\gamma = ClassAssertion(C, c),$ $\delta = ClassAssertion(D, d),$ $\epsilon = ClassAssertion(E, e),$ $\gamma = SubClassOf(A', ObjectUn.(A, B, C, D, E))\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.62: Test of ObjectUnionOf with five classes

Name	`objectUnionOfManyPreTest`
Tested sub-concepts and axioms	ClassAssertion
Input	$\mathcal{O} = \{\alpha = ClassAssertion(A, a),$ $\beta = ClassAssertion(B, b),$ $\gamma = ClassAssertion(C, c),$ $\delta = ClassAssertion(D, d),$ $\epsilon = ClassAssertion(E, e),$ $\gamma = SubClassOf(A', ObjectUn.(A, B, C, D, E))\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.63: Test of $\mathcal{O}_{pre}$ for ObjectSomeValuesFrom

Name	`objectSomeValuesFromPreTest`
Tested sub-concepts and axioms	SubClassOf, ObjectPropertyDomain, InverseObjectProperties(r,r')
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, A'),$ $\beta = ObjectPropertyDomain(r, A''),$ $\gamma = InverseObjectProperties(r', r),$ $\delta = SubClassOf(B, B')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4\}$ with $\mathcal{O}_1 = \{\alpha\}$, $\mathcal{O}_2 = \{\beta\}$, $\mathcal{O}_3 = \{\gamma\}$ and $\mathcal{O}_4 = \{\delta\}$

Table A.64: Test of ObjectSomeValuesFrom

Name	`objectSomeValuesFromTest`
Tested sub-concepts and axioms	ObjectSomeValuesFrom
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, A'),$ $\beta = ObjectPropertyDomain(r, A''),$ $\gamma = InverseObjectProperties(r', r),$ $\delta = SubClassOf(B, B'),$ $\epsilon = SubClassOf(A, ObjectSomeVal.(r, A))\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha, \beta, \epsilon\}$ and $\mathcal{O}_2 = \{\gamma, \delta\}$

A.8 Test of Object Property Restrictions

Existential, universal object property quantification as well as minimum, maximum and exact object property cardinality restriction will connect subgraphs similarly: They will connect the subgraph of the sub concept C with the subgraph of R_0 for the respective property R and the subgraph of R_1 for the property R with the concept A of the sub concept (e.g. $\exists R.B$).

The axiom we want to test is $\epsilon = SubClassOf(A, ObjectSomeValuesFrom(r, B))$. To test this axiom we create a $\mathcal{O}_{pre}$ that splits into four subgraphs, each containing either A, r_0, r_1 and B. The addition of δ will force the subgraph of A and r_0 as well as the subgraphs of r_1 and B together (see Table A.63). The test for the addition of ϵ can be found in Table A.64.

The tests for universal object property quantification as well as minimum, maximum and exact object property cardinality restriction are analogous (replacing ObjectSomeValuesFrom with the corresponding axiom type).

Object property cardinality restriction without class expression are tested with the $\mathcal{O}_{pre}$ of Table A.65. Exemplarly we describe the test for minimum object property cardinality restriction in Table A.66. The test for maximum and exact object property

Table A.65: Test of $\mathcal{O}_{pre}$ for ObjectMinCardinality with no class expression

Name	`objectMinCardinalityNoClassExpressionPreT.`
Tested sub-concepts and axioms	SubClassOf, ObjectPropertyDomain
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, A'),$ $\beta = ObjectPropertyDomain(r, A'')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta\}$

Table A.66: Test of ObjectMinCardinality with no class expression

Name	`objectMinCardinalityNoClassExpressionTest`
Tested sub-concepts and axioms	SubClassOf, ObjectPropertyDomain
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, A'),$ $\beta = ObjectPropertyDomain(r, A''),$ $\epsilon = SubClassOf(A, ObjectMinCard.(r, 2))\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

cardinality restrictions are analogous.

Another form of object property restrictions are individual value restriction axioms. The tests for this axiom type is defined similiarly to the tests for existential object property quantification. The $\mathcal{O}_{pre}$ test is defined in Table A.67 and the test for the axiom type is defined in Table A.68.

Self-restriction axioms will connect R_0 and R_1 for their entity R. We define a $\mathcal{O}_{pre}$ that has two subgraphs containing either R_0 or R_1 (the test for this $\mathcal{O}_{pre}$ is described in Table A.69). We test self-restriction axioms by adding $\gamma = ObjectHasSelf(r)$ to $\mathcal{O}_{pre}$. The test is described in Table A.70.

A.9 Test of Data Property Restrictions

Existential, universal, literal data property restictions as well as minimum, maximum and exact data property cardinality restrictions all add only a connection between the subconcept C and the data property P.

The addition of a axiom of the form

$$\beta = SubClassOf(A, DataSomeValuesFrom(P, xsd : integer)$$

will only add a connection between the vertices for A, $DataSomeV.(P, xsd : integer)$ and P. The first axiom containing $DataSomeValuesFrom(P, xsd : integer)$ will add the connection between $DataSomeValuesFrom(P, xsd : integer)$ and P. Therefore the only meaningful test is, to test if the axiom β will be added to the subgraph of P. Consequently $\mathcal{O}_{pre}$ will only contain the axiom $\alpha = SubDataPropertyOf(P, P')$. The test for $\mathcal{O}_{pre}$ is

Table A.67: Test of $\mathcal{O}_{pre}$ for ObjectHasValue

Name	`objectHasValuePreTest`
Tested sub-concepts and axioms	SubClassOf, ObjectPropertyDomain, InverseObjectProperties(r,r'), SameIndividual
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, A'),$ $\beta = ObjectPropertyDomain(r, A''),$ $\gamma = InverseObjectProperties(r', r),$ $\delta = SameIndividual(b, b')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3, \mathcal{O}_4\}$ with $\mathcal{O}_1 = \{\alpha\}$, $\mathcal{O}_2 = \{\beta\}$, $\mathcal{O}_3 = \{\gamma\}$ and $\mathcal{O}_4 = \{\delta\}$

Table A.68: Test of ObjectHasValue

Name	`objectHasValueTest`
Tested sub-concepts and axioms	ObjectHasValue
Input	$\mathcal{O} = \{\alpha = SubClassOf(A, A'),$ $\beta = ObjectPropertyDomain(r, A''),$ $\gamma = InverseObjectProperties(r', r),$ $\delta = SameIndividual(b, b'),$ $\epsilon = SubClassOf(A, ObjectHasValue(r, b))\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha, \beta, \epsilon\}$ and $\mathcal{O}_2 = \{\gamma, \delta\}$

Table A.69: Test of $\mathcal{O}_{pre}$ for ObjectHasSelf

Name	`objectHasSelfPreTest`
Tested sub-concepts and axioms	ObjectPropertyDomain, InverseObjectProperties
Input	$\mathcal{O} = \{\alpha = ObjectPropertyDomain(r, A),$ $\beta = InverseObjectProperties(r', r)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}_1, \mathcal{O}_2\}$ with $\mathcal{O}_1 = \{\alpha\}$ and $\mathcal{O}_2 = \{\beta\}$

Table A.70: Test of ObjectHasSelf

Name	`objectHasSelfTest`
Tested sub-concepts and axioms	ObjectHasSelf
Input	$\mathcal{O} = \{\alpha = ObjectPropertyDomain(r, A),$ $\beta = InverseObjectProperties(r', r),$ $\gamma = ObjectHasSelf(r)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

described in Table A.71 and the description of the test for the addition of β can be found in Table A.72.

Table A.71: Test of $\mathcal{O}_{pre}$ for DataSomeValuesFrom

Name	`dataSomeValuesFromPreTest`
Tested sub-concepts and axioms	SubDataPropertyOf
Input	$\mathcal{O} = \{\alpha = SubDataPropertyOf(P, P')\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

Table A.72: Test of DataSomeValuesFrom

Name	`dataSomeValuesFromTest`
Tested sub-concepts and axioms	DataSomeValuesFrom
Input	$\mathcal{O} = \{\alpha = SubDataPropertyOf(P, P'),$ $\beta = SubClassOf(A, DataSomeV.(P, xsd : int.)\}$
Expected Output	$\mathbb{O} = \{\mathcal{O}\}$

All other data property restriction tests are analogous (using the same $\mathcal{O}_{pre}$).

A.10 Test of datatype definition

The handling of datatype definition axioms is a special case. It will not add additional connections into the constraint graph, therefore we will not define additional tests for this axiom type

A.11 Tests with Top as Input

Additional special case tests for Top and Bottom (in OWL they are called owl:Thing and owl:Nothing) as input were implemented. For this tests we take the previous described tests, with the exception of the “Many”-tests, and replace the inputs with owl:Thing or owl:Nothing. The goal of this tests is to see if the implementation can handle this cases (without throwing an exception) and if they are handled correctly.

This tests are expecially necessary, because we will often not create an extra vertex for Top or Bottom if they are not necessary. The only exception to this are key axioms.

A.12 Results

The test results can be found in Table A.73. All tests were succesfull.

Table A.73: Test Results

Name of Test	Duration	Success
Class: RunThroughExampleTest		
runThroughExampleTest()	16ms	✓
Class: ClassExpressionAxiomsTest		
subClassOfPreTest()	6ms	✓
subClassOfTest()	2ms	✓
equivalentClassesTest()	7ms	✓
disjointClassesTest()	3ms	✓
disjointUnionPreTest()	9ms	✓
disJointUnionTest()	5ms	✓
equivalentClassesManyPreTest()	10ms	✓
equivalentClassesManyTest()	6ms	✓
disjointClassesManyTest()	4ms	✓
disjointUnionManyTest()	5ms	✓
Class: ObjectPropertyAxiomsTest		
subObjectPropertyOfPreTest()	19ms	✓
subObjectPropertyOfTest()	7ms	✓
subObjectPropertyChainPreTest()	17ms	✓
subObjectPropertyChainTest()	13ms	✓
equivalentObjectPropertiesTest()	13ms	✓
disjointObjectPropertiesTest()	3ms	✓
equivalentObjectPropertiesManyPreTest()	15ms	✓
equivalentObjectPropertiesManyTest()	3ms	✓
disjointObjectPropertiesManyTest()	28ms	✓
inverseObjectPropertiesPreTest()	6ms	✓
inverseObjectPropertiesTest()	3ms	✓
objectPropertyDomainPreTest()	14ms	✓
objectPropertyDomainTest()	3ms	✓
objectPropertyRangeTest()	5ms	✓
functionalObjectPropertyTest()	29ms	✓
reflexiveObjectPropertyPreTest()	6ms	✓
reflexiveObjectPropertyTest()	10ms	✓
irreflexiveObjectPropertyTest()	10ms	✓
symmetricObjectPropertyTest()	12ms	✓
asymmetricObjectPropertyTest()	17ms	✓
transitiveObjectPropertyTest()	21ms	✓
Class: DataPropertyAxiomsTest		
subDataPropertyOfPreTest()	34ms	✓
subDataPropertyOfTest()	10ms	✓
equivalentDataPropertiesTest()	8ms	✓
disjointDataPropertiesTest()	14ms	✓
dataPropertyDomainPreTest()	17ms	✓

dataPropertyDomainTest()	30ms	✓
rangeFunctionalDataTest()	40ms	✓
equivalentDataPropertiesManyPreTest()	32ms	✓
equivalentDataPropertiesManyTest()	281ms	✓
disjointDataPropertiesManyTest()	30ms	✓
Class: KeyAxiomsTest		
keyPreTest()	3ms	✓
keyTest()	2ms	✓
keyManyPreTest()	20ms	✓
keyManyTest()	8ms	✓
Class: AssertionAxiomsTest		
sameIndividualPreTest()	14ms	✓
sameIndividualTest()	7ms	✓
differentIndividualsPreTest()	17ms	✓
differentIndividualsTest()	5ms	✓
classAssertionPreTest()	14ms	✓
classAssertionTest()	23ms	✓
objectPropertyAssertionPreTest()	29ms	✓
objectPropertyAssertionTest()	23ms	✓
dataPropertyAssertionPreTest()	7ms	✓
dataPropertyAssertionTest()	34ms	✓
negativeObjectPropertyAssertionTest()	80ms	✓
negativeDataPropertyAssertionTest()	18ms	✓
sameIndividualManyPreTest()	14ms	✓
sameIndividualManyTest()	12ms	✓
differentIndividualsManyTest()	5ms	✓
Class: ConnectivesAndEnumerationTest		
objectIntersectionPreTest()	8ms	✓
objectIntersectionTest()	4ms	✓
objectUnionTest()	2ms	✓
objectComplementOfTest()	5ms	✓
objectOneOfTest()	2ms	✓
objectIntersectionManyPreTest()	14ms	✓
objectIntersectionManyTest()	6ms	✓
objectUnionManyTest()	6ms	✓
objectOneOfManyTest()	23ms	✓
Class: ObjectPropertyRestrictionTest		
objectSomeValuesFromPreTest()	11ms	✓
objectSomeValuesFromTest()	7ms	✓
objectAllValuesFromTest()	18ms	✓
objectMinCardinalityTest()	17ms	✓
objectMaxCardinalityTest()	4ms	✓
objectExactCardinalityTest()	18ms	✓
objectMinCardinalityNoClassExpressionPreTest(	9ms	✓

objectMinCardinalityNoClassExpressionTest()	3ms	✓
objectMaxCardinalityNoClassExpressionTest()	12ms	✓
objectExactCardinalityNoClassExpressionTest()	22ms	✓
objectHasValuePreTest()	16ms	✓
objectHasValueTest()	21ms	✓
objectHasSelfPreTest()	2ms	✓
objectHasSelfTest()	5ms	✓
Class: DataPropertyRestrictionTest		
dataSomeValuesFromPreTest()	1ms	✓
dataSomeValuesFromTest()	12ms	✓
dataAllValuesFromTest()	12ms	✓
dataMinCardinalityTest()	2ms	✓
dataMaxCardinalityTest()	4ms	✓
dataExactCardinalityTest()	11ms	✓
dataMinCardinalityTestNoDataRange()	6ms	✓
dataMaxCardinalityTestNoDataRange()	13ms	✓
dataExactCardinalityTestNoDataRange()	2ms	✓
dataHasValueTest()	7ms	✓
Class: TopClassExpressionAxiomsTest		
subClassOfWithTopSubTest()	11ms	✓
subClassOfWithTopSuperTest()	5ms	✓
equivalentClassesWithTopTest()	8ms	✓
equivalentClassesWithTopTest2()	3ms	✓
disjointClassesWithTopTest()	8ms	✓
disJointUnionWithTopInUnionTest()	3ms	✓
disJointUnionWithTopAsClassTest()	5ms	✓
Class: TopObjectPropertyAxiomsTest		
objectPropertyRangeWithTopTest()	20ms	✓
objectPropertyDomainWithTopTest()	17ms	✓
Class: TopDataPropertyAxiomsTest		
dataPropertyDomainWithTopTest()	9ms	✓
Class: TopKeyAxiomsTest		
keyWithTopTest()	30ms	✓
Class: TopAssertionAxiomsTest		
classAssertionWithTopTest()	10ms	✓
Class: TopConnectivesAndEnumerationTest		
objectComplementOfWithTopTest()	4ms	✓
objectUnionWithTopTest()	7ms	✓
objectIntersectionWithTopTest()	5ms	✓
Class: TopObjectPropertyRestrictionTest		
objectAllValuesFromWithTopTest()	12ms	✓
objectSomeValuesFromWithTopTest()	11ms	✓
objectMinCardinalityWithTopTest()	11ms	✓
objectMaxCardinalityWithTopTest()	4ms	✓

objectExactCardinalityWithTopTest()	23ms	✓
Class: BottomClassExpressionAxiomsTest		
subClassOfWithBottomSubTest()	14ms	✓
subClassOfWithBottomSuperTest()	13ms	✓
equivalentClassesWithBottomTest()	32ms	✓
equivalentClassesWithBottomTest2()	31ms	✓
disjointClassesWithBottomTest()	22ms	✓
disJointUnionWithBottomAsClassTest()	18ms	✓
disJointUnionWithBottomInUnionTest()	67ms	✓
Class: BottomObjectPropertyAxiomsTest		
objectPropertyDomainWithBottomTest()	14ms	✓
objectPropertyRangeWithBottomTest()	12ms	✓
Class: BottomDataPropertyAxiomsTest		
dataPropertyDomainWithBottomTest()	10ms	✓
Class: BottomKeyAxiomsTest		
keyWithBottomTest()	7ms	✓
Class: BottomAssertionAxiomsTest		
classAssertionWithBottomTest()	13ms	✓
Class: BottomConnectivesAndEnumerationTest		
objectComplementOfWithBottomTest()	23ms	✓
objectIntersectionWithBottomTest()	21ms	✓
objectUnionWithBottomTest()	16ms	✓
Class: BottomObjectPropertyRestrictionTest		
objectSomeValuesFromWithBottomTest()	23ms	✓
objectMinCardinalityWithBottomTest()	29ms	✓
objectMaxCardinalityWithBottomTest()	20ms	✓
objectAllValuesFromWithBottomTest()	22ms	✓
objectExactCardinalityWithBottomTest()	27ms	✓

Appendix B

Constraint Graphs

B.1 Constraint Graph of Koala

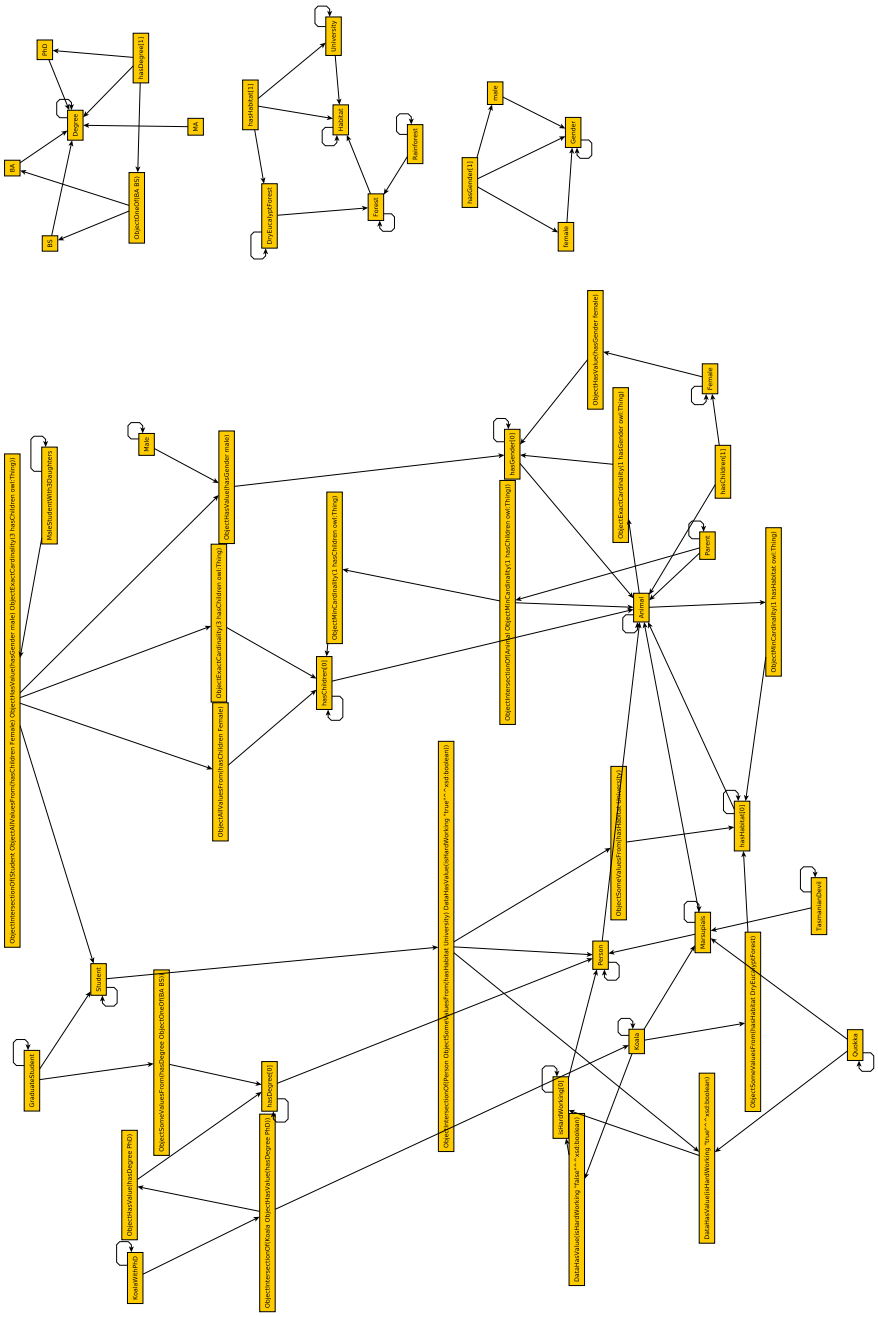

Fig. B.1: Constraint Graph of the Koala Ontology

B.2 Constraint Graph of the modified Koala by Vescovo et al. (2019)

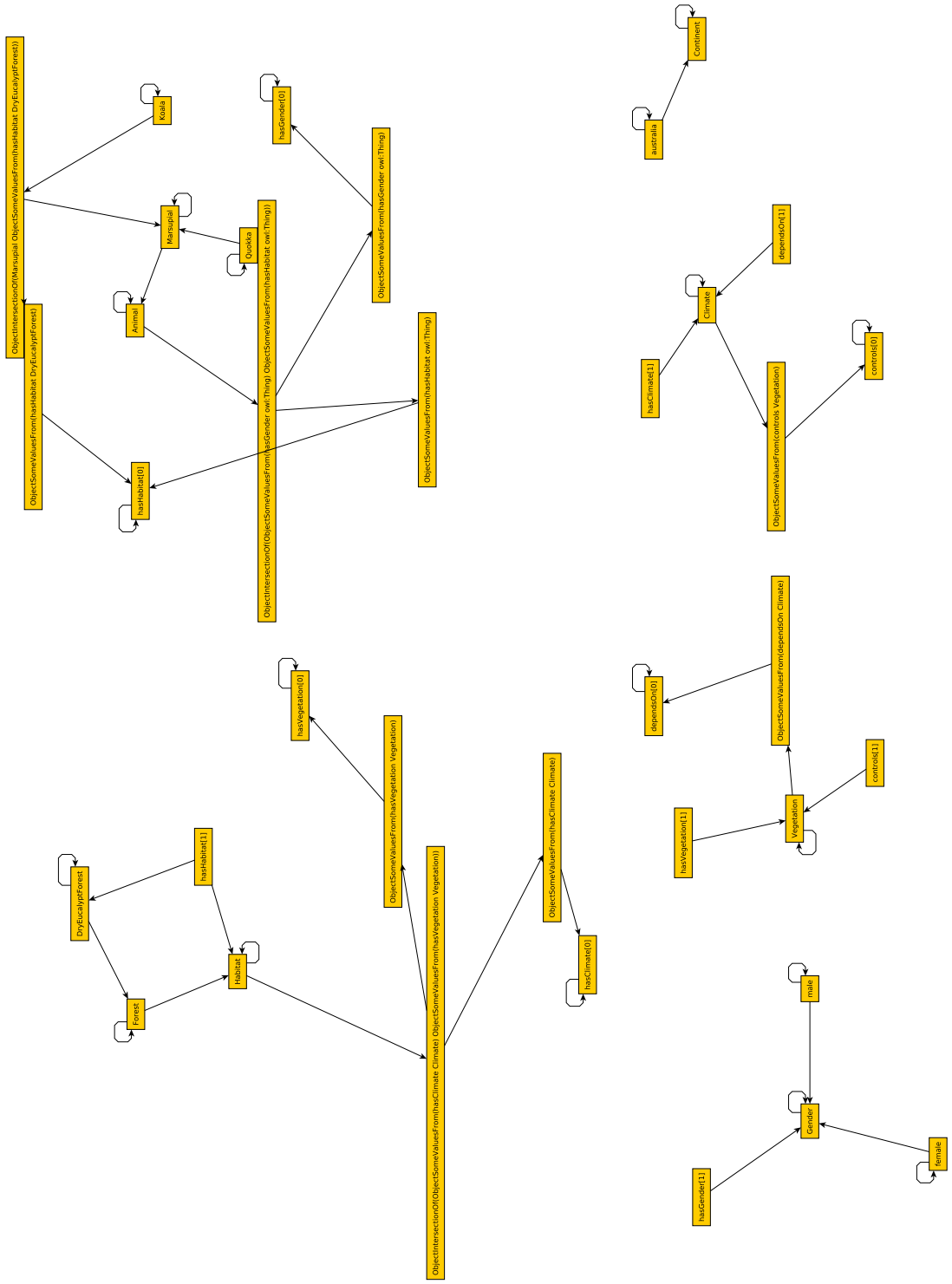

Fig. B.2: Constraint Graph of the Koala Ontology