Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

DL Abduction API

Diploma Thesis

2022
Bc. Zuzana Hlávková

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

DL Abduction API

Diploma Thesis

Study Programme:	Applied Computer Science
Field of Study:	Computer Science
Department:	Department of Applied Informatics
Supervisor:	doc. RNDr. Martin Homola, PhD.

Bratislava, 2022
Bc. Zuzana Hlávková

53979371

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Zuzana Hlávková
Študijný program:	aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: DL Abduction API
API pre abdukciu v deskripčných logikách

Anotácia: Existuje viacero abduktívnych inferenčných strojov pre DL (napr. AAA, MXP-MHS, MergeXplain, QuickXplain). Ich integrácia do rôznych praktických nástrojov je však zatiaľ len na úrovni ad hoc. Neexistuje API rozhranie pre ich modulárnu integráciu, podobné ako napr. OWL API, ktoré možno považovať za zlatý štandard pre integráciu deduktívnych DL inferenčných strojov.

Cieľ: 1. Vyabstrahovať funkcionalitu abduktívnych inferenčných strojov pre DL.
2. Navrhnúť a implementovať knižnicu pre API rozhranie pre integráciu abuduktívnej inferencie do iných softvérov.
3. Implementovať navrhnuté API rozhranie do jedného inferenčného stroja a do jedného vybraného nástroja.

Literatúra: 1. Pukancová, J. and Homola, M., 2020. The AAA ABox Abduction Solver. Künstliche Intelligenz 34(4)
2. Shchekotykhin, K., Jannach, D., Schmitz, T., 2015. MergeXplain: Fast computation of multiple conflicts for diagnosis. In IJCAI
3. Homola, M., Pukancová, J., Gablíková, J., Fabianová, K., 2020. Merge, Explain, Iterate. In: DL
4. Schekotihin, K., Rodler, P. and Schmid, W., 2018. Ontodebug: Interactive ontology debugging plug-in for protégé. In FoIKS
5. Horridge, M., Bechhofer, S., 2011. The OWL API: A Java API for OWL ontologies. Semantic Web 2(1):11-21

Vedúci:	doc. RNDr. Martin Homola, PhD.
Konzultant:	Mgr. Júlia Pukancová, PhD.
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	prof. Ing. Igor Farkaš, Dr.
Dátum zadania:	11.12.2020

Dátum schválenia: 11.12.2020 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

....
študent vedúci práce

53979371

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Bc. Zuzana Hlávková
Study programme:	Applied Computer Science (Single degree study, master II. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Diploma Thesis
Language of Thesis:	English
Secondary language:	Slovak

Title:	DL Abduction API
Annotation:	While multiple DL abduction solvers have been developed (e.g. AAA, MXP-MHS, MergeXplain, QuickXplain) their integration into various practical tools so far has been ad hoc. There is yet no DL abduction API that would enable such integration to be modular, similar to the well known OWL API that is now the golden standard for integration of deductive DL reasoners.
Aim:	1. Synthesize functionalities and capabilities of different DL abduction solvers. 2. Develop an API interface library for integrating abduction slovers into other software. 3. Implement the API in one solver and one other tool using the solver.
Literature:	1. Pukancová, J. and Homola, M., 2020. The AAA ABox Abduction Solver. Künstliche Intelligenz 34(4) 2. Shchekotykhin, K., Jannach, D., Schmitz, T., 2015. MergeXplain: Fast computation of multiple conflicts for diagnosis. In IJCAI 3. Homola, M., Pukancová, J., Gablíková, J., Fabianová, K., 2020. Merge, Explain, Iterate. In: DL 4. Schekotihin, K., Rodler, P. and Schmid, W., 2018. Ontodebug: Interactive ontology debugging plug-in for protégé. In FoIKS 5. Horridge, M., Bechhofer, S., 2011. The OWL API: A Java API for OWL ontologies. Semantic Web 2(1):11-21

Supervisor:	doc. RNDr. Martin Homola, PhD.	
Consultant:	Mgr. Júlia Pukancová, PhD.	
Department:	FMFI.KAI - Department of Applied Informatics	
Head of department:	prof. Ing. Igor Farkaš, Dr.	
Assigned:	11.12.2020	
Approved:	11.12.2020	prof. RNDr. Roman Ďurikovič, PhD. Guarantor of Study Programme

....
Student   Supervisor

Contents

Chapter 1

Introduction

Part I

State of the art

Chapter 2

Ontologies and Description Logics

In the next chapter, we will introduce the concept ontology in computer science and the description logics.

2.1 Ontologies

In this section, we will focus on the knowledge needed to understand ontologies. We are not refering to an ontology known from philosophy, but to an ontology within the computer sciences. It is a comprehensive formal description of concepts (in ontology called classes), the properties of these concepts, their instances, and their interrelationships in a particular domain. Ontology contains a taxonomy of classes and derivation rules by which new truths can be derived from known facts. The descriptions are precise so as to avoid misunderstandings in human communication and to ensure that the software that works with them does not behave unpredictably and works well with other software. Ontologies give a semantics between these concepts. Thanks to ontologies, it is possible to link data, reasonably analyze it and pass it on to other people, who can disseminate it and use it in their ontological domains. Working with them is simplified by the fact that they can be represented in semantic graphs, which can be used for presentation and easier understanding for the public.[10] [11] [2]

As an example, let's take the ontology used in the well-known [5] tutorial. So let's look at the pizza domain. In the image 2.1 we can see a sample semantic graph representing just part of this entire ontology. The ovals represent the classes, the solid arrows point to the subclasses, and the solid arrows indicate the relationship between the classes. We see that there is one relationship used multiple times in the picture. It's the `hasTopping` relationship, which deals with the fact that a pizza contains some topping (we understand the

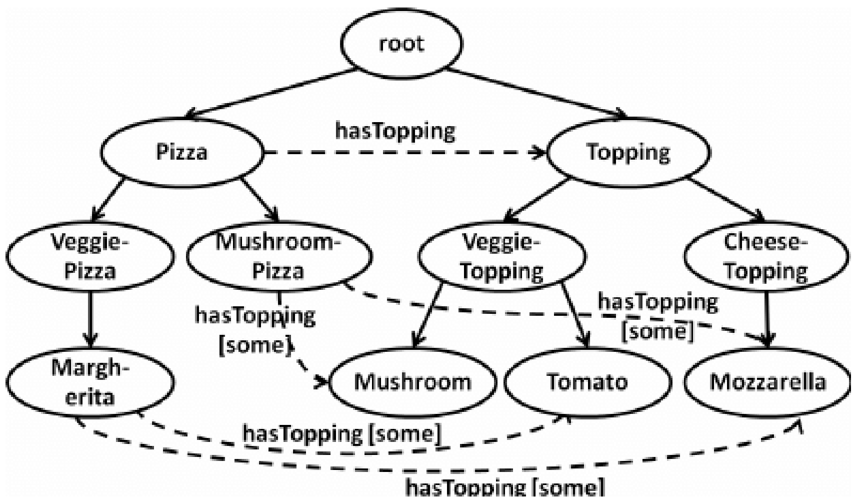

Figure 2.1: Ontology pizza example [7]

ingredient). We may also notice that in some cases `hasTopping` carries the English word `some` (translated as *some*) in square brackets. This indicates an existential condition that states that the class contains at least one toping. In cases where `[some]` is not present, it is not a condition that the pizza has any topping. The main entities in this image are pizza and topping, from which their subclasses derive (using solid arrows). Pizza can be vegetarian or mushroom. Topping can also be vegetarian or raw. This ontology lists margarita pizza as a subclass of vegetarian pizza, to which we already assign the property of having at least one tomato topping

```
Margherita hasTopping [some] Tomato
```

and at least one mozzarella toping

```
Margherita hasTopping [some] Mozzarella.
```

So, in order for pizza to become a margarita, its ingredients must include mozzarella cheese and tomato. There is also a rule that mushroom pizza must carry mushroom and mozzarella topping

```
Mushroom Pizza hasTopping [some] Mushroom
```

and

```
Mushroom Pizza hasTopping [some] Mozzarella.
```

In general, any pizza can have any of the toppings shown in the picture. This was a brief demonstration of how to imagine such an ontology and how to easily present it to the general public.

2.2 Description Logics

Description Logics (DLs) is an approach to represent and work with knowledge in a formal logic-based way and this family of logics is one of the main knowledge representation formalisms.

Knowledge representation is an part of artificial intelligence which is dedicated to representation of information we have about the world in an unique formal logic-based form.

These logics rechead an huge importance in past few years because they become a base form for the most expressive ontology languages which are the well known OWL family uses as Web Onology Lanuages.

They represent concepts, roles, individuals, and relationships between them. An axiom is an elemental notion of modeling a relationship between concepts and roles. Comparing them to other formalisms like first-order logic they are less expressive, on the other hand they bring an enormous difference, they are decidable. We can say that to call some formalism a DL, decidability is neccesary to be achievable.
Concept decidability means that there is a general way (an generic algorithm) to compute an answer yes or no to any problem of the class which is called decidable.[1]. Information we will provide below will be according to Foundation of Description Logics from author Sebastian Rudolph. [14].

2.2.1 Syntax

Let us take a look at some syntax details of DLs. Like we metioned before DL vocabulary contains three main sets:

Definition 1 ***(DL vocabulary)*** *Sets of:*

1. *individuals* $N_I = \{a, b, ...\}$
2. *atomic concepts* $N_C = \{A, B, ...\}$
3. *roles* $N_R = \{R, S, ...\}$

There exists specific concepts:

- $\top$: Top concept which includes all individuals.
- $\bot$: Bottom concept which is en empty concept.

Definition 2 ***(Top and bottom concepts)*** *The top* ($\top$) *and bottom* ($\bot$) *concepts are defined as syntactic abbreviation:*

- $(\top)$ *is a placeholder for* $A \sqcup \neg A$*;*
- $(\bot)$ *is a placeholder for* $A \sqcap \neg A$*;*

where A *is any atomic concept.*

We can see some important parts of syntax and semantics of concept constructors at the table 2.1, where C and D are concepts.

Name	Syntax	Symbol
Top	$\top$	87837
Bottom	$\bot$	78
Intersection	$\mathcal{C} \sqcup \mathcal{D}$	778
Union	$\mathcal{C} \sqcap \mathcal{D}$	18744
Negation	$\neg\mathcal{C}$	788
Value restriction	$\forall\mathcal{R}.\mathcal{C}$	788
Existential quant.	$\exists\mathcal{R}.\mathcal{C}$	788

Table 2.1: opis

Theory of desciption logic is well known as knowledge base (KB). This knowledge base is divided into two main parts:

- Assertional part:
  - ABox
- Terminological part:
  - TBox
  - RBox

In particular we can introduce a few examples of definitions to approach those parts.
ABox is an assertional knowledge which represents invididuals in concrete situations. It is related directly do data in a database.

Definition 3 ***(ABox)*** *An ABox* $\mathcal{A}$ *is a finite set of assertion axioms* ϕ *of the form:*

$$\phi ::= a : C \mid a, b : R$$

where $a, b \in N_I, R \in N_R$, and C is any concept.

In the definition above we can see that an assertion can be of both type concept assertion and role assertion.

Definition 4 ***(ABox assertions)*** *Assertions forms:*

- *C(a) | concept assertion*
- *r(a, b) | role assertion*
- *¬r(a, b) | negated role assertion*
- *$a \approx b$ | equality statement*
- *$a \not\approx b$ | inequality statement*

where $a, b \in N_I$ invididual names, $R \in N_R$ role and C is any concept.

ABox concists concept assertions like:

Example 1 ***(ABox concept assertion)***

Parent(jane)

where Parent is concept and jane is individual. This axiom tells us that an invididual called jane is part of the set off all parents.

Example 2 ***(ABox role assertion)***

sibilings(jane, adam)

where sibilings is a role and jane and adam are individuals. This axiom indicates that jane and adam are invididuals in the relation od being sibilings.

On the other hand TBox concists universal statements which describe properties of roles and concepts. It is related to the schema of database.

Definition 5 ***(TBox)*** *A TBox $\mathcal{T}$ is a finite set of GCI (general concept inclusion) axioms ϕ of the form:*

$$\phi ::= C \sqsubseteq D$$

where C, D are concepts.

Example 3 ***(Simple TBox axiom)*** *A very simple exmaple of TBox axiom is an subsumption:*

$$Parent \sqsubseteq Child$$

where Parent and Child are concept. Axiom is indicating that every parent is a child.

Example 4 ***(Complex TBox axiom)*** *More comples TBox axiom:*

$$Parent \equiv Human \sqcap (\exists hasChild.Human)$$

where Parent and Human are concepts and hasChild is a role. It tells us that Parent is someone who is human and at the same time has at list one child who is human too.

And at the and RBox which allows roles.

Definition 6 ***(RBox)*** *An RBox $\mathcal{R}$ is a finite set of RIAs (role inclusion axioms) ϕ of the form:*

$$\phi ::= R \sqsubseteq S$$

where R, S are roles.

Example 5 ***(RBox axiom)*** *RBox:*

$$married \sqsubseteq loves$$

where married and loves are roles. It tells us that everybody who is married to somebody is loving them.

In this paper we will use description logic $\mathcal{ALC}$ which is the basic DL ($\mathcal{AL}$) and its extension $\mathcal{C}$. Extension $\mathcal{C}$ adds to $\mathcal{AL}$ a new concept constructor a complement operator ($\neg$) which allows concept a negation. [1] [14] DL $\mathcal{ALC}$ means attributive language with complements. [15]

Let C and D be concepts and R a role.

Expressivity of the basic DL ($\mathcal{AL}$) allows:

$$\begin{gathered}
\mathcal{C}, \mathcal{D} \longrightarrow \mathcal{A} \mid \text{(atomic concept)} \\
\top \mid \text{(universal concept)} \\
\bot \mid \text{(bottom concept)} \\
\neg\mathcal{A} \mid \text{(atomic negation)} \\
\mathcal{C} \sqcap \mathcal{D} \mid \text{(intersection)} \\
\forall \mathcal{R}.\mathcal{C} \mid \text{(value restriction)} \\
\exists \mathcal{R}.\top \mid \text{(limited existential quantification).}
\end{gathered}$$

Let us have a proper definition for ALC allowing those concept constructors and creating a comeplex concepts with them:

Definition 7 ***(Complex concepts)*** *Concepts are recursively constructed as the smallest set of expressions of the forms:*

$$C, D ::= A \mid \neg C \mid C \sqcup D \mid C \sqcap D \mid \exists R.C \mid \forall R.C \mid \{a\}$$

where $A \in N_C$, $R \in N_R$, and C, D are concepts.

Some of complex concepts examples are:

- ¬Human
- Human ⊔ Animal
- Parent ≡ Human ⊓ (∃hasChild.Human)

2.2.2 Semantics

Now we have explained important parts of DL syntax. Let us have a look at semantics.

Semantics of any logic has at least one thing at common. Its definitions determine whether an axiom logically follows from a given set of axioms. Now we can introduce interpretations which we can consider to be "worlds".

Definition 8 ***(Interpretation)*** *An interpretation of a given DL KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is a pair $\mathcal{I} = (\Delta^I, \cdot^I)$ which contains:*

- *a domain $\Delta^I \neq \emptyset$ or also universe of discourse which can be understood as all of individuals or things existing in the "world" that interpretation represents,*
- *an interpretation function $\cdot^I$ which connects the vocabulary elements to $\cdot^{\mathcal{I}}$, by providing:*

  $a^I \in \Delta^I$ *for all* $a \in N_I$

  $A^I \subseteq \Delta^I$ *for all* $A \in N_C$

  $R^I \subseteq \Delta^I \times \Delta^I$ *for all* $R \in N_R$

- *and for any C , D and R, the interpretation of complex concepts is recursively defined as follows:*

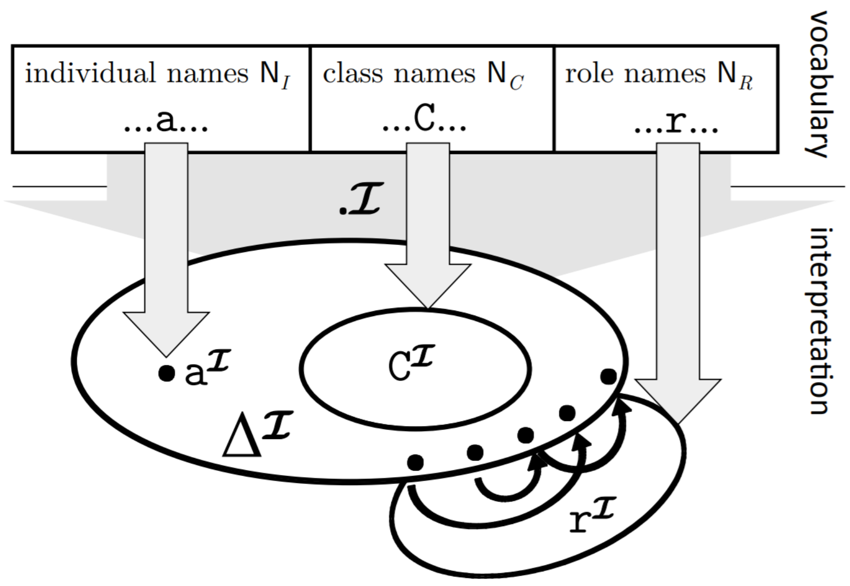

Figure 2.2: Structure of DL interpretations [14]

$$\neg C^I = \Delta^I \setminus C^I$$

$$C \sqcap D^I = C^I \cap D^I$$

$$C \sqcup D^I = C^I \cup D^I$$

$$\exists R.C^I = x \in \Delta^I \mid \exists y \in \Delta^I : \ \langle x, y \rangle \in R^I \wedge y \in C^I$$

$$\forall R.C^I = x \in \Delta^I \mid \forall y \in \Delta^I : \ \langle x, y \rangle \in R^I \Rightarrow y \in C^I$$

At figure 2.2 we can see interpretation defitnition in graphical form.

Term satisfaction of axioms means that we have given an interpretation and a given set of axioms and its determine whether the axiom is true with respect to that interpretation. Better definition can be seen below.

Definition 9 ***(Satisfaction*** $\models$***)*** *Given an axiom ϕ, an interpretation $\mathcal{I} = (\Delta^I, \cdot^I)$ satisfies ϕ ($\mathcal{I} \models \phi$) depending on its type:*

$$C \sqsubseteq D : \mathcal{I} \models C \sqsubseteq D \textit{ iff } C^I \subseteq D^I$$

$$a : C : \mathcal{I} \models a : C \textit{ iff } a^I \in C^I$$

$$a, b : R : \mathcal{I} \models a, b : R \textit{ iff } \langle a^I, b^I \rangle \in R^I$$

Definition 10 ***(Model)*** *An interpretation $\mathcal{I} = (\Delta^I, \cdot^I)$ is a model of a DL KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ iff $\mathcal{I}$ satisfies every axiom in $\mathcal{T}$ and $\mathcal{A}$.*

Definition 11 ***(Knowledge base equivalence)*** *Knowledge bases $\mathcal{K}\infty$ and $\mathcal{K}\in$ are equivalent iff each interpretation $\mathcal{I}$ is a model of $\mathcal{K}\infty$ iff $\mathcal{I}$ is a model of $\mathcal{K}\in$.*

Lemma 1 ***(Top and bottom semantics)*** *In any interpretation $\mathcal{I}$, $\top^{\mathcal{I}} = \Delta^{I}$ and $\bot^{\mathcal{I}} = \emptyset$.*

2.2.3 Decision Problems

Thinking of decision problems we refer to problems where yes and no are the answers.

Definition 12 ***((Decision Problems)*** *Given a DL KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, and two concepts $\mathcal{C}, \mathcal{D}$ we say that:*

- *C is satisfiable w.r.t. $\mathcal{K}$ iff there is a model $\mathcal{I}$ of $\mathcal{K}$ s.t. $\mathcal{C}^{\mathcal{I}} \neq \emptyset$;*
- *C is subsumed by D w.r.t. $\mathcal{K}$ (denoted $\mathcal{K} \models \mathcal{C} \sqsubseteq \mathcal{D}$) iff $\mathcal{C}^{\mathcal{I}} \subseteq \mathcal{D}^{\mathcal{I}}$ in every model $\mathcal{I}$ of $\mathcal{K}$;*
- *C and D are equivalent w.r.t. $\mathcal{K}$ (denoted $\mathcal{K} \models \mathcal{C} \sqsubseteq \mathcal{D}$) iff $\mathcal{C}^{\mathcal{I}} = \mathcal{D}^{\mathcal{I}}$ in every model $\mathcal{I}$ of $\mathcal{K}$;*
- *C and D are disjoint w.r.t. $\mathcal{K}$ iff $\mathcal{C}^{\mathcal{I}} \cap \mathcal{D}^{\mathcal{I}} = \emptyset$ in every model $\mathcal{I}$ of $\mathcal{K}$;*

In this case if $\mathcal{K} = \emptyset$ then satisfiability, subsumption, equivalence, and disjointness of concepts are defined in general by the definition. In such a case we omit "$\mathcal{K} \models$" from the notation.

Some reduction lemmata:

Lemma 2 *Given a DL KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and a concept C: C is satisfiable w.r.t. $\mathcal{K}$ iff $\mathcal{K} \not\models C \sqsubseteq \bot$.*

Lemma 3 *Given a DL KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and concepts C, D: $\mathcal{K} \models C \sqsubseteq D$ iff C $\sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{K}$.*

Lemma 4 *Given a DL KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and concepts C, D: $\mathcal{K} \models C \equiv D$ iff both $\mathcal{K} \models C \sqsubseteq D$ and $\mathcal{K} \models D \sqsubseteq C$.*

Lemma 5 *Given a DL KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and concepts C, D: C and D are disjoint w.r.t. $\mathcal{K}$ iff $\mathcal{C} \sqcap \mathcal{D}$ is unsatisfiable w.r.t. $\mathcal{K}$.*

Lemma 6 *Given a DL KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, an individual a and a concept $C : \mathcal{K} \models$ C(a) iff $\mathcal{K}$' = ($\mathcal{T}$, $\mathcal{A} \cup \{\neg C(a)\}$) is inconsistent.*

Lemma 7 *Given a DL KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, and some concept C : C is satisfiable w.r.t. $\mathcal{K}$ iff $\mathcal{K}$' = ($\mathcal{T}$, $\mathcal{A} \cup \{C(a)\}$) is consistent, for some new individual a not appearing in $\mathcal{K}$.*

Chapter 3

Ontology decision tasks

In this chapter we will focus at reasoning tasks used in ontologies. We will derive information from a very important article from Peirce [12].

There exist different types of reasoning. We will introduce deduction, the most known reasoning task, induction and abduction, which will be the main reasoing task for us. We will discuss it better later.

3.1 Deduction

The most known reasoning task is deduction. Is well known for public too, because is part of our every-day lifes. People are used to use this term in bad way. They just say deduction instead of reason, but deduction has its own rules, just like other reasoing tasks.
Let's say that there is a restaurant someone likes to go, because all of meals they made are good and she likes them. Imagine any of this meals. In that moment that person can say that that meal is good, becuase she used deduction and she deriveted a result.

Example 6 ***(Deduction)***

- ***Rule****: All the balls in this box are black.*
- ***Case:*** *These balls are from this box.*
- ***Derive result:*** *These balls are black. [12]*

3.2 Induction

Inductive reasoning is when we generalize from an amount of true cases (facts) and generate hypotheses that the same thing is true of a whole class of these cases. Or, there is an option to find a concrete thing to be a true of a concrete part of cases and infer that it is true of the same part of the whole class. It is important to mention that inductive reasoning is not truth-preserving. Generated hypotheses may be falsified.

Example 7 ***(Induction)***

- ***Case:*** *These balls are from this box.*
- ***Result:*** *These balls are black.*
- ***Derive rule:*** *All the balls in this box are black.*

As we can see at the example above there is a case that there are some balls from concrete box and they are black. With induction we derive a rule that all the balls in this box are black, but that could be falsified, because we did not mention if there are any other balls in that box or just ones we are talking about.

Definition 13 ***(Induction)*** *Let Γ and $\Delta^+ = \Delta^+ \uplus \Delta^-$ be sets of formulae in some language $\mathcal{L}$. A set of formulae ϕ is an inductive generalization of Δ (with background theory Γ) if:*

- $\Gamma \not\models \Delta^+$
- $\Gamma \cup \Delta$ *is consistent*
- $\Gamma \cup \phi \models \Delta^+$
- $\Gamma \cup \phi \cup \Delta^-$ *is consistent*

where Δ^+ are positive observations ands Δ^- are negative observations.

3.3 Abduction

The most important reasoning type for us is abduction also called hypothesis. It is reasoning in which we have some very peculiar conditon/circumstance. We find an explanation that it was a case of a certain general rule. Thanks to that explanation we can say that abduction is “premise guessing.” Unlike deduction and like induction abduction is a not truth-preserving reasonig method.

More exactly, given my "theory" and some observations (observed consequences) what facts can explain them?

Example 8 ***(Abduction)***

- ***Rule:*** *All the balls in this box are black.*
- ***Result:*** *These balls are black.*
- ***Derive case:*** *These balls are from this box.*

Example 9 ***(Abduction)*** *Let $\mathcal{K}$ be a theory:*

$rain \rightarrow wet\_road$

$rain \rightarrow wet\_grass$

$sun \leftrightarrow \neg\ rain$

$irrigation \rightarrow wet\_grass$

$sun \land hot\_day \rightarrow irrigation$

Observing $O = wet\_grass$, what can we conclude? Nothing really. But we can say that $E = rain$ is a possible explanation of O.

Definition 14 ***(Abduction Problems)*** *Given two sets of formulae $\mathcal{O}$ (possible effects, observations) and $\mathcal{H}$ (possible explanations, hypotheses), a knowledge base $\mathcal{K}$, an abductive explanation of an observation $O \subseteq \mathcal{O}$ w.r.t. $\mathcal{K}$ is any finite set $\mathcal{E} \subseteq \mathcal{H}$. such that $\mathcal{K} \cup \mathcal{E} \models \mathcal{O}$.*

In abduction problems convention is that $\mathcal{H}$ is usually a set of facts – we are interested in diagnosis: what facts could possibly cause the observation?

$\mathcal{E}$ is consistent if $\mathcal{E} \cup \mathcal{K} \not\models \perp$ i.e. $\mathcal{E}$ is consistent w.r.t. $\mathcal{K}$;

$\mathcal{E}$ is relevant if $\mathcal{E} \not\models$ O i.e. $\mathcal{E}$ does not entail O;

$\mathcal{E}$ is explanatory if $\mathcal{K} \not\models$ O i.e. $\mathcal{K}$ does not entail O;

In abduction not every explanation is equally preferred us other ones. Explanation $\mathcal{E}_2$ can be deduced from $\mathcal{E}_3$ using $\mathcal{K}$. We say that $\mathcal{E}_3$ is stronger then $\mathcal{E}_2$.

Definition 15 ***(Strength of explanation)*** *Given $\mathcal{K}$, $O \subseteq \mathcal{O}$, and $\mathcal{E}, \mathcal{E}' \subseteq \mathcal{H}$ s.t. $\mathcal{E} \neq \mathcal{E}'$. We say that the explanation $\mathcal{E}'$ is*

- *stronger than $\mathcal{E}$ w.r.t. $\mathcal{K}$ if $\mathcal{L} \cup \mathcal{E}' \models \mathcal{E}$ and $\mathcal{L} \cup \mathcal{E} \not\models \mathcal{E}'$*
- *independent w.r.t. $\mathcal{K}$ if no stronger explanation of O w.r.t. $\mathcal{K}$ exists.*

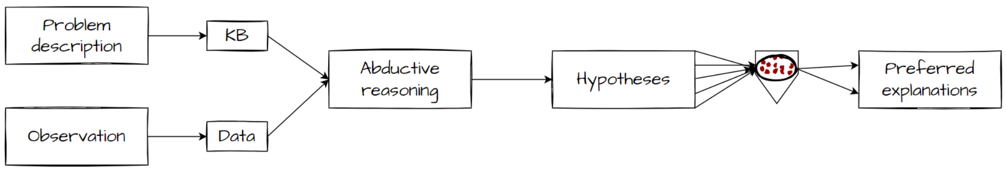

Figure 3.1: Explanation computation in abduction

Explanations $\mathcal{E}_1$ and $\mathcal{E}_4$ are independent, still $\mathcal{E}_4$ can be seen as less preferred because it has a proper subset ($\mathcal{E}_3$ and $\mathcal{E}_2$) which is an explanation too.

Definition 16 ***(Minimality of explanations)*** *Given $\mathcal{K}$, $O \subseteq \mathcal{O}$, we say that an explanation $\mathcal{E} \subseteq \mathcal{H}$ of O w.r.t. $\mathcal{K}$ is minimal if there is no $\mathcal{E}' \subsetneq \mathcal{E}$ that is also an explanation of $\mathcal{O}$ w.r.t. and $\mathcal{K}$.*

On the figure 3.1 we can see the order od the process how explanations are computed.

Computing explanations

Example 10 ***(DL Abduction)*** *Tu bude priklad.*

Finding explanations

- $\mathcal{E}$ is an explanation of $\mathcal{P}$ =($\mathcal{K}$, O) if $\mathcal{K} \cup \mathcal{E} \models$ O
- entailment is reducible to consistency checking
- $\mathcal{E}$ is an explanation of $\mathcal{P}$ if $\mathcal{K} \cup \mathcal{E} \cup \neg$O is inconsistent
- one can find such explanations by considering all models of $\mathcal{K} \cup \neg$O and constructing the set
- $\mathcal{E}$ by selecting and negating one assertion from each of these models
- $\mathcal{K} \cup \mathcal{E} \cup \neg$O will have no models and hence $\mathcal{K} \cup \mathcal{E} \cup \models$ O
- to construct a set containing one negated assertion of each model we use minimal hitting set algorithm

Minimal Hitting Set

- hitting set (HS) for a collection of sets $\mathcal{M}$ – any set $\mathcal{E}$ s.t. $\mathcal{E} \cap$ M $\not\models$ for every M of $\mathcal{M}$

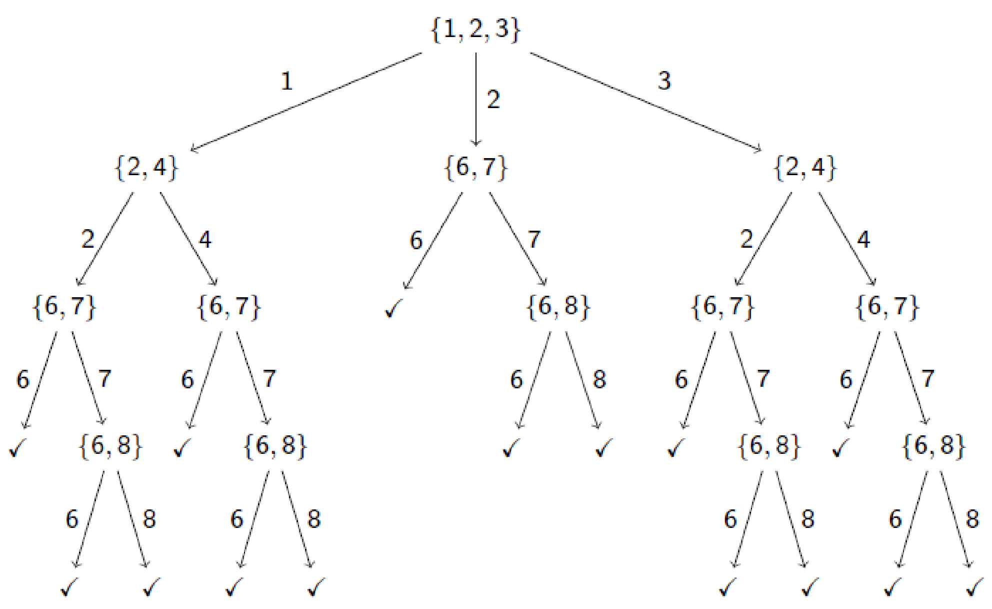

Figure 3.2: Hitting Set Tree example

- minimal HS for $\mathcal{M}$ – HS $\mathcal{E}$ s.t. there is no other HS $\mathcal{E}' : \mathcal{E}' \subsetneq \mathcal{E}$

Example 11 ***(Minimal Hitting Set example)***

$$\mathcal{M} = \{\{1,2,3\}, \{2,4\}, \{6,7\}, \{6,8\}\}$$

$$\mathcal{E}_1 = \{2,6\} \qquad \mathcal{E}_4 = \{1, 4, 7, 8\}$$
$$\mathcal{E}_2 = \{2,7,8\} \qquad \mathcal{E}_5 = \{3,4,6\}$$
$$\mathcal{E}_3 = \{1,4,6\} \qquad \mathcal{E}_6 = \{3,4,7,8\}$$

Definition 17 ***(Hitting Set Tree (Reiter))*** *HS-tree Given $\mathcal{T}$ respective to Given $\mathcal{M}$ is the smallest labelled tree where $H(n)$ is the set of edge-labels on the path from root to node n and $L(n)$ is the label of n s.t.:*

- *for each node n: $L(n) = M \in \mathcal{M}$ s.t. $M \cap H(n) =$ or $L(n) = \checkmark$ if no such set exists*
- *each node n s.t. $L(n) = M \in \mathcal{M}$ has a successor n_σ for each $\sigma \in M$ joined to n by an edge labelled by σ*
- *a node n s.t. $L(n) = \checkmark$ has no successors in $\mathcal{T}$*

On the figure 3.2 we can see a given minimal hitting set $\mathcal{M}$ from example 11.

Minimal Hitting Set Algorithm (Reiter)

To optimize HS-tree and filter not-minimal hitting sets:

- breadth-first search
- pruning

A branch can be pruned in n ($L(n) = ✗$) if:

- either there is n' s.t. $H(n') \subseteq H(n)$ and $L(n') = \checkmark$
- or there is n' s.t. $H(n') = H(n)$ and $L(n') = M \in \mathcal{M}$.

Theorem 1 ***(Minimal Hitting Set Algorithm (Reiter))*** *Let $\mathcal{M}$ be a collection of sets, and T a pruned HS-tree for $\mathcal{M}$, as previously described. Then $\{H(n) \mid n$ is a node of T labelled by $\checkmark\}$ is the collection of minimal hitting sets for $\mathcal{M}$.*

From theorem 1 we can say that algorithm is composed from two steps. At first we create a pruned HS-tree for $\mathcal{M}$ and as the second step we choose $\{ H(n) \mid n$ is a node of T labelled by $\checkmark\}$ which is a collection od minimal hitting sets.

Chapter 4

OWL API

To talk about OWL API at first we will introduce OWL language.

4.1 OWL language

Ontologies are represented in the logical language OWL - Web Ontology Language, which not only stores information about classes, their properties or their relationships, but also has the ability to verify the consistency of data. Enriches the original RDF language. It is possible, for example, to determine the equality of classes or the difference of individual instances. This helps the user to choose the right concepts, even if there are several sources describing these sources differently. They also provides removal of ambiguity, in other words, clarification if there are different instances that have a common name or description.[4] [11]

The following example shows how to use the OWL language easily. We will use Manchester syntax for the example. Let us have a `Smartphone` entity with the `hasModel` property. First we create a class `Smartphone`. Then we create the data property `hasModel`, which domain, which is what we will assign it to, is our entity `Device` and its value range will be the data type `string`. We can also add an annotation which is a brief description. Let's write it like this:

```
Class: Smartphone
DataProperty: hasModel
  Annotations:
    rdfs:comment "Smartphone has a model."
  Domain:
    Smartphone
  Range:
    xsd:string
```

4.2 OWL API

The OWL high level Application Programming Interface (API) is impelmented in language JAVA and it is avaible as open source. It is created to facilitate manipulation, creation and serialising OWL Ontologies. Also reasoning over ontologies and the working with them in applications is provided. At the time of writing this article, the OWL API supports OWL 2. The OWL API was brought to the public at 2003 from when it evolved in the most important and used API for OWL. Its implementation is very flexible as it provides alternative implementations for all major components. Some of big project where it is used are: Protégé-4 [8], OntoTrack [9], the Pellet reasoner [16] and others. [6]

Chapter 5

Abductive solvers

Abductive reasoner is an solver which takes a problem and inferences to the best explanation (also explanations) for it using abductive reasoning.

Our DL Abduction API results from generalization over a few software solutions (solvers). We will took a look at those in detail in the next chapter.

Part II

Contribution

Chapter 6

Abductive solvers analysis

This chapter is a short analysis of solvers.
Research.

Main characteristics of every abductive solver The API will try to be general enough to encapsulate different DL abduction solvers. To our current knowledge, the inputs that need to be considered are summarized as follows:

1. Input - formats: OWL file, org.semanticweb.owlapi.model.OWLOntology

2. Output - TXT file, sets of axioms (org.semanticweb.owlapi.model.OWLOntology)

3. Observation/s - single, multiple observations (org.semanticweb.owlapi.model.OWLOntology) exceptions

4. Abducibles - there will be 3 ways how to work with them
   - 1. switches - enabling/disabling: loops, role assertions, concept assertions, complex concepts and concept complement
   - 2. abducible enumeration - for example: 4 concepts and 1 individual
   - 3. assertion enumeration - for example: jack is not a parent
   - Option number 3 is exclusive and cannot be combined with any of the others. Meanwhile option 1 and option 2 can be combined but used individually too.

5. Solver's internal settings as string

- Next: solver comparison with table (in progress).

Solvers								
Name	Input	Observation	Output	Loops	Depth	Abducibles	Timeout	Prefixes
AAA	OWL file	string	TXT file	yes	number	string	x	x
MHS-MXP	OWL file	string - <class>(<ind>)	TXT file	x	number	string	seconds	string
ABox Abduction via Forgetting in ALC	x	x	x	x	x	x	x	x

ABox Abduction via Forgetting in ALC - The abduction algorithm takes as input an ALC ontology O, an observation OB as a set of ABox axioms and a forgetting signature F.

MHS-MX - Observation string in the form <class>(<ind>) where <class> and <ind> are IRIs of the class and the individual forming the concept assertion

Chapter 7

Abductive solvers - coverage

Our approach is to create an API which will be abstractly generalized over abductive solvers. It will cover needs from previous analysis.

7.1 Communication

At first we will discuss a communication between solver (abductive inference machine) and application (or any tool where solver will be integrated). There will be no direct communication between them. Communication will be provided by DL Abduction API which will use Threads in order to be able to return explanations incrementally in real-time, when they are computed. Threads are a very good option because we can provide every new explanation computed by solver in real time to the application thanks to them.

The communication is represented graphically at the figure 7.1.

1. First, all settings need to be prepared.

2. Next the application initializes a new thread which will represent an instance of API.AbductionManager – where new explanations are computed (button START).

3. At that point a monitor is set to wait for a new notification from API.

4. In the API.run method new explanations are computed. After every new computed explanation the method API.show is called. From that method a new explanation is added to the monitor and a notification is sent to the monitor.

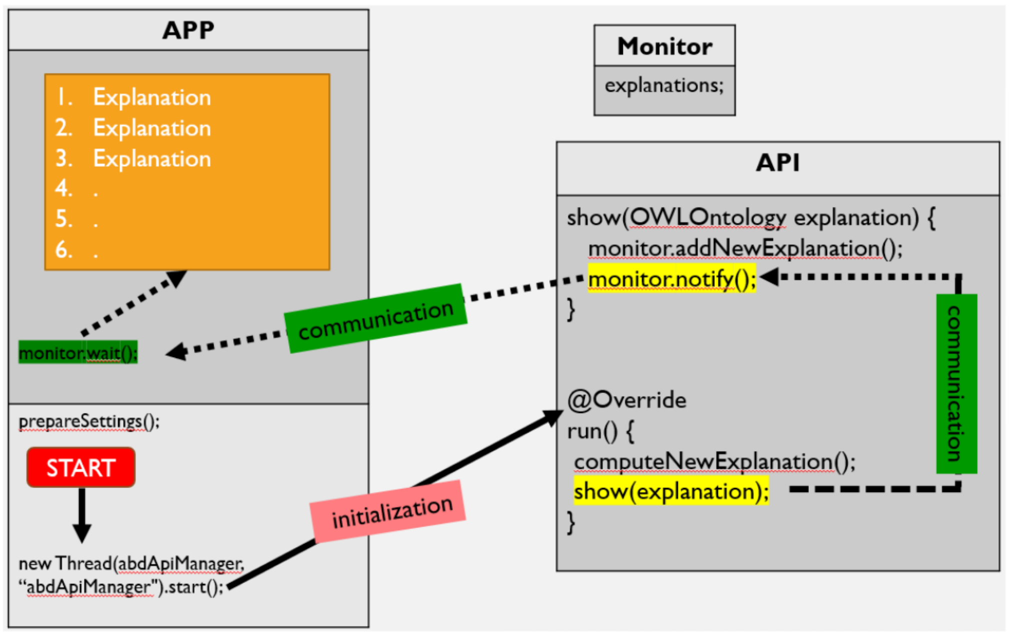

Figure 7.1: Communication between solver and application.

5. At a moment a notification is passed to the application, the application can readily show the new explanation.

Chapter 8

API proposal

In the next chapter we will propose the API draft we created based on analysis and coverage from past chapters. We will show UML diagrams to model the API well.

On the figure 8.1 we can see UML Class Diagram of DL Abduction API.

- TODO: flow/state diagram

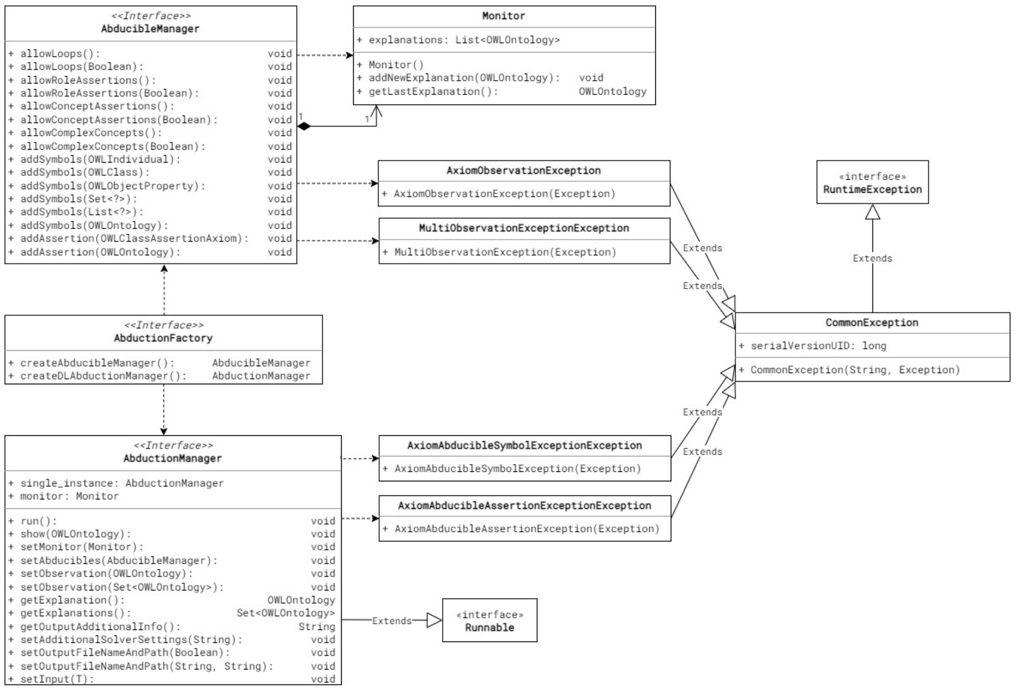
<<Interface>>
AbducibleManager
+ allowLoops(): void
+ allowLoops(Boolean): void
+ allowRoleAssertions(): void
+ allowRoleAssertions(Boolean): void
+ allowConceptAssertions(): void
+ allowConceptAssertions(Boolean): void
+ allowComplexConcepts(): void
+ allowComplexConcepts(Boolean): void
+ addSymbols(OWLIndividual): void
+ addSymbols(OWLClass): void
+ addSymbols(OWLObjectProperty): void
+ addSymbols(Set<?>): void
+ addSymbols(List<?>): void
+ addSymbols(OWLOntology): void
+ addAssertion(OWLClassAssertionAxiom): void
+ addAssertion(OWLOntology): void
Monitor
+ explanations: List<OWLOntology>
+ Monitor()
+ addNewExplanation(OWLOntology): void
+ getLastExplanation(): OWLOntology
1
1
AxiomObservationException
+ AxiomObservationException(Exception)
MultiObservationExceptionException
+ MultiObservationException(Exception)
«interface»
RuntimeException
Extends
Extends
Extends
CommonException
+ serialVersionUID: long
+ CommonException(String, Exception)
<<Interface>>
AbductionFactory
+ createAbducibleManager(): AbducibleManager
+ createDLAbductionManager(): AbductionManager
Extends
Extends
<<Interface>>
AbductionManager
+ single_instance: AbductionManager
+ monitor: Monitor
+ run(): void
+ show(OWLOntology): void
+ setMonitor(Monitor): void
+ setAbducibles(AbducibleManager): void
+ setObservation(OWLOntology): void
+ setObservation(Set<OWLOntology>): void
+ getExplanation(): OWLOntology
+ getExplanations(): Set<OWLOntology>
+ getOutputAdditionalInfo(): String
+ setAdditionalSolverSettings(String): void
+ setOutputFileNameAndPath(Boolean): void
+ setOutputFileNameAndPath(String, String): void
+ setInput(T): void
AxiomAbducibleSymbolExceptionException
+ AxiomAbducibleSymbolException(Exception)
AxiomAbducibleAssertionExceptionException
+ AxiomAbducibleAssertionException(Exception)
Extends
«interface»
Runnable

Figure 8.1: DL Abduction API class diagram

Chapter 9

Implementation

Chapter 10

Implementation at solver

A manual how to use DL Abduction API.

Chapter 11

Conclusions

Results and conclusions together.
Results interpretation.

Bibliography

[1] Franz Baader. Description logic terminology. *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 485–495, 01 2003.

[2] Thomas Gruber. A translation approach to portable ontology specifications. 5, 11 1992.

[3] Nicola Guarino, Daniel Oberle, and Steffen Staab. *What Is an Ontology?*, pages 1–17. 05 2009.

[4] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter Patel-Schneider, and Sebastian Rudolph. Owl 2 web ontology language primer (second edition), 01 2012. `https://www.w3.org/TR/owl2-primer/#Introduction`[10.2.2020].

[5] Matthew Horridge. A practical guide to building owl ontologies using protégé 4 and co-ode tools edition 1.3. 03 2011.

[6] Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semantic Web*, 2:11–21, 01 2011.

[7] Mijung Kim, Jake Cobb, Mary Harrold, Tahsin Kurc, Alessandro Orso, Joel Saltz, Andrew Post, S. Malhotra, and Shamkant Navathe. Efficient regression testing of ontology-driven systems. *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, pages 320–330, 07 2012.

[8] Holger Knublauch, Ray Fergerson, Natasha Noy, and Mark Musen. The protégé owl plugin: An open development environment for semantic web applications. *Third International Semantic Web Conference*, 3298:229–243, 11 2004.

[9] Thorsten Liebig and Olaf Noppens. Ontotrack: Combining browsing and editing with reasoning and explaining for owl lite ontologies. 3298:244–258, 11 2004.

[10] N. Noy and Deborah Mcguinness. Ontology development 101: A guide to creating your first ontology. *Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880*, 32, 01 2001.

[11] Ontotext. *What are Ontologies?* `https://www.ontotext.com/knowledgehub/fundamentals/what-are-ontologies/`[10.2.2020].

[12] Charles S. Peirce. Deduction, Induction, and Hypothesis. *Popular Science Monthly*, 13:470–482, 1878.

[13] Júlia Pukancová and Martin Homola. The aaa abox abduction solver. *KI - Künstliche Intelligenz*, 34(4):517–522, 2020.

[14] Sebastian Rudolph. Foundations of description logics. volume 6848, pages 76–136, 01 2011.

[15] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[16] Evren Sirin, Bijan Parsia, Bernardo Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5:51–53, 06 2007.