Explaining Reasoning Results for OWL Ontologies with EVEE

Christian Alrabbaa[1] , **Stefan Borgwardt**[1] , **Tom Friese**[1] , **Anke Hirsch**[2] , **Nina Knieriemen**[2] ,
Patrick Koopmann[3] , **Alisa Kovtunova**[1] , **Antonio Krüger**[2] , **Alexej Popovič** , **Ida Siahaan**[1]
[1]Technische Universität Dresden, Faculty of Computer Science, Dresden, Germany
[2]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[3]Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands
{christian.alrabbaa, stefan.borgwardt, tom.friese, alisa.kovtunova,
ida_sri_rejeki.siahaan}@tu-dresden.de, {anke.hirsch, nina.knieriemen, antonio.krueger}@dfki.de,
p.k.koopmann@vu.nl, alexej.popovic@protonmail.com

Abstract

One of the advantages of formalizing domain knowledge in OWL ontologies is that one can use reasoning systems to infer implicit information automatically. However, it is not always straightforward to understand why certain entailments are inferred, and others are not. The popular ontology editor PROTÉGÉ offers two explanation services to deal with this issue: justifications for OWL 2 DL ontologies, and proofs generated by the reasoner ELK for lightweight OWL 2 EL ontologies. Since justifications are often insufficient for explaining inferences, there is thus only little tool support for more comprehensive explanations in expressive ontology languages, and there is no tool support at all to explain why something was not derived. In this paper, we present EVEE, a Java library and a collection of plug-ins for PROTÉGÉ that offers advanced explanation services for both inferred and missing entailments. EVEE explains inferred entailments using proofs in description logics up to $\mathcal{ALCH}$. Missing entailments can be explained using counterexamples and abduction. We evaluated the effectiveness and the interface design of our plug-ins with description logic experts, ontology engineers, and students in two user studies. In these experiments, we were able to not only validate the tool but also gather feedback and insights to improve the existing designs.

1 Introduction

Description logics (DLs) (Baader et al. 2017a) have gained popularity through the standardization of the Web Ontology Language OWL[1], the development of an OWL Java API (Horridge and Bechhofer 2011), editing tools such as PROTÉGÉ (Musen 2015), and various reasoning systems. A central application of ontologies is to use reasoning systems to derive implicit subsumption relationships from these ontologies. However, due to the complexity of modern ontologies and the expressivity of description logics, it is not always straight-forward to understand why certain relationships can be derived and others cannot. To help with this, we developed EVEE (EVincing Expressive Entailments), a collection of tools that in its current version 0.3 offers services to explain both logical entailments and missing entailments from OWL ontologies.

Research on **explaining logical entailments** in DLs first considered *proofs* that provide detailed inference steps through which a consequence can be obtained (McGuinness 1996; Borgida, Franconi, and Horrocks 2000). Later literature focussed on *justifications*, minimal sets of ontology axioms that are sufficient for the entailment, as explanations (Kalyanpur et al. 2007; Schlobach and Cornet 2003; Baader, Peñaloza, and Suntisrivaraporn 2007; Horridge 2011). However, if justifications become very large or the ontology is formulated in an expressive DL, providing intermediate inference steps between a justification and its consequence may be required for understanding (Horridge, Parsia, and Sattler 2009; Horridge, Parsia, and Sattler 2010). One way to produce proofs is by directly using the inference rules underlying a consequence-based reasoner like ELK (Kazakov, Krötzsch, and Simancik 2014), which however only supports $\mathcal{EL}_{\bot}^{+}$. There is already a plugin for the ontology editor PROTÉGÉ that allows to access proofs generated using ELK (Kazakov, Klinov, and Stupnikov 2017), but nothing for other proof generation methods, and therefore also nothing for DLs beyond $\mathcal{EL}_{\bot}^{+}$.

Proofs based on inference rules sometimes appear tedious and can only be generated if a set of inference rules exists and is implemented. In the literature, we can also find entirely different approaches: proofs generated through heuristic search for possible intermediate inferences (Horridge, Parsia, and Sattler 2010), concept interpolation (Schlobach 2004) and forgetting (Alrabbaa et al. 2020a), which is a technique for reducing the vocabulary of an ontology. Independently of the proof generation method, proofs can be optimized w.r.t. various measures, e.g. proof size or depth (Alrabbaa et al. 2020a; Alrabbaa et al. 2020b; Alrabbaa et al. 2021). For example, from a set of instantiated inference rules, e.g. computed by ELK, one can extract a proof of minimal depth in polynomial time (Alrabbaa et al. 2021).

There is no conclusive answer on which proof generation method and which measure is the best: One finding from a series of quantitative user studies on preferred proof presentations was that—despite similar performance across various proof representations—participants show distinct subjective preferences (Alrabbaa et al. 2022b). Consequently, EVEE can generate different types of proofs to explain logical entailments: detailed ones based on ELK and the reasoning calculus of the forgetting tool LETHE (Koopmann 2020) (the latter one also supporting the more expressive DL $\mathcal{ALCH}$),

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

Explanation	DL	Assert.
ELK-based proofs [3.1]	$\mathcal{EL}_\bot^+$	$+$
Elimination proofs [3.2]	$\mathcal{ALCH}$	$+$
LETHE-based proofs [3.3]	$\mathcal{ALCH}$	$+$
Small counterexamples [4.1]	$\mathcal{EL}_\bot$	$-$
Canonical counterexamples [4.1]	$\mathcal{EL}$	$-$
Connection-minimal abduction [4.2]	$\mathcal{EL}$	$-$
Complete signature-based abduction [4.2]	$\mathcal{ALC}$	$+$

Table 1: Summary of all entailment (at the top) and missing entailment (at the bottom) explanation services offered by EVEE, their corresponding supported description logics, and whether they also support explaining class and object property assertions in addition to subclass relationships.

as well as more abstract proofs that are based on forgetting. The latter so-called *elimination proofs* can support different DLs, depending on the forgetting tool that is used internally in a black-box fashion. Our library currently uses FAME (Zhao and Schmidt 2018) and LETHE, which leads to proof methods supporting DLs up to the expressivity of $\mathcal{ALCH}$. Different to the existing PROTÉGÉ proof plug-in (Kazakov, Klinov, and Stupnikov 2017), we can optimize ELK-based proofs w.r.t. to different proof measures.

While reasoning can sometimes reveal unexpected entailments that need explaining, very often the problem is not what is entailed, but what is *not* entailed. In order to **explain missing entailments**, and offer suggestions on how to repair them, there are basically two approaches: counterexamples and abduction. A *counterexample* is a model of the ontology that does not satisfy the entailment. We provide different methods for counterexamples: a tableau-based method that optimizes the size of the model, and methods that also focus on the part of the model that is relevant for the missing entailment, and may contrast a counterexample with a positive example (Alrabbaa and Hieke 2022). In *abduction*, the missing entailment is explained by means of *hypotheses*, which are sets of axioms that can be added to the ontology in order to entail the missing consequence (Elsenbroich, Kutz, and Sattler 2006). There is a vast literature on abduction in DLs. We mainly focussed on recent methods that target the explanation of missing entailments, for which we rely on external abduction tools: CAPI,[2] which computes hypotheses that satisfy a criterion called *connection-minimality* (Haifani et al. 2022), and which itself relies on the first-order theorem prover SPASS;[3] and LETHE-abduction, which implements *complete signature-based abduction* (Koopmann et al. 2020). Users can explore the space of hypotheses by restricting the set of entities that can be used, and use them in combination with the proof-based explanation methods to explain how a hypothesis would make the missing entailment possible. Table 1 summarizes all the explanation types offered in EVEE.

[2]https://lat.inf.tu-dresden.de/~koopmann/CAPI

[3]https://www.mpi-inf.mpg.de/departments/automation-of-logic/software/spass-workbench/classic-spass-theorem-prover

Our tools target PROTÉGÉ, which is the most used freely available ontology editor, to improve the ontology engineering experience with more advanced explanation techniques. EVEE features a Java library EVEE-LIBS, which forms the basis for a collection of PROTÉGÉ plug-ins under the umbrella name EVEE-PROTEGE. In the following, we use the name EVEE to collectively refer to both components. However, the focus of this paper is on the PROTÉGÉ plug-ins.

Since our tools were for the first time integrated into Protégé, we evaluated the effectiveness and the interface design of EVEE in two *qualitative* user studies. The aim of the studies was to get a first impression on whether these types of explanation services are helpful, and what improvements might be needed. In contrast to the quantitative research in (Alrabbaa et al. 2022b), qualitative studies with Protégé users were sufficient for this exploration. The received feedback was mostly positive: the study participants confirmed that EVEE meets the intended requirements. Moreover, we collected opinions and suggestions for further improvements.

The source code and installation instructions for EVEE are available at https://github.com/de-tu-dresden-inf-lat/evee. The resources used for the user studies are available on Zenodo (Alrabbaa et al. 2024).

2 Preliminaries

We recall the basics of OWL and DLs, based on the *Manchester syntax* used in PROTÉGÉ and our plug-ins; for more details, see (Baader et al. 2017a; Hitzler et al. 2012). OWL ontologies are built from *classes* such as **Pizza**, representing the class of all pizzas, *object properties*, such as **hasTopping**, expressing relations between pizzas and their toppings, and *individuals*, such as **Italy**, representing the (unique) country *Italy*. Classes, object properties and individuals are collectively called *entities*. A *signature* (or *vocabulary*) is a finite set of entities. Different DLs offer different constructors by which more complex expressions can be built from entities. For example, the lightweight description logic $\mathcal{EL}$ supports the class constructors **and** and **some**. Using these, one can build *class expressions* like **Pizza and hasTopping some Broccoli**, describing all pizzas that have broccoli as topping. In addition, the special class **owl:Thing** refers to everything in the domain.

Axioms represent knowledge about classes, for example the *equivalence axiom*

MeatyPizza EquivalentTo Pizza and hasTopping some Meat

defines the class of meaty pizzas, and the *subclass axiom*

AmericanPizza SubClassOf Pizza

declares American pizzas to be a more specific subclass of pizzas. In addition, the *class assertion* **Italy Type Country** states that Italy belongs to the class of countries, whereas the *object property assertion* **Italy hasNeighbor France** expresses that Italy and France share a land border. Each such axiom corresponds to a sentence in first-order logic (Baader et al. 2017a), so that we can speak of logical *entailment* between sets of axioms. In the context of our paper, an *ontology* $\mathcal{O}$

is just a set of such axioms, and we are interested in *logical entailments* $\mathcal{O} \models \alpha$ of axioms α from ontologies $\mathcal{O}$. For example, adding the axiom

AmericanPizza SubClassOf hasTopping some Meat

(American pizzas contain meat) to the previous ones yields an ontology that entails the axiom

AmericanPizza SubClassOf MeatyPizza

(American pizzas are meaty pizzas).

There are other DLs relevant for this paper. The DL $\mathcal{EL}_\bot$ is obtained by adding the empty class **owl:Nothing** to $\mathcal{EL}$, which allows formulating *disjointness axioms* like

Pizza and Country SubClassOf owl:Nothing

(abbreviated as **Pizza DisjointWith Country**). The more complex DL $\mathcal{ALC}$ extends $\mathcal{EL}_\bot$ by the constructors **or**, **not**, and **only**, using which we can express

Pizza and hasTopping only (not Meat)

(pizzas with only non-meat toppings) and

Pizza and hasTopping only (Mozzarella or Tomato)

(pizzas where all toppings are either mozzarella or tomato sauce). $\mathcal{ALCH}$ denotes the extension of $\mathcal{ALC}$ by *subproperty axioms* like

hasTopping SubPropertyOf hasIngredient

(every topping of a pizza is also an ingredient of it). Finally, $\mathcal{EL}_\bot^+$ extends $\mathcal{EL}_\bot$ by more complex object property inclusions (including transitivity), and $\mathcal{EL}^{++}$ extends $\mathcal{EL}_\bot^+$ by nominals and so-called *concrete domains*, and is the basis for the efficient OWL 2 EL profile.

3 Explaining Entailments

Without additional plugins, PROTÉGÉ only includes a single explanation service for entailments: consequences discovered by the reasoner have a "?"-labeled button next to them that opens an explanation window. The built-in service provides explanations in the form of *justifications*, which are subset-minimal sets of axioms that are sufficient to produce the entailment (Schlobach and Cornet 2003), and, intuitively, are "responsible" for the entailment. Kazakov, Klinov, and Stupnikov (2017) developed an alternative explanation service that can show proofs generated by the reasoner ELK. With the EVEE plug-ins installed, the user can choose between different additional proof services. Each of them provides a detailed explanation in the form of a *proof tree* (see Figure 1), which is a tree where nodes are labeled with axioms, the root is the entailment to be explained, and the leafs are either axioms from the ontology or tautologies. Each inner node is labeled with an axiom that is a logical consequence of the labels of its child nodes, and is thus the result of a logical inference step. The way these inferences work depends on the chosen *proof method* offered by EVEE:

[4]https://protege.stanford.edu/ontologies/pizza/pizza.owl

1. *optimized* ELK*-based proofs*,
2. *elimination proofs*, and
3. LETHE-based *detailed proofs*.

These methods differ in the amount of detail in the resulting proofs, as well as in the DLs they support. Figure 1 shows the proofs generated by those three methods.

3.1 ELK-Based Proofs

ELK-based proofs have a high level of detail, can be generated very quickly, and support all inferences that can be performed by the popular ELK reasoner (Kazakov, Krötzsch, and Simancik 2014). This means that ELK-based proofs support most of the OWL EL profile. Internally, ELK uses a set of *inference rules* that can also be used to generate proofs. The Java library of ELK makes it possible to access all inferences steps that can be used for deriving a specific entailment (Kazakov and Klinov 2014). This functionality is also used in ELK's own PROTÉGÉ proof service (Kazakov, Klinov, and Stupnikov 2017). The main difference to our implementation is that we only show one proof at a time, while the other plug-in shows all possible inferences that can lead to the entailment within a single tree structure. We can extract a single proof according to one of the following optimization criteria: minimal *size* (number of axioms), *depth* of the proof tree, or *weighted size* (total length of all axioms); for details, see (Alrabbaa et al. 2021).

3.2 Elimination Proofs

Elimination proofs were originally introduced in (Alrabbaa et al. 2020a) under the name *forgetting-based proofs*, with the motivation of supporting DLs for which no implemented proof method exists yet. Elimination proofs are computationally more expensive, support the more expressive DL $\mathcal{ALCH}$, and offer the lowest level of detail, thus giving a more high-level explanation.

As the name suggests, every inference in an elimination proof eliminates one or several entities from previously derived axioms. In particular, inference steps have the form

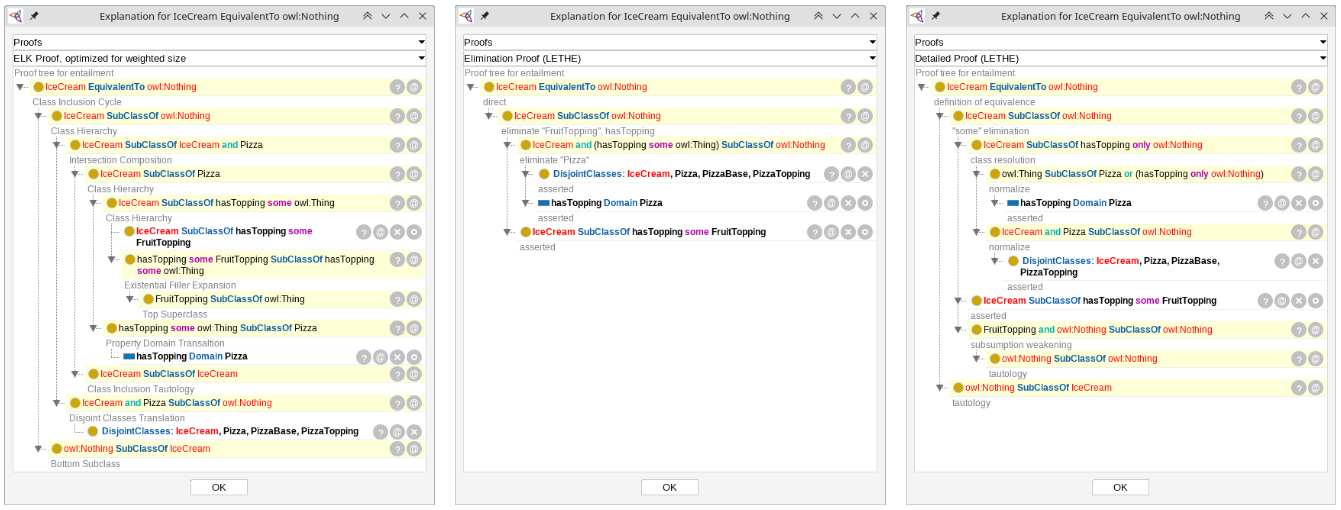
$$\frac{\alpha_1 \qquad \dots \qquad \alpha_n}{\beta} \text{ eliminate } X,$$

where $\{\alpha_1, \dots, \alpha_n\} \models \beta$, and the entities in X occur in the premises, but not in the conclusion. At least one entity is eliminated in each step, but several entities can be eliminated at the same time if this does not make the proof more complex, e.g. by increasing the number of premises of the inference step.

To compute elimination proofs, we make use of the *forgetting tools* LETHE (Koopmann 2020) and FAME (Zhao and Schmidt 2018). Which tool is used affects the shape of the axioms that are used in the proofs. EVEE offers four different variants of elimination proofs: a fast *heuristic* method, and slower *optimized* proofs (of minimal *size*, *weighted size*, or *number of elimination steps*).

3.3 LETHE-based Detailed Proofs

These proofs offer a level of detail comparable to that of ELK-based proofs, while supporting axioms in the more ex-



```
Explanation for IceCream EquivalentTo owl:Nothing
Proofs
ELK Proof, optimized for weighted size
Proof tree for entailment
IceCream EquivalentTo owl:Nothing
  Class Inclusion Cycle
    IceCream SubClassOf owl:Nothing
      Class Hierarchy
        IceCream SubClassOf IceCream and Pizza
          Intersection Composition
            IceCream SubClassOf Pizza
              Class Hierarchy
                IceCream SubClassOf hasTopping some owl:Thing
                  Class Hierarchy
                    IceCream SubClassOf hasTopping some
                    FruitTopping
                    hasTopping some FruitTopping SubClassOf hasTopping
                    some owl:Thing
                      Existential Filler Expansion
                        FruitTopping SubClassOf owl:Thing
                          Top Superclass
                hasTopping some owl:Thing SubClassOf Pizza
                  Property Domain Transaltion
                    hasTopping Domain Pizza
            IceCream SubClassOf IceCream
              Class Inclusion Tautology
        IceCream and Pizza SubClassOf owl:Nothing
          Disjoint Classes Translation
            DisjointClasses: IceCream, Pizza, PizzaBase, PizzaTopping
    owl:Nothing SubClassOf IceCream
      Bottom Subclass
OK
```

```
Explanation for IceCream EquivalentTo owl:Nothing
Proofs
Elimination Proof (LETHE)
Proof tree for entailment
IceCream EquivalentTo owl:Nothing
  direct
    IceCream SubClassOf owl:Nothing
      eliminate "FruitTopping", hasTopping
        IceCream and (hasTopping some owl:Thing) SubClassOf owl:Nothing
          eliminate "Pizza"
            DisjointClasses: IceCream, Pizza, PizzaBase, PizzaTopping
              asserted
            hasTopping Domain Pizza
              asserted
        IceCream SubClassOf hasTopping some FruitTopping
          asserted
OK
```

```
Explanation for IceCream EquivalentTo owl:Nothing
Proofs
Detailed Proof (LETHE)
Proof tree for entailment
IceCream EquivalentTo owl:Nothing
  definition of equivalence
    IceCream SubClassOf owl:Nothing
      "some" elimination
        IceCream SubClassOf hasTopping only owl:Nothing
          class resolution
            owl:Thing SubClassOf Pizza or (hasTopping only owl:Nothing)
              normalize
                hasTopping Domain Pizza
                  asserted
            IceCream and Pizza SubClassOf owl:Nothing
              normalize
                DisjointClasses: IceCream, Pizza, PizzaBase,
                PizzaTopping
                  asserted
        IceCream SubClassOf hasTopping some FruitTopping
          asserted
        FruitTopping and owl:Nothing SubClassOf owl:Nothing
          subsumption weakening
            owl:Nothing SubClassOf owl:Nothing
              tautology
    owl:Nothing SubClassOf IceCream
      tautology
OK
```

Figure 1: Completely unfolded ELK-based proof, LETHE-based elimination proof and LETHE-based detailed proof, generated for the pizza ontology[4] and the entailment **IceCream EquivalentTo owl:Nothing**.

pressive DL $\mathcal{ALCH}$. They can be used in case an elimination proof offers too little detail to understand the entailment. The forgetting tool LETHE used for elimination proofs internally uses a set of inference rules. We can thus trace the inference steps used by LETHE, similarly as in ELK. LETHE uses a simplified calculus with only few inference rules (Koopmann and Schmidt 2013), which operate on a normalized version of the ontology. Before we can use the traced inference steps in a proof, we have to denormalize them, which may introduce further inference steps. Details on this can be found in (Alrabbaa et al. 2022a).

3.4 Using Signatures of Known Entities

The user also has the possibility to select a *signature of known entities*, consisting of classes, object properties and individuals. The idea is that these describe a vocabulary with which the user is familiar, so that entailments that are in this signature do not need further explanation. If such a signature is selected, axioms that are expressed using only known entities will not get a sub-proof, that is, they become leafs of the proof tree. This is particularly useful in combination with an optimization criterion for the proofs, since the optimization of proofs takes the known entities into account: a proof is optimal w.r.t. all proofs in which sub-proofs for axioms using only known entities have been removed.

4 Explaining Missing Entailments

EVEE can also provide explanations in case an expected entailment α is missing, i.e. when $\mathcal{O} \not\models \alpha$. To understand why this is the case, or even to repair the ontology to make it entail α, we provide several approaches that can be grouped into two categories, *counterexamples* and *abduction*.

4.1 Counterexamples

In general, a counterexample is a model of $\mathcal{O}$ that does not satisfy the missing entailment α. So far, we only support

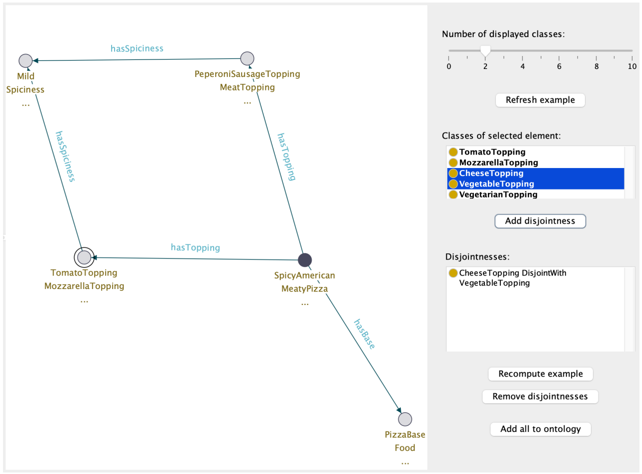


Figure 2: A counterexample generated for **SpicyAmerican SubClassOf SpicyPizza** in a modified version of the Pizza ontology.

the case where α is a subclass axiom **C SubClassOf D**, which means that the counterexample must contain an element that belongs to **C**, but does not belong to **D**. Such a model can be presented as a labeled directed graph, where the nodes are domain elements labeled by classes and the edges are labeled by object properties (see Figure 2). EVEE provides two kinds of counterexamples: *small counterexamples* and *canonical counterexamples*.

Small counterexamples are computed using a tableaux algorithm that generates models for $\mathcal{EL}_\bot$ ontologies (Alrabbaa et al. 2023). Starting from an assertion a^* **Type C** for a fresh individual a^*, we apply tableau expansion rules to generate a model of the whole ontology that satisfies **C**. For example, if **C EquivalentTo r some B**, then new assertions a^* **r b** and **b Type B** will be added, where **b** is either an existing or a new individual. In this process, individuals are reused whenever possible to keep the model small. Moreover, the algorithm ensures that a^* does not become an instance of D, since the goal is to obtain a counterexample for C SubClassOf D.

Canonical counterexamples instead focus on relevant fragments of models, using the methods described in (Alrabbaa and Hieke 2022). They consider $\mathcal{EL}$ ontologies and use the reasoner ELK (Kazakov, Krötzsch, and Simancik 2014). The main idea is to use the so-called *canonical model* of $\mathcal{O}$ (Baader et al. 2017b), which is already a counterexample, since it does not satisfy any GCIs that are not entailed by the ontology. However, since the whole canonical model of the ontology would be very large, we only include the parts of the model that are relevant to the missing entailment C SubClassOf D. For example, we only include elements that are connected to the element representing C. Using canonical models, we can also compute *contrastive counterexamples*, which include another individual satisfying D that serves as a point of comparison to the individual satisfying C.

Since our counterexamples focus on non-entailed subclass axioms, they ignore all assertions in $\mathcal{O}$. However, it should be possible to extend these methods to support assertions as well.

4.2 Abduction

Counterexamples always reflect only one model at a time, and do not always make it obvious how to fix the missing entailment. For this reason, we also offer abduction as explanation service.

Given a non-entailment $\mathcal{O} \not\models \alpha$, abduction computes a set of *hypotheses* $\mathcal{H}$, which are sets of axioms such that $\mathcal{O} \cup \mathcal{H} \models \alpha$. Without further restrictions, $\{\alpha\}$ itself is already a hypothesis, which is why useful abduction requires additional constraints on the solution space. *Complete signature-based abduction* (Koopmann et al. 2020) supports the DL $\mathcal{ALC}$ and relies on a user-given vocabulary. It is the only service that also supports assertions and combinations of axioms as missing entailment, and computes a set of alternative hypotheses that only use names from the vocabulary, such that any other such hypothesis can be obtained by strengthening or combining these hypotheses. This set can in some cases be infinite. In contrast, *connection-minimal abduction* computes hypotheses satisfying a minimality criterion called *connection-minimality* (Haifani et al. 2022), with the aim of focusing on those hypotheses that have a more direct connection to the observation. Connection-minimal abduction only supports explaining subclass axioms over $\mathcal{EL}$ ontologies and hypotheses are always without object property restrictions.

5 The PROTÉGÉ Plug-ins

While EVEE can be used as a Java library, the easiest way to use our proof generators and missing entailment explanations is via plug-ins for PROTÉGÉ, a popular editor for OWL ontologies (Musen 2015). The user can put all plug-ins, or only a selection based on their preferences, into the *plugins* folder of PROTÉGÉ to install them. Our modular architecture makes it easy to develop new plug-ins for other explanation methods based on proofs, abduction or counterexamples. Details on how to do this can be found in (Alrabbaa et al. 2023; Alrabbaa et al. 2022a).

If an explanation service does not support the full expressivity of the currently active ontology, it first filters out all unsupported axioms before computing the explanation, which means that explanations may be incomplete in such cases. EVEE will show a message whenever axioms are ignored because of this. EVEE also adds several settings in the PROTÉGÉ preferences under the *Explanations* tab, e.g. for selecting different explanation services or switching between different variants of the user interface.

5.1 Proof Services

To explain reasoning results in PROTÉGÉ, one can click on the "?"-button next to an entailment. The standard explanation consists of a list of justifications for the axiom, but this has been extended to support proofs via the *protege-proof-explanation*[5] plug-in, which relies on the *proof utility library*[6] (PULi) (Kazakov, Klinov, and Stupnikov 2017).

Our proof generators are available in several PROTÉGÉ plug-ins that utilize this existing functionality. This means that, to explain an entailment shown in PROTÉGÉ, users can simply click on the "?" next to it. They can then select one of the proof methods, and navigate the computed proof starting from the entailment, as shown in Figure 1.

In some cases, the elimination and LETHE-based detailed proof methods can take some time to complete. This is in particular the case for optimized elimination proofs, which are disabled by default for this reason. If proof generation takes too long, users can abort the process by clicking the "Cancel"-button or by closing the progress window. Some of the proof services will then show the best proof found so far, provided they already found one. If this is the case, a warning message appears to inform the user that the proof may be sub-optimal.

The proof plug-ins also add a menu entry *Proofs* $\rightarrow$ *Manage vocabulary* through which one can specify the vocabulary of known entities, as well as save and load different vocabularies. In the generated proofs, axioms that are only using the known entities do not get sub-proofs, which is taken into account when computing optimal proofs.

5.2 Missing Entailments

If an entailment is missing, users cannot just click on a "?" somewhere—they have to specify what is missing. For this, our plug-ins provide a new tab available via *Window* $\rightarrow$ *Tabs* $\rightarrow$ *Missing Entailment Explanation.*

In this tab, one can choose one of the installed *missing entailment explanation services* and enter the missing entailment using a text field with PROTÉGÉ auto-complete functionality. Only *OWL logical axioms* are allowed, e.g. subclass-, equivalence-, and disjointness axioms and assertions. The missing entailment can also be saved to or loaded

[5] https://github.com/liveontologies/protege-proof-explanation
[6] https://github.com/liveontologies/puli

from an OWL ontology file. The computation can then be started using the *Generate explanation* button.

In the *Vocabulary* sub-tab, users can restrict the vocabulary used by the explanations, which can be seen in Figure 3. Here, the *Permitted vocabulary* is shown at the top, and the *Forbidden vocabulary* is at the bottom. The arrow buttons can be used to add or remove names to and from the permitted vocabulary, which can also be saved to and loaded from an external file. By default, the whole ontology vocabulary is *permitted*, but this can be changed in the plug-in preferences. If the entered missing entailment and vocabulary are not supported by the chosen explanation service, the *Generate explanation* button will be disabled and an explanatory message will be shown.

While the explanation is generated, a progress window is used to indicate the computation status and show additional information. The computation can be canceled by closing the window or clicking the *Cancel* button.

As a running example to illustrate the different explanation services, we consider a modified, incomplete version of the Pizza ontology (Alrabbaa et al. 2024). This version is missing some axioms to make it entail **SpicyAmerican** SubClassOf **SpicyPizza**. It will turn out that other axioms are missing in this ontology as well, and the plug-ins will help the user in adding those missing parts.

Counterexamples We visualize counterexamples using GraphStream,[7] a Java library for modeling, analyzing and visualizing graphs (see Figure 2). Its functionality allows not only to visualize models, but also to dynamically make changes to models when the ontology changes. In the generated graphs, domain elements are depicted as circles and object property connections as arrows. To facilitate exploring large counterexamples, the graphical view allows zooming and dragging of individual nodes or the whole graph. Additionally, we highlight elements that are of particular importance for understanding the generated model. For instance, each counterexample contains a *root element*, marked in black, which satisfies the class on the left-hand side of the missing subclass axiom.

For readability, only some of the classes satisfied by each domain element are shown; the number of classes can be adapted using the *Number of displayed classes* slider on the right panel of the counterexample view. To make the graphical representation of a counterexample more informative, we display classes in order from more specific to less specific, i.e. we would show the class **MeatyPizza** before the class **DomainThing**. When the user selects an element, the *Classes of selected element* list in the right panel displays all classes satisfied by the selected element.

The visualized model can also reveal missing disjointness axioms in the ontology. In Figure 2, the circled element is both a **MozzarellaTopping** and a **TomatoTopping**. The reason is a missing disjointness between **CheeseTopping** and **VegetableTopping**. The right panel allows the user to add new disjointness axioms as needed and visualize the result. For this, the user can select the corresponding classes and press *Add disjointnesses*. A disjointness axiom with the selected names is then added to the *Disjointnesses* list, as shown in the figure. By pressing the *Recompute example* button, the user sees an updated model with the new disjointness applied. If the user is not satisfied with the changes to the model resulting from the new axioms, they could delete them using the *Remove disjointnesses* button. Finally, axioms from the *Disjointnesses* list can be added to the active ontology with a click of the *Add all to ontology* button.

[7]https://graphstream-project.org

Abduction For our running example, Figure 3 shows an explanation based on complete signature-based abduction. This tells us that, to entail **SpicyAmerican** SubClassOf **SpicyPizza**, we could for example add the axiom **SpicyAmerican** SubClassOf **hasTopping** some **ChilliTopping** to the ontology.

For this kind of explanation, it can happen that the set of hypotheses is infinite. Therefore, the user can specify the number of results shown (10 is the default). When the *Generate explanation* button is clicked again, the same number of results is added to the current list. Using additional buttons shown at each hypothesis, the user can easily get an explanation why the hypothesis entails the missing entailment (using any of the explanation services for entailments), forbid the vocabulary of the selected hypothesis (to force the service to generate a new hypothesis), and add the hypothesis to the ontology (to repair the missing entailment).

To use the connection-minimal abduction, the FOL theorem prover Spass needs to be installed separately. An adapted version of Spass is required, which can be installed following the instructions on the web page of the abduction tool Capi.[8]

6 Evaluation of Evee Proof Services

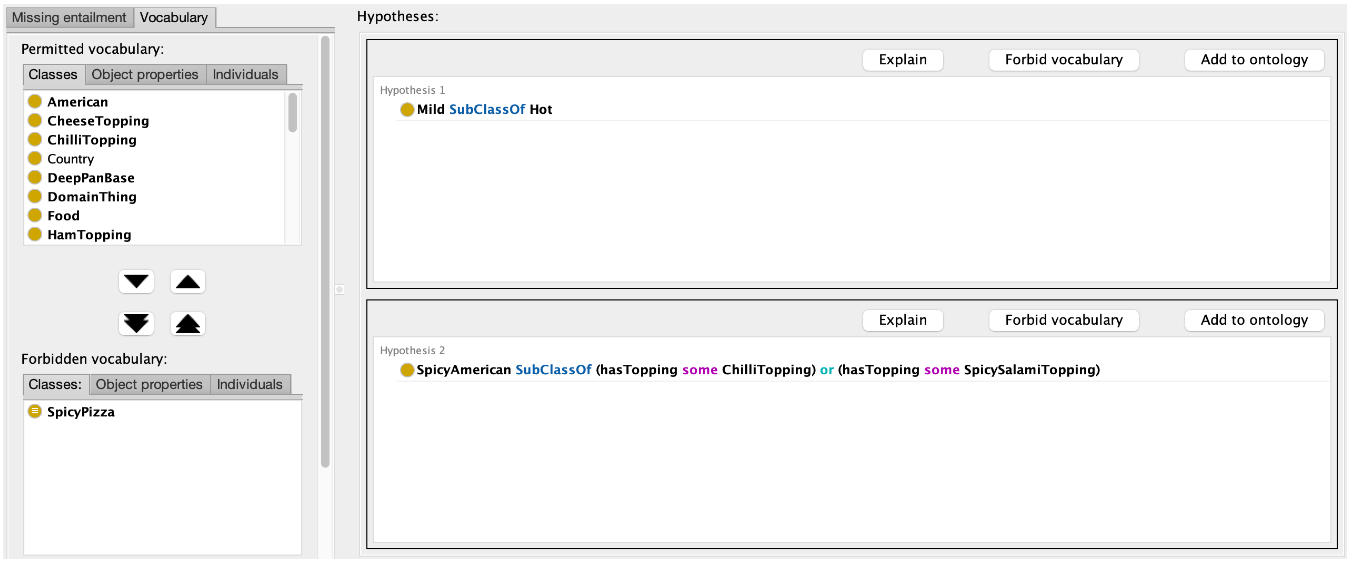
We conducted two user studies to evaluate the two types of explanations provided by Evee. The first was performed in 2022 and concerns the explanations of *entailments*, whereas the second one happened in 2024 and focused on the explanations of *missing entailments*. Both are *qualitative studies* with Protégé users conducted as online interviews using the think-aloud protocol, intended to get first feedback on the usefulness of these types of explanation plug-ins for Protégé, and what improvements are needed. In both cases the participants were informed about the purpose and content of the study and they have given informed consent.

In the first study, since the user interface for presenting proofs in Protégé already existed, our main goal was to compare the different proof methods, to find out their individual strengths and weaknesses.

6.1 Study Setup

Participant selection We interviewed ten researchers from the TU Dresden, all of whom often work with DLs. The participants were between 25 and 39 years old with a mean age of 32 ($SD = 4.36$). Two of the participants were female, and eight were male.

[8]When using this abduction plug-in for the first time, it will ask for the directory that Spass was installed to. This directory can later be changed in the Protégé preferences.



Permitted vocabulary:

American
CheeseTopping
ChilliTopping
Country
DeepPanBase
DomainThing
Food
HamTopping

Forbidden vocabulary:

SpicyPizza

Hypotheses:

Hypothesis 1
Mild SubClassOf Hot

Hypothesis 2
SpicyAmerican SubClassOf (hasTopping some ChilliTopping) or (hasTopping some SpicySalamiTopping)

Figure 3: Abduction results from LETHE for SpicyAmerican SubClassOf SpicyPizza with forbidden symbols SpicyPizza and hasSpiciness.

Environment setup Participants were asked to install PROTÉGÉ and to download EVEE before the interview. The sessions were held online using ZOOM.

Data collection We used CRYPTPAD[9] forms to collect the answers of the participants.

6.2 Study Procedure

Installation The experimenter guided the participant through the installation of EVEE.

Training The experimenter showed relevant features of EVEE to the participant.

Tasks In total there were 5 tasks. Each task specified an entailment and two proof methods, focusing on different aspects of proof composition that we wanted to analyze. Participants were asked to generate two proofs using these two methods, and compare them regarding their comprehensibility. Task 1 investigated the proof size of LETHE-based proofs. Task 2 utilized elimination proofs that are similar in size, but represent the same class expressions differently. Task 3 compared elimination- and consequence-based methods. In the remaining two tasks participants encountered progressively smaller proofs of the same entailment.

Closing questions Participants were asked at the end if there were additional information or features they would like to have seen.

6.3 Results

Task 1 (SebaceousGland SubClassOf HolocrineGland)[10] Participants proved the axiom using the LETHE-*based detailed proof method* and the LETHE-*based elimination proof with the heuristic method*. Six of the participants found the elimination proof easier to understand. One preferred the detailed proof, and three saw no difference between the two proofs. Most participants who noted no difference stated that it took them the same amount of effort to understand both proofs. For the other participants, the structure of the proofs, the axioms, and the types of expressions used were the reasons for preferring one proof over the other.

Task 2 (SebaceousGland SubClassOf HolocrineGland) Here, participants used the LETHE- *and* FAME-*based elimination proofs with the heuristic method*. Five of the participants found the FAME-based proof easier to understand. Three preferred the LETHE-based proof, and two saw no difference between the two proofs. Participants that preferred the FAME proof pointed out a contradiction caused by the interplay between the some and only constructors. For these participants, inferences like that are easier to follow. At the same time, the lack of such interplay in the LETHE proof, which used only some, is the reason why most of the other participants preferred that proof.

Task 3 (SpicyIceCream SubClassOf owl:Nothing)[11] Participants used the ELK *proof plug-in* (Kazakov, Klinov, and Stupnikov 2017) and the *size-optimized* FAME-*based elimination proof method*. Seven of the participants found the inference steps of the ELK proof easier to understand. Two preferred the inference steps of the elimination proof, and one saw no difference. However, opinions varied when participants were asked whether a large proof with simple inferences is easier to understand than a smaller proof with intricate inferences. Three of the participants agreed with the statement; three agreed

[9]https://cryptpad.fr/

[10]From the *Skin Physiology Ontology*: https://bioportal.bioontology.org/ontologies/SPO

[11]From another modified version of the Pizza ontology, see (Alrabbaa et al. 2024)

conditionally, provided a certain ratio between the size of the proof and the complexity of its inferences is not exceeded. One participant noted that, although they agreed, they believe that people would eventually become accustomed to the more complex inferences. Conversely, three of the participants completely disagreed, arguing that a shorter proof provides a clearer overview.

Task 4 (Analyzing SubClassOf Condition)[12]
Participants proved the axiom using the ELK *proof plug-in* and the *size-optimised* ELK *proof method*. All participants found the optimised ELK proof to be the easiest to understand, primarily because it was shorter, and uses simpler axioms and inference steps.

Task 5 (Analyzing SubClassOf Condition)
Now, participants proved the same axiom using the *size-optimised* ELK *proof method* and the LETHE-*based detailed proof method*. Unanimously, they agreed that the detailed proof is the easier of the two proofs. The reason, yet again, is that this proof is shorter and simpler.

The diversity of the answers confirms our assumption that the best method for generating proofs is often subjective. This justifies the inclusion of many different proof generation methods in EVEE.

The study took place in April 2022, and, since then, numerous improvements have been implemented in EVEE. Participants commented on the presentation and interaction with proofs in PROTÉGÉ. They asked for the possibility of expanding the entirety of a proof instead of one inference at a time, the capability of abbreviating entity names, and the option to choose between alternative forms of expressions. Since they are difficult to implement in PROTÉGÉ, these features have been addressed in our tool EVONNE (Méndez et al. 2023), a web-based application that builds upon EVEE and utilizes graph visualization techniques to explain DL entailments.

7 Evaluation of Missing Entailment Services

We conducted another user study to evaluate our explanations for missing entailments. We used a different study design since the goals of this study were different. In contrast to the existing proof interface, we newly implemented a tab for missing entailments in PROTÉGÉ and two interfaces for showing counterexamples and abduction results, so in the second study we wanted to find out whether the interface itself was intuitive for users of PROTÉGÉ with various backgrounds. Moreover, explaining missing entailment is a less common task than explaining entailments, so we also wanted to evaluate the suitability of the different types of explanations for this task.

For simplicity, we only included one abduction method (complete signature-based abduction) and one counterexample method (contrastive counterexamples) in this study. These methods represent the key aspects of counterexamples and abduction, making them suitable for evaluating the general usefulness of these modes of explanation. We again held individual online interviews, where participants used EVEE on a virtual machine, performed three tasks, and answered questions related to these tasks. Each task specified a missing entailment, and the participants were asked to fix the ontology to obtain the entailment. In the end, participants could provide feedback about what additional features they would like to have and whether they found EVEE useful and would recommend it. We ran a pilot study with four participants before conducting 18 interviews for the main study.

7.1 Study Setup

Participant selection We selected candidates who have previously used PROTÉGÉ, but we did not prescribe any level of proficiency. We advertised the study among the students of DL-related courses at TU Dresden and Vrije Universiteit Amsterdam, on several relevant mailing lists used by researchers and PROTÉGÉ users, on the KR Discord channel, on LinkedIn, and by direct e-mail to some selected researchers and ontology engineers. Each participant was given an online voucher with a value of 20 € in the currency of their country of residence.

Participants were between 23 and 74 years old, with a mean age of 37 (*SD* = 13.53). Of the 18 participants, three were female and 15 male. Their experience with PROTÉGÉ varied: some of the participants are working with PROTÉGÉ in industry, others use it for research and teaching, or for their university studies. Four participants said they use PROTÉGÉ only rarely, and four said they use it frequently. Fourteen participants regularly use the reasoning facilities of PROTÉGÉ, and one participant had never used a reasoner in PROTÉGÉ.

Environment setup We created four identical virtual machines (VMs) hosted by TU Dresden and installed PROTÉGÉ and EVEE on those. The experimenter connected to one VM and hosted the session using ZOOM. The participant connected to the ZOOM session, and then interacted with PROTÉGÉ on the VM via ZOOM remote control.

Data collection We used a LimeSurvey[13] questionnaire to collect demographic data, recorded video of the VM screen and audio of the ZOOM session, and took notes on essential information during the interview.

Data analysis We transcribed the recorded audio using Whisper[14] on a local server to facilitate the processing of participants' answers to our questions. We then used *inductive coding* (Rivas 2012): three experts (*coders*) agreed on *codes*, such as "graph presentation is good", based on the collected answers, and then independently assigned each answer to the appropriate codes.

7.2 Study Procedure

Initial survey Before the user study session, we asked participants to fill out a short survey with demographic questions.

[12]From the *BioTop ontology*: https://bioportal.bioontology.org/ontologies/BT

[13]https://www.limesurvey.org

[14]https://github.com/openai/whisper

Initial questions We asked participants about their experience with PROTÉGÉ such as how they usually use PROTÉGÉ, for which tasks they use PROTÉGÉ, and if they use reasoners in PROTÉGÉ.

Initial training The experimenter showed relevant features of PROTÉGÉ to make sure that every participant has the same basic knowledge, and then let the participant solve a training task and ask questions.

EVEE training The experimenter demonstrated the relevant features of *contrastive counterexamples* and *complete signature-based abduction*. The participant then solved the training task again using the two explanation services.

Tasks Since we needed ontologies that contain mistakes (missing entailments), we manually built three ontologies about pizzas, animals, and university lectures. Each participant was given one ontology and was asked to fix missing subclass entailments, like **Lion** SubClassOf **CarnivorousAnimal**, by extending or adapting the existing axioms about **Lion** and related classes. Each participant worked on the given ontology in three phases (each with different missing entailments): (N) using no EVEE plug-ins, (C) using only the counterexamples, and (A) using only abduction. The order of the phases was randomized, but Phase (N) was either the first or the last, and the EVEE training was given directly before (C) or (A), whichever came first. After each section, participants were asked which aspects they found especially easy or hard to do, and whether they thought that the explanation service was useful for fixing the ontologies.

Closing questions We asked participants which explanation service was more useful, which features they liked or not, whether they thought EVEE would be useful for their regular work with PROTÉGÉ, and which additional features they would like to have (see (Alrabbaa et al. 2024) for the precise question formulations).

7.3 Pilot Study

In the pilot study, every participant saw three different ontologies in the three phases. However, the study revealed that working with different problem domains significantly increased cognitive load, induced fatigue, and extended problem-solving time. Moreover, there was a natural preference towards simpler ontologies, which resulted in a biased comparison of the phases. Consequently, we modified the main study to focus on a single ontology for each participant and excluded the data of the pilot study from the analysis.

7.4 Results

The intercoder reliability was calculated from the codes of the three experts. The result is a *Fleiss' kappa* of 0.66, which is considered *substantial* (Landis and Koch 1977) since it exceed the threshold of 0.61. The agreement between the coders is therefore sufficiently high for the results to be interpretable. In order to merge the three codings of participants' statements, we considered a statement as given if it was coded as present by at least two of the coders.

Overall, most of the 18 participants in our sample preferred using the EVEE services for the ontology repair tasks, compared to having no explanation support (16 participants for (A) and 15 for (C)). Ten participants preferred the abduction service over the counterexamples, whereas six said the opposite. We now report more detailed feedback on the three phases of the study. Each phase has positive comments and negative comments, which are analyzed separately.

For Phase (N), six participants had positive comments. These comments convey that some tasks were so easy that they could be solved with the existing functionality of PROTÉGÉ. However, 16 participants commented negatively. The negative comments mainly pertained to the difficulties of finding all classes that are relevant for the task, of keeping everything in mind, and of understanding all relevant axioms.

In Phase (C), 15 participants had positive comments about the graphical presentation, including that it provides a good overview of the classes related to the task. Negative comments were mentioned by 17 participants, in particular that the graph was sometimes too confusing, and that they would need more time to get used to this kind of explanation. Moreover, they were concerned that the counterexamples would get less useful if too many classes are shown at the same time.

Phase (A) received positive comments from all participants, and negative comments from 17 of them. The first main positive comment is that the abduction service allows solving the task by directly adding a hypothesis to the ontology without the need to manually enter it. The second positive comment is that the step-wise restriction of the vocabulary allows exploring the space of possible hypothesis to improve the understanding of the problem. The main negative comment is that the interface for vocabulary selection was not easy to use. Moreover, it lacks a search functionality, such that participants had to find the relevant entities by scrolling through a long list. Other negative comments are that participants would need more time to get used to the abduction service, and perceived it as difficult to find a good vocabulary and hypothesis.

Fifteen participants suggested additional features for EVEE, most of them related to improving the vocabulary input interface and the connection with the core PROTÉGÉ functionality, e.g. selecting a missing entailment directly from the PROTÉGÉ *Classes* tab, or navigating from the explanation views back to the PROTÉGÉ views, potentially in a split view. For counterexamples, two participants mentioned that they would like a feature that lets them automatically add a suitable axiom to the ontology, similar to abduction. Finally, 10 participants said they would use the EVEE services in their own work with PROTÉGÉ, 15 said they would recommend them to others, and a few commented that the services would be useful for learning and teaching about OWL and description logics. The participants that would not use the services said it is because they are working with very large databases or knowledge graphs with only few logical axioms, or they would not have specific missing entailments in mind. Some also commented that one cannot use PROTÉGÉ with large amounts of assertions in practice.

8 Discussion and Conclusion

EVEE offers a collection of methods for generating explanations for (missing) entailments for OWL ontologies that can be explored in various ways in the ontology editor PROTÉGÉ. We believe that our plug-ins are an important step towards making consequences and missing entailments more transparent to ontology engineers, researchers, and students. Our missing entailment component uses the PROTÉGÉ extension point architecture, which means that other developers can easily add other abduction and counterexample methods.

Throughout the development of EVEE, we have conducted qualitative user studies to validate our approaches. While EVEE is of limited use for practitioners who only rarely use a reasoner, the general perception was positive. The responses of the participants suggest that there is no single best explanation method, but preferences are subjective. Thus, our approach of offering many different explanation methods allows each user to use the ones they are most comfortable with. There is still room for improvement, as evidenced by the number of feature suggestions. In the future, we plan to conduct quantitative studies on specifics aspects of the explanations.

We will continue to develop and improve the user interface of EVEE in the future. This also includes adding support for more expressive DLs, e.g. proof generation using consequence-based reasoners such as Sequoia (Bate et al. 2016), or counterexamples for (Horn) DLs beyond $\mathcal{EL}_\bot$.

Acknowledgments This work was supported by DFG grant 389792660 as part of TRR 248 – CPEC. We would like to thank Mikhail Kotlov for his help in implementing the EVEE counterexample component.

References

Alrabbaa, C., and Hieke, W. 2022. Explaining non-entailment by model transformation for the description logic $\mathcal{EL}$. In Artale, A.; Calvanese, D.; Wang, H.; and Zhang, X., eds., *Proceedings of the 11th International Joint Conference on Knowledge Graphs, IJCKG*, 1–9. ACM.

Alrabbaa, C.; Baader, F.; Borgwardt, S.; Koopmann, P.; and Kovtunova, A. 2020a. Finding small proofs for description logic entailments: Theory and practice. In Albert, E., and Kovács, L., eds., *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 73 of *EPiC Series in Computing*, 32–67. EasyChair.

Alrabbaa, C.; Baader, F.; Borgwardt, S.; Koopmann, P.; and Kovtunova, A. 2020b. On the complexity of finding good proofs for description logic entailments. In Borgwardt, S., and Meyer, T., eds., *Proceedings of the 33rd International Workshop on Description Logics (DL 2020)*, volume 2663 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Alrabbaa, C.; Baader, F.; Borgwardt, S.; Koopmann, P.; and Kovtunova, A. 2021. Finding good proofs for description logic entailments using recursive quality measures. In Platzer, A., and Sutcliffe, G., eds., *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction*, volume 12699 of *Lecture Notes in Computer Science*, 291–308. Springer.

Alrabbaa, C.; Borgwardt, S.; Friese, T.; Koopmann, P.; Méndez, J.; and Popovic, A. 2022a. On the eve of true explainability for OWL ontologies: Description logic proofs with Evee and Evonne. In Arieli, O.; Homola, M.; Jung, J. C.; and Mugnier, M., eds., *Proceedings of the 35th International Workshop on Description Logics (DL)*, volume 3263 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Alrabbaa, C.; Borgwardt, S.; Hirsch, A.; Knieriemen, N.; Kovtunova, A.; Rothermel, A. M.; and Wiehr, F. 2022b. In the head of the beholder: Comparing different proof representations. In Governatori, G., and Turhan, A., eds., *Rules and Reasoning - 6th International Joint Conference on Rules and Reasoning, RuleML+RR 2022, Berlin, Germany, September 26-28, 2022, Proceedings*, volume 13752 of *Lecture Notes in Computer Science*, 211–226. Springer.

Alrabbaa, C.; Borgwardt, S.; Friese, T.; Koopmann, P.; and Kotlov, M. 2023. Why not? explaining missing entailments with evee. In Kutz, O.; Lutz, C.; and Ozaki, A., eds., *Proceedings of the 36th International Workshop on Description Logics (DL 2023)*, volume 3515 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Alrabbaa, C.; Borgwardt, S.; Friese, T.; Hirsch, A.; Knieriemen, N.; Koopmann, P.; Kovtunova, A.; Krüger, A.; Popovic, A.; and Siahaan, I. 2024. Explaining Reasoning Results for OWL Ontologies with Evee - Resources. https://doi.org/10.5281/zenodo.11127460.

Baader, F.; Horrocks, I.; Lutz, C.; and Sattler, U. 2017a. *An Introduction to Description Logic*. Cambridge University Press.

Baader, F.; Horrocks, I.; Lutz, C.; and Sattler, U. 2017b. *An Introduction to Description Logic*. Cambridge University Press.

Baader, F.; Peñaloza, R.; and Suntisrivaraporn, B. 2007. Pinpointing in the description logic EL^+. In Hertzberg, J.; Beetz, M.; and Englert, R., eds., *KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI*, volume 4667 of *Lecture Notes in Computer Science*, 52–67. Springer.

Bate, A.; Motik, B.; Grau, B. C.; Simancik, F.; and Horrocks, I. 2016. Extending consequence-based reasoning to $\mathcal{SRIQ}$. In Baral, C.; Delgrande, J. P.; and Wolter, F., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016*, 187–196. AAAI Press.

Borgida, A.; Franconi, E.; and Horrocks, I. 2000. Explaining ALC subsumption. In Horn, W., ed., *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence*, 209–213. IOS Press.

Elsenbroich, C.; Kutz, O.; and Sattler, U. 2006. A case for abductive reasoning over ontologies. In Grau, B. C.; Hitzler, P.; Shankey, C.; and Wallace, E., eds., *Proceedings of the OWLED*06 Workshop on OWL: Experiences and Directions*, volume 216 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Haifani, F.; Koopmann, P.; Tourret, S.; and Weidenbach, C. 2022. Connection-minimal abduction in $\mathcal{EL}$ via translation to FOL. In Blanchette, J.; Kovács, L.; and Pattinson, D., eds., *Automated Reasoning*, 188–207. Cham: Springer International Publishing.

Hitzler, P.; Krötzsch, M.; Parsia, B.; Patel-Schneider, P. F.; and Rudolph, S., eds. 2012. *OWL 2 Web Ontology Language: Primer (Second Edition)*. W3C Recommendation. W3C. https://www.w3.org/TR/owl2-primer/.

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

Horridge, M.; Parsia, B.; and Sattler, U. 2009. Lemmas for justifications in OWL. In Grau, B. C.; Horrocks, I.; Motik, B.; and Sattler, U., eds., *Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, July 27-30, 2009*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Horridge, M.; Parsia, B.; and Sattler, U. 2010. Justification oriented proofs in OWL. In Patel-Schneider, P. F.; Pan, Y.; Hitzler, P.; Mika, P.; Zhang, L.; Pan, J. Z.; Horrocks, I.; and Glimm, B., eds., *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010*, volume 6496 of *Lecture Notes in Computer Science*, 354–369. Springer.

Horridge, M. 2011. *Justification Based Explanation in Ontologies*. Ph.D. Dissertation, University of Manchester, UK.

Kalyanpur, A.; Parsia, B.; Horridge, M.; and Sirin, E. 2007. Finding all justifications of OWL DL entailments. In Aberer, K.; Choi, K.; Noy, N. F.; Allemang, D.; Lee, K.; Nixon, L. J. B.; Golbeck, J.; Mika, P.; Maynard, D.; Mizoguchi, R.; Schreiber, G.; and Cudré-Mauroux, P., eds., *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, 267–280. Springer.

Kazakov, Y., and Klinov, P. 2014. Goal-directed tracing of inferences in $\mathcal{EL}$ ontologies. In Mika, P.; Tudorache, T.; Bernstein, A.; Welty, C.; Knoblock, C. A.; Vrandecic, D.; Groth, P.; Noy, N. F.; Janowicz, K.; and Goble, C. A., eds., *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II*, volume 8797 of *Lecture Notes in Computer Science*, 196–211. Springer.

Kazakov, Y.; Klinov, P.; and Stupnikov, A. 2017. Towards reusable explanation services in protege. In Artale, A.; Glimm, B.; and Kontchakov, R., eds., *Proceedings of the 30th International Workshop on Description Logics*, volume 1879 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Kazakov, Y.; Krötzsch, M.; and Simancik, F. 2014. The incredible ELK - from polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. *J. Autom. Reason.* 53(1):1–61.

Koopmann, P., and Schmidt, R. A. 2013. Forgetting concept and role symbols in $\mathcal{ALCH}$-ontologies. In McMillan, K. L.; Middeldorp, A.; and Voronkov, A., eds., *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19*, volume 8312 of *Lecture Notes in Computer Science*, 552–567. Springer.

Koopmann, P.; Del-Pinto, W.; Tourret, S.; and Schmidt, R. A. 2020. Signature-based abduction for expressive description logics. In Calvanese, D.; Erdem, E.; and Thielscher, M., eds., *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR*, 592–602.

Koopmann, P. 2020. LETHE: Forgetting and uniform interpolation for expressive description logics. *Künstliche Intell.* 34(3):381–387.

Landis, J. R., and Koch, G. G. 1977. The measurement of observer agreement for categorical data. *Biometrics* 33(1):159–174.

McGuinness, D. L. 1996. *Explaining Reasoning in Description Logics*. Ph.D. Dissertation, Rutgers University, NJ, USA.

Musen, M. A. 2015. The Protégé project: A look back and a look forward. *AI Matters* 1(4):4–12.

Méndez, J.; Alrabbaa, C.; Koopmann, P.; Langner, R.; Baader, F.; and Dachselt, R. 2023. Evonne: A visual tool for explaining reasoning with OWL ontologies and supporting interactive debugging. *Computer Graphics Forum* 42:e14730.

Rivas, C. 2012. Coding and analysing qualitative data. *Researching society and culture* 3(2012):367–392.

Schlobach, S., and Cornet, R. 2003. Non-standard reasoning services for the debugging of description logic terminologies. In Gottlob, G., and Walsh, T., eds., *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 355–362. Morgan Kaufmann.

Schlobach, S. 2004. Explaining subsumption by optimal interpolation. In Alferes, J. J., and Leite, J. A., eds., *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004*, volume 3229 of *Lecture Notes in Computer Science*, 413–425. Springer.

Zhao, Y., and Schmidt, R. A. 2018. FAME: An automated tool for semantic forgetting in expressive description logics. In Galmiche, D.; Schulz, S.; and Sebastiani, R., eds., *Automated Reasoning - 9th International Joint Conference, IJCAR 2018*, volume 10900 of *Lecture Notes in Computer Science*, 19–27. Springer.