On the Eve of True Explainability for OWL Ontologies: Description Logic Proofs with Evee and Evonne

Christian Alrabbaa[1], Stefan Borgwardt[1], Tom Friese[1], Patrick Koopmann[1], Julián Méndez[2] and Alexej Popovič[1]

[1]Institute of Theoretical Computer Science, Technische Universität Dresden, 01062 Dresden, Germany
[2]Interactive Media Lab Dresden, Technische Universität Dresden, 01062 Dresden, Germany

Abstract

When working with description logic ontologies, understanding entailments derived by a description logic reasoner is not always straightforward. So far, the standard ontology editor Protégé offers two services to help: (black-box) justifications for OWL 2 DL ontologies, and (glass-box) proofs for lightweight OWL EL ontologies, where the latter exploits the proof facilities of reasoner Elk. Since justifications are often insufficient in explaining inferences, there is thus only little tool support for explaining inferences in more expressive DLs. In this paper, we introduce Evee-libs, a Java library for computing proofs for DLs up to $\mathcal{ALCH}$, and Evee-protege, a collection of Protégé plugins for displaying those proofs in Protégé. We also give a short glimpse of the latest version of Evonne, a more advanced standalone application for displaying and interacting with proofs computed with Evee-libs.

Keywords

Proofs, Explanation, Protégé Plug-in, Forgetting

1. Introduction

Description logics (DLs) [1] have gained popularity through the standardization of the Web Ontology Language OWL[1] and the development of an OWL Java API [2], editing tools, and reasoners. This popularity comes with the need of explaining description logic reasoning to domain experts who are not familiar with DLs. We consider here the problem of explaining the entailment of logical consequences from DL ontologies (as opposed to explaining *non*-consequences, which requires other techniques [3, 4]). Research on explaining description logic inferences in the beginning considered *proofs* that provide detailed inference steps through which the consequence can be obtained [5, 6]. In many cases, especially in light-weight DLs, it can be sufficient to compute a *justification*, a minimal set of ontology axioms that entail the consequence [7, 8, 9]. However, if justifications become very large or the ontology is formulated in a more expressive DL, providing intermediate inference steps between a justification and its

DL 2022: 35th International Workshop on Description Logics, August 7–10, 2022, Haifa, Israel
christian.alrabbaa@tu-dresden.de (C. Alrabbaa); stefan.borgwardt@tu-dresden.de (S. Borgwardt); tom.friese@tu-dresden.de (T. Friese); patrick.koopmann@tu-dresden.de (P. Koopmann); julian.mendez2@tu-dresden.de (J. Méndez); alexej.popovic@tu-dresden.de (A. Popovič)
0000-0002-2925-1765 (C. Alrabbaa); 0000-0003-0924-8478 (S. Borgwardt); 0000-0001-5999-2583 (P. Koopmann); 0000-0003-1029-7656 (J. Méndez)

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

consequence may be required for understanding. Generating proofs is achievable via heuristic search for possible intermediate inferences [10], concept interpolation [11], forgetting [12], or directly using the inference rules underlying a consequence-based reasoner like Elk [13], which however only supports $\mathcal{EL}^+$. Recently, a plug-in for the ontology editor Protégé [14] was developed that shows proofs generated by Elk in a human-readable form [15]. Related work investigated the complexity of finding proofs that are optimal w.r.t. various measures, e.g. proof size or depth [12, 16, 17]. 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 [17].

In this paper, we present a collection of tools for proof generation, also for expressive DLs other than $\mathcal{EL}^+$. Our Java library Evee-libs (EVincing Expressive Entailments) implements various proof generation methods that form the basis for Evee-protege, a collection of Protégé plug-ins for interactively inspecting proofs, and Evonne, a more advanced proof visualization tool. This library extends our previously reported implementations of proof generation algorithms [12, 18] in various ways. One can extract proofs that are minimized w.r.t. size or depth from the output of Elk, generate *elimination proofs* [12, 18] that can additionally be optimized w.r.t. several parameters, or generate more *detailed proofs* directly via the resolution-based calculus used in the uniform interpolation (UI) tool Lethe [19]. Elimination proofs can support different DLs, depending on the UI tool that is used internally in a black-box fashion — our library currently uses a version of Fame [20] that supports $\mathcal{ALC}$, and Lethe, which supports $\mathcal{ALCH}$. While we discussed and evaluated the different methods for elimination proofs before [18], the method for detailed proofs is new. We provide a quantitative comparison of this method with the elimination proofs based on ontologies from BioPortal.

To use the proofs in practice to support ontology engineers with explanation services, we offer two tools to show proofs generated by Evee-libs to users. Evee-protege utilizes PULi (the Proof Utility Library) and the Protégé proofs plug-in [15] to show the proofs directly in the ontology editor Protégé. This might be the easiest way for users to use our services, and allowed us to perform a qualitative user study for our different proof methods presented in this paper. Evonne is a stand-alone web application, early prototypes of which were presented in [21, 22]. Since then, it has seen many visual improvements and new features, and offers now many different ways of displaying and interacting with proofs, together with a visualization of the modular structure of ontologies, and guidance for repairing undesired entailments. A larger publication discussing and evaluating the design decisions made in Evonne is currently under preparation. We here just give a brief overview of its proof visualization facilities. The source code and installation instructions for Evee-libs and Evee-protege can be found at https://github.com/de-tu-dresden-inf-lat/evee (version 0.1), where also the plugins can be downloaded. Evonne can be tested online and downloaded under https://imld.de/evonne/.

2. Preliminaries

We consider a consequence $\mathcal{O} \models \alpha$ that is to be explained, where $\mathcal{O}$ is a DL ontology [1] and α an axiom. A *justification* for α in $\mathcal{O}$ is a subset-minimal $\mathcal{J} \subseteq \mathcal{O}$ such that $\mathcal{J} \models \alpha$ [7]. In the worst case, there can be exponentially many justifications for an entailment. Following [12], we define *proofs* of $\mathcal{O} \models \alpha$ as finite, acyclic, directed hypergraphs, where vertices v are labeled

with axioms $\ell(v)$ and hyperedges are of the form (S, d), with S a tuple of vertices and d a vertex such that $\{\ell(v) \mid v \in S\} \models \ell(d)$; the leafs of a proof must be labeled by elements of $\mathcal{O}$ and the root by α. We depict hyperedges either using horizontal bars (e.g. $\frac{p \quad p \to q}{q}$) or using graphs with two kinds of nodes (see e.g. Figure 1). In this paper, all proofs are *trees*, i.e., no vertex can appear in the first component of multiple hyperedges. The *size* of such a proof is the number of its vertices (this is called *tree size* in [12, 17]), and its *depth* is the length of the longest path from a leaf to the root. Given a finite collection of valid inference steps (hyperedges), one can extract a (tree) proof that is composed from a subset of these steps and has minimal size or minimal depth in polynomial time [12, 17].

3. Proof Generation with Evee-libs

Evee-libs implements several (families of) proof generation methods with different advantages. They all use the Java interface `IProofGenerator<OWLAxiom>` with a method `getProof` that takes as input an `OWLAxiom` and returns an `IProof<OWLAxiom>` that can either be serialized into JSON format using a `JsonProofWriter`, explored using the method `getInferences`, or measured using an `IProofEvaluator`. For entailments that only depend on axioms in $\mathcal{EL}^+$, we implemented a method that optimizes proofs generated by Elk using various proof measures (in the *evee-elk-proof-extractor* library). For $\mathcal{ALC}$ and $\mathcal{ALCH}$, we implemented two other proof generation methods, *elimination proofs* (*evee-elimination-proofs*) and *Lethe-based proofs* (*evee-lethe-proof-extractor*).

3.1. Optimized Elk Proofs

We implemented the Dijkstra-like proof search algorithm from [12, 16, 17] that takes as input a set of instantiated inference rules that prove a target consequence and outputs an optimized proof. This works for any *recursive* measure of proof quality—intuitively, measures that can be computed recursively over the tree structure of the proof. Examples include the (tree) size weighted size, and depth of a proof. We apply this algorithm to the output of Elk, providing a different view on Elk proofs than the existing plug-in [15], which may show multiple ways to prove a DL axiom at the same time. Due to the fast reasoning with Elk, and because our algorithm runs only on the relatively small input extracted from Elk, this is the fasted proof generation method in Evee-libs. An experimental comparison of size-minimal Elk proofs and elimination proofs can be found in [12].

3.2. Elimination Proofs

For entailments that require more expressive DLs, such as $\mathcal{ALC}$, we implemented the forgetting-based approach from [12]. We now call those proofs *elimination proofs*, because they perform inferences by eliminating names. Inferences in an elimination proof look as follows:

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

where X is a concept or a role name, $\{\alpha_1, \dots, \alpha_n\} \models \beta$, no subset of $\{\alpha_1, \dots, \alpha_n\}$ entails β, and X does not occur in β. The general idea is that β is derived from the premises $\alpha_1, \dots, \alpha_n$

by performing inferences on X. Since usually the axiom we want to prove contains less names than the axioms it is derived from, this allows for a step-wise sequence of inferences from the ontology to the entailment. The result is a more high-level explanation than the other proof types. Elimination proofs do not depend on a specific logic – provided we have a method to compute elimination steps.

We compute elimination proofs by making use of existing tools for *forgetting*: given an ontology $\mathcal{O}$ and a name X, the result of forgetting X from $\mathcal{O}$ is an ontology $\mathcal{O}^{-X}$ entailed by $\mathcal{O}$ that does not contain X, and preserves all entailments of $\mathcal{O}$ that can be expressed without using X. Forgetting has bad worst-case complexities [23, 24], and may not always be successful, for instance if $\mathcal{O}$ contains cycles. Nonetheless, forgetting-tools like FAME and LETHE often perform well in practice when applied to realistic ontologies [19, 20]. Since forgetting is not always possible, these tools may introduce fresh names to simulate cyclic dependencies in the result. FAME may even not return a result at all because it is not complete.

We implemented three methods for computing elimination proofs using forgetting. All three start from a justification $\mathcal{J}$[2] for the entailment $\mathcal{O} \models \alpha$ to be explained, and then generate a sequence of ontologies $\mathcal{J} = \mathcal{O}_1, \ldots \mathcal{O}_n$ by eliminating the names that do not occur in α one after the other. For robustness, we internally use a wrapper for the chosen forgetting method — if the forgetting method "fails" to eliminate X (exceeds a fixed timeout set to 10 seconds by default, contains a definer, or fails for any other reason), we keep X and continue forgetting the next name. The wrapper also makes sure that forgetting results are cached. From this sequence of ontologies, the elimination proof is then constructed using justifications: specifically, for $i > 1$, we construct an inference step for $\alpha \in \mathcal{O}_i \setminus \mathcal{O}_{i-1}$ by taking an arbitrary[3] justification for α from $\mathcal{O}_{i-1}$. To further optimize the presentation, we added a technique to merge several inference steps into one: in an inference $\frac{\alpha_1 \quad \ldots \quad \alpha_n}{\beta}$, we may replace premises α_i with the premises used in the inference producing α_i, provided that the number of premises in the resulting inference step does not increase. This often leads to easier proofs in which several names are eliminated at the same time, e.g. using the following inference:

$$\frac{A \sqsubseteq \forall r.(B_1 \sqcap B_2 \sqcap B_3) \qquad \exists r.(B_1 \sqcap B_2 \sqcap B_3) \sqsubseteq B}{A \sqcap \exists r.\top \sqsubseteq B} \text{ eliminate } B_1, B_2, B_3$$

For generating the sequence of ontologies, we employed three strategies. 1) The *heuristic method* selects names using the heuristic described in [12], 2) the *name-optimized method* uses best-first search to optimize the order in which names are eliminated with the aim of finding a shortest sequence of ontologies (thus, essentially minimizing the number of names being eliminated), 3) the *size-optimized method* also uses best-first search, but optimizes the final proof rather based on other measures than the length of the sequence: size and weighted size. Forgetting role names early often leads to very short sequences and small proofs, but also hides inferences. For this reason, we added the additional constraint that role names can only be eliminated if they only occur in role restrictions without other names (e.g. $\exists r.\top$, $\forall r.\bot$). An example of an elimination proof generated using the heuristic approach is shown in Figure 1.

[2]This is computed using the black-box justification algorithm [25] with HERMIT [26].

[3]We also implemented the possibility of looking for the best justification in each case. However, this adversely affected the running time and did not lead to significantly smaller proofs.

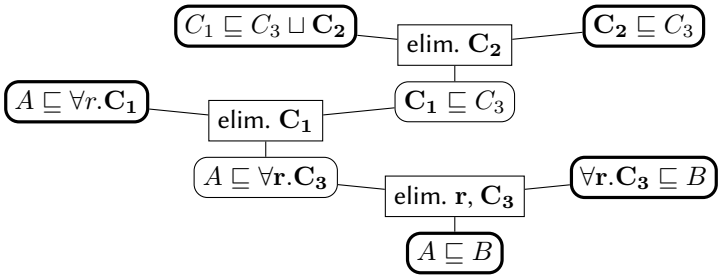

Figure 1: Elimination proof. Premises and conclusion highlighted, eliminated symbols in bold face.

Lethe and Fame may reformulate axioms that do not involve the forgotten name due normalization, which results in elimination inferences that do not really eliminate a symbol. For example, such a inference could produce $C \sqsubseteq D$ from $C \equiv D$. Note that the inference pattern above allows this. Because they improve readability of the proof, we keep those inferences, but call them "normalization" rather than elimination inferences.

We note that the version of Fame we use only supports $\mathcal{ALC}$-TBoxes, while the version of Lethe supports $\mathcal{ALCH}$-TBoxes. The generated proofs often look quite different because Lethe puts more emphasis on "beautifying" the resulting axioms, that is, reformulating them to look more human-readable. Thus, elimination proofs generated using Lethe may contain axioms that are easier to read than those generated using Fame. On the other hand, it can happen as a result of reformulating axioms that the relation between the premises and the conclusion of an inference step becomes less obvious.

Details and an experimental evaluation of the optimized algorithms can be found in [18].

3.3. Detailed Proofs Extracted from Lethe

Elimination proofs provide high-level explanations, since the detailed inferences for eliminating a symbol are hidden. For situations where this representation is too coarse, we provide a method for generating more detailed proofs, which uses the forgetting tool Lethe in a different way. When forgetting names in $\mathcal{ALCH}$ TBoxes, Lethe internally uses a consequence-based reasoning procedure based on a normalized representation of axioms and the calculus shown in Figure 2 [27]. To explain $\mathcal{O} \models A \sqsubseteq B$, we forget all names other than A and B from $\mathcal{O}$, and track the inferences performed by Lethe. The final set of clauses will contain a clause from which $A \sqsubseteq B$ can be straight-forwardly deduced.

We modified the code of Lethe so that it logs all performed inferences, including information on the side conditions. In addition to applying the inferences from Figure 2, normally Lethe repeatedly normalizes and denormalizes relevant parts of the ontology, and applies further simplifications to keep the current representation of the forgetting result small. For the logging to operate properly, we added a logging mode where all optimizations are turned off, and the entire ontology is normalized once at the beginning. This leads to slower reasoning performance, which is okay since we apply the method only on justifications rather than large ontologies. To avoid complicated inferences using the last rule in Figure 2, we make sure that inferences on role names are performed at the very end. At this stage, all relevant information for the

$$\frac{\top \sqsubseteq C_1 \sqcup A \qquad \top \sqsubseteq C_2 \sqcup \neg A}{\top \sqsubseteq C_1 \sqcup C_2} \qquad \frac{\top \sqsubseteq C \sqcup \exists r.D \qquad r \sqsubseteq s}{\top \sqsubseteq C \sqcup \exists s.D} \qquad \frac{\top \sqsubseteq C \sqcup \forall r.D \qquad s \sqsubseteq r}{\top \sqsubseteq C \sqcup \forall s.D}$$

$$\frac{\top \sqsubseteq C_1 \sqcup \exists r.D_1 \qquad \top \sqsubseteq C_2 \sqcup \forall s.D_2}{\top \sqsubseteq C_1 \sqcup C_2 \sqcup \exists r.(D_1 \sqcap D_2)} \text{ where } \mathcal{O} \models r \sqsubseteq s$$

$$\frac{\top \sqsubseteq C_1 \sqcup \forall r_1.D_1 \qquad \top \sqsubseteq C_2 \sqcup \forall r_2.D_2}{\top \sqsubseteq C_1 \sqcup C_2 \sqcup \forall s.(D_1 \sqcap D_2)} \text{ where } \mathcal{O} \models s \sqsubseteq r_1, s \sqsubseteq r_2$$

$$\frac{\top \sqsubseteq C_1 \sqcup \exists r_1.D_1 \quad \top \sqsubseteq C_2 \sqcup \forall r_2.D_2 \quad \dots \quad \top \sqsubseteq C_n \sqcup \forall r_n.D_n}{\top \sqsubseteq C_1 \sqcup \dots \sqcup C_n} \text{ where } \mathcal{O} \models \prod_{i=1}^{n} D_i \sqsubseteq \bot, \; \mathcal{O} \models r_1 \sqsubseteq r_2, \dots, r_1 \sqsubseteq r_n$$

Figure 2: Inference rules of LETHE for $\mathcal{ALCH}$ TBoxes. The last rule uses HERMIT for satisfiability testing.

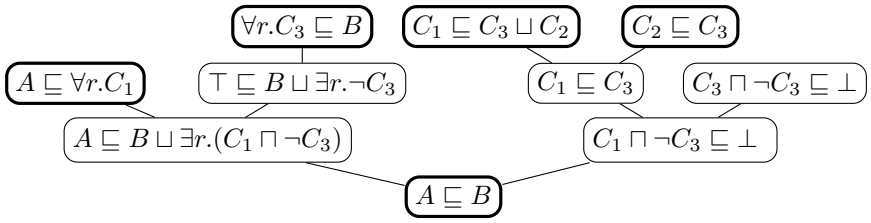

Figure 3: Detailed proof generated based on inference logging in LETHE (rules omitted).

unsatisfiability is usually already propagated into a single existential role restriction, which means that the inference has only one premise and the side conditions are not needed anymore. Indeed, we found that the resulting inference steps were usually very straightforward.

To produce a proof from the logging information, some additional steps are necessary: 1) the normalization as well as the reasoning procedure introduce fresh names, which have to be replaced again by the concepts they represent, 2) in the normal form, axioms are always represented as concept disjunctions, which is not very user-friendly, and 3) the normalized axioms still have to be linked to (non-normalized) input axioms, as well as to the conclusion. For 3), we again use justifications computed using HERMIT. For 2), we use the "beautification" functionality of LETHE to produce more human-friendly forgetting results. This involves applying some obvious simplifications of concepts ($A \sqcup \neg A \equiv A \sqcup \top \equiv \forall r.\top \equiv \top$, $A \sqcap \neg A \equiv A \sqcap \bot \equiv \exists r.\bot \equiv \bot$), pushing negations inside complex concepts, and moving disjuncts to the left-hand side of a GCI if this allows to eliminate further negation symbols (eg. $A \sqsubseteq B \sqcup \exists r.\neg C \Rightarrow A \sqcap \forall r.C \sqsubseteq B$). Applying those simplifications too rigorously, however, may again hide inferences performed by LETHE. We thus "beautify" the axioms *before* we replace the introduced names for 1), so that inferences performed on introduced names are still easily visible. The effect of this can be seen in the proof in Figure 3, where negations that would otherwise be "beautified away" are still shown. After transforming the logged inferences in this way, we use our Dijkstra-style proof search algorithm to extract a single proof.

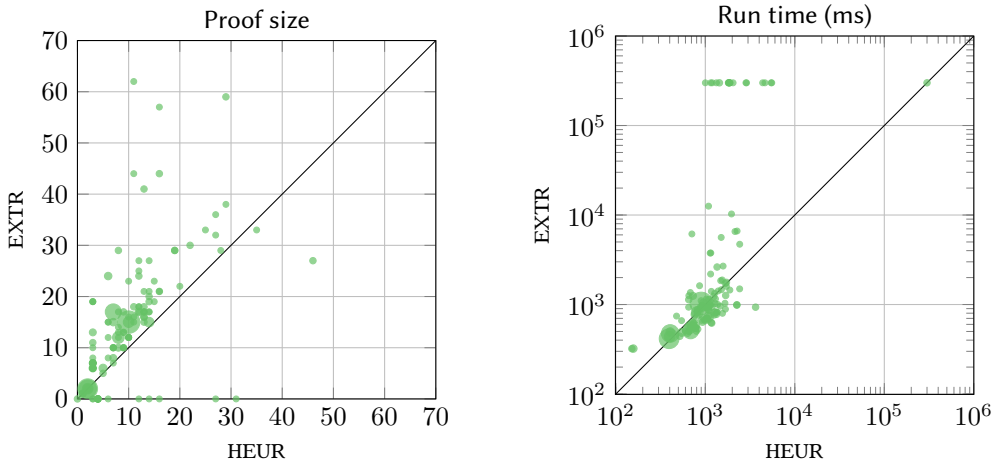

Figure 4: Run time and proof size for the heuristic elimination approach (using Lethe) vs. detailed proof extraction from Lethe. Marker size indicates how often each pattern occurred in the BioPortal snapshot.

3.3.1. Comparing Elimination Proofs to Lethe-based Proofs

We compared the elimination proofs constructed using Lethe and the heuristic method (HEUR) with the detailed proofs extracted directly from Lethe (EXTR) on a collection of proof tasks from the 2017 snapshot of BioPortal [28] (see also [18]). Up to isomorphism, we extracted 138 distinct $\mathcal{ALCH}$ proof tasks (consisting of an entailed axiom and the union of all its justifications) that are not fully expressible in $\mathcal{ELH}$, representing 327 different entailments within the BioPortal ontologies. We then computed proofs using the two methods, imposing a timeout of 5 minutes. This was successful for 325/327 entailments using HEUR. Due to the deactivated optimizations, EXTR timed out in 24/327 cases, which mostly happened when the signature of the task was large (more than ~18 symbols). Users should thus use EXTR only if the signature of the justification is sufficiently small. In Figure 4, we compare the proof size (left) and the run time (right) of both methods. On average, the proof size of EXTR was 90% higher than for HEUR, but the average inference step complexity[4] was 10% lower, supporting our expectation that EXTR computes more detailed proofs. Interestingly, the *maximum* step complexity of EXTR was 29% higher, indicating that the elimination proofs can hide complex inferences.

4. Evee Protégé Plugins

The easiest way to use our proof generators is via plug-ins for Protégé, a popular editor for OWL ontologies [14]. In Protégé, users can navigate the concept and role hierarchies of the ontology. When selecting a concept or role name, further information is shown, such as annotations and asserted or inferred equivalent concepts and superconcepts. To explain reasoning results in Protégé, one can click on the "?"-button next to an entailment. The standard explanation

[4]This is measured using the justification complexity from [29].

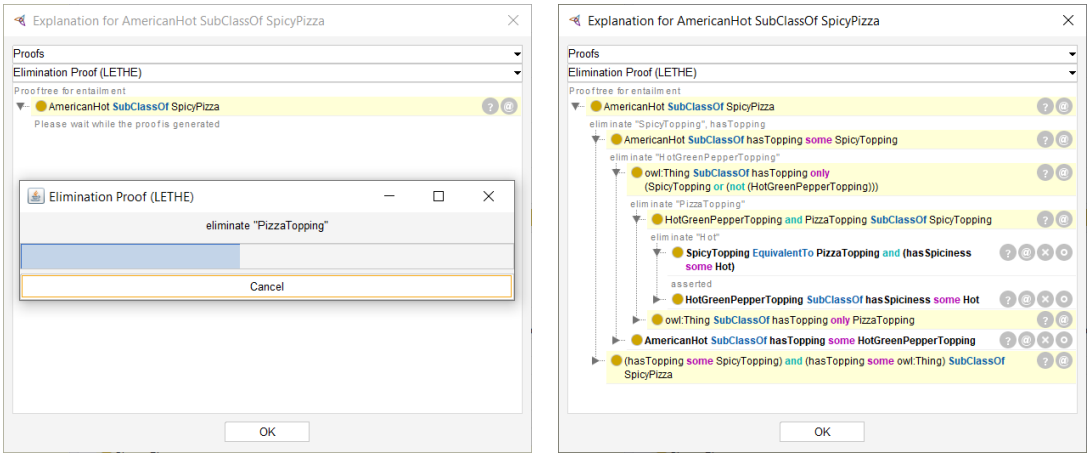

Figure 5: Loading screen during proof generation (left) and a finished proof (right)

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) [15].

Our proof generators are available in several Protégé plug-ins under the umbrella name Evee-protege, utilizing this existing functionality. The plug-ins can be installed by copying their JAR-files into Protégé's `plugin` folder, where the elimination proof methods are available in two versions for Lethe and Fame. Each proof generation method is registered as a *ProofService* for the *protege-proof-explanation* plug-in, which can be accessed from a drop-down list in the explanation dialog of Protégé (see Figure 5). Once selected, a ProofService starts generating the actual proof and a progress bar shows updates to the user. After proof generation is completed, the result can be explored interactively in a tree view. By clicking on the grey triangle next to an axiom, the proof steps that lead to this conclusion can be recursively unraveled.

Since proof generation can take some time to complete, users can abort the process by clicking the "Cancel"-button or by closing the progress window. The proof generators that are based on search will then assemble 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.

To facilitate modularity, the common functionalities of the proof services are implemented in *evee-protege-core*, which is not a plug-in itself (in the sense that it does not provide any extension to Protégé). Rather, evee-protege-core exports a collection of Java class files to Protégé that are required by the actual plug-ins. Each of our plug-ins that provides a ProofService uses these classes by inheriting from *AbstractEveeProofService* and instantiating one of our proof generators.

In order to add a ProofService for Protégé, the method *getProof* needs to be implemented. This method returns a *DynamicProof* which in turn needs to implement a method *getInferences* to provide the actual inferences to Protégé. Our plug-ins define this method in the class *EveeDynamicProofAdapter* of evee-protege-core. The dynamic nature of the proof comes in very handy as it allows for the proof generation to be executed in a background thread, and to later

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

update the proof explanation window with the constructed proof. To our knowledge [15], a dynamic proof was originally intended to be automatically updated after an asserted axiom is deleted. For this, a DynamicProof utilizes *ChangeListeners* which can be informed about any changes made to a proof that is currently shown to the user. Our plug-ins use this feature by first showing a proof consisting only of a single "placeholder" inference that is displayed when the proof generation is started. This inference has as its conclusion the axiom for which the proof is created and as its name a short message indicating that the proof is currently being computed (see Figure 5). Once the proof generation is complete, the method *inferencesChanged* is called on all registered ChangeListeners, which causes Protégé to update the UI with the actual proof. If no proof can be shown due to a cancellation or if an error occurred during proof generation, the "placeholder" inference will be updated accordingly.

4.1. User Study

We conducted a small qualitative user study to evaluate the usability of the plug-ins, and to validate the variety of proof generation options that Evee-protege provides. We held individual interviews online, where participants installed the plug-ins, performed five tasks,[7] and answered questions related to these tasks. Each task specified an entailment, for which the participants were asked to generate two proofs using different methods, and compare them in terms of ease of understanding (See the extended version [30] for all proofs used in the study). Lastly, participants could provide feedback about what additional features they would like to have. In total, we interviewed 10 participants, all of which often work with Description Logics.

In Tasks 1 and 2, participants were asked to compare proofs generated for an atomic concept inclusion in the Skin Phsyiology Ontology (SPO).[8] For Task 1, they compared the detailed Lethe-based proof with the elimination proof generated using Lethe with the heuristic method. Asked which proof is the easier to understand, 60% chose the elimination proof, 10% the detailed proof and 30% found no difference between the two. Most of the participants that found no difference stated that it took them the same amount of effort to understand both proofs. For the other participants, the structure of proofs and axioms and the type of expressions used were the reasons to prefer one proof over the other.

For Task 2, participants compared elimination proofs generated by Lethe and Fame using the heuristic method. Here, 50% prefered the Fame proof and 30% the Lethe proof in terms of understandability, while 20% of the participants found no differences between them. Participants that prefered the Fame proof pointed out a contradiction caused by the interplay between a universal and an existential restriction. 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 existential restrictions, is the reason why most of the other participants prefered this proof.

In Task 3, participants were asked to prove a concept unsatisfiability in a modified version of the pizza ontology[9] using the proofs provided by the original Elk proof plugin [15] and the size-optimized elimination proof using Fame. Asked which *inference steps* were easier to understand, 70% chose the Elk proof, 20% preferred the elimination proof, and 10% found

[7] Tasks and ontologies used in the study: https://cloud.perspicuous-computing.science/s/cy5Y3kj94AGYSDB
[8] https://bioportal.bioontology.org/ontologies/SPO
[9] https://protege.stanford.edu/ontologies/pizza/pizza.owl

no difference. However, when asked if they agree that a large proof with simple inferences is easier to understand than a smaller proof with intricate inferences, the answers varied. Some agreed with the statement, some agreed only as long as a certain ratio between the size of a proof and the difficulty of its inferences is not exceeded. Some pointed out that despite their agreement they think that people will eventually get used to the more complex inferences and how they work. On the other hand, there were some participants who totally disagreed with the statement. The main reason is that a short proof provides a better overview.

For the last two tasks, participants looked into proofs for an atomic concept inclusion in the BioTop ontology.[10] In Task 4, the comparison was between the original Elk plugin and the Elk proofs optimized for size. In Task 5, the comparison was between the optimized Elk Proof and detailed proof generated by Lethe. For both tasks, the answers were unanimous. Participants found the easier proofs to be the optimized Elk proof and the detailed proof, respectively. The reason is that the proofs get shorter with simpler axioms and inferences.

The diversity of the answers confirms our assumption that the best method for generating proofs is often subjective, which justifies having all the different proof generation methods in Evee-protege. Some participants commented on the presentation and interaction with the proofs in Protégé (e.g. the possibilities to expand the entire proof directly and not one inference at a time, to abbreviate names, and to choose between alternative forms of the same expression, etc.). Some of these comments have already been addressed in our tool Evonne.

5. Evonne

The proof visualization and interaction techniques in Evee-protege are restricted by the capabilities of Protégé's explanation service. More advanced proof exploration is offered by Evee's big sister Evonne [18, 21, 22], a web application that can be tried online and installed locally using Docker.[11] Evonne visualizes proofs with various interaction components and layouts to support the exploration of large proofs. The proof visualization is linked to a second view showing the context of the proof in the ontology by visualizing a subset (module) of the ontology that is relevant to the proof. We only give an overview of the proof visualisation here.

When starting Evonne, users need to create a new project and associate an OWL ontology file. They can then select an entailed atomic concept inclusion to be explained, and choose between the different proof generation methods described in Section 3. In addition, users can choose to provide a set of terms that they are familiar with (a signature) – axioms using only those terms are then assumed to be known to the user, and consequently are not explained in the proof view (see [18] for more details on this).

Proofs are shown as graphs with two kinds of vertices: colored vertices for axioms, gray ones for inference steps. By default, proofs are illustrated using a *tree layout*. However, it is also possible to present proofs in a *vertical layout*, placing axioms linearly below each other, with inferences represented through edges on the side (without the inference vertices). It is possible to automatically re-order vertices to minimize the distance between conclusion and premises in each step. Another layout option is a *bidirectional layout*, which is a tree layout

[10] https://bioportal.bioontology.org/ontologies/BT
[11] https://www.docker.com/

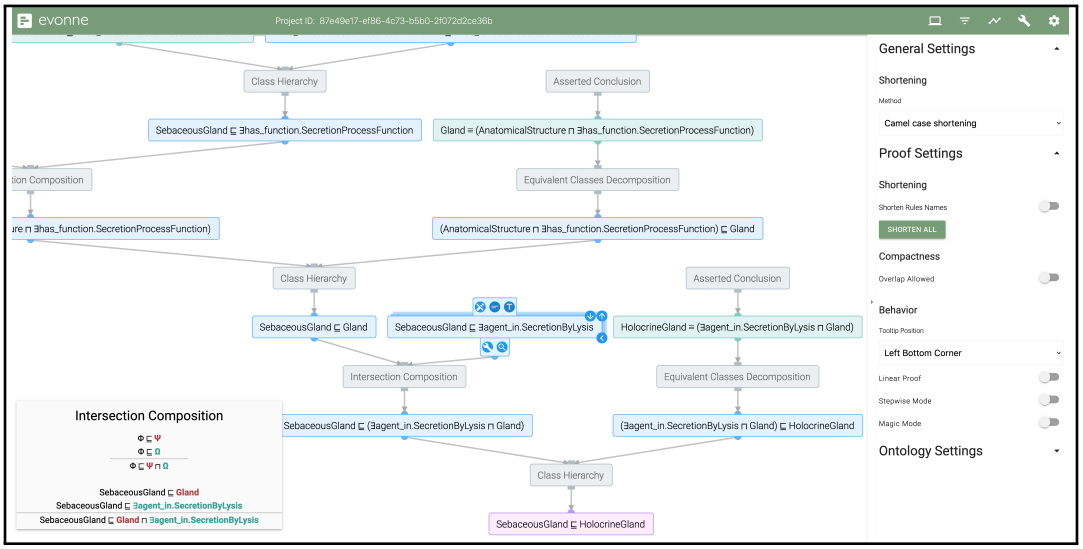

Figure 6: Evonne: Overview of the proof visualization component

where, initially, the entire proof is collapsed into a *magic vertex* that links the conclusion directly to its justification, and from which individual inference steps can be pulled out and pushed back from both directions. In all views, each vertex is equipped with multiple functionalities for exploring a proof. For proofs generated using Elk, clicking on an inference vertex shows the inference rule used, and the particular inference with relevant sub-elements highlighted in different colors (see Figure 6). From each axiom vertex, users can alter the visual structure of the proof by hiding branches, revealing the previous inference step from the current node or showing the entire proof. Each of the different proof layouts has specific controls in this regard - for example, the vertical layout does not show inference rules as nodes and instead allows inspection on demand. Additionally, trays of buttons above and below nodes can be used to manipulate the representation of axioms (by shortening names or using natural language), and to highlight justifications and diagnoses in the ontology view [21].

6. Conclusion

We have presented a collection of methods for generating DL proofs that can be explored in various ways. Future work includes adding support for specifying a "known" signature to Evee-protege, the development of proof generation methods for even more expressive logics (e.g. using consequence-based reasoners such as Sequoia [31]), and visualizing approaches for explaining *missing* entailments such as counter-interpretations [3] and abduction [4].

Acknowledgments

This work was supported by DFG grant 389792660 as part of TRR 248 – CPEC, and the DFG Research Training Group QuantLA, GRK 1763.

References

[1] F. Baader, I. Horrocks, C. Lutz, U. Sattler, An Introduction to Description Logic, Cambridge University Press, 2017. URL: http://dltextbook.org/. doi:10.1017/9781139025355.

[2] M. Horridge, S. Bechhofer, The OWL API: A java API for OWL ontologies, Semantic Web 2 (2011) 11–21. doi:10.3233/SW-2011-0025.

[3] C. Alrabbaa, W. Hieke, A. Turhan, Counter model transformation for explaining non-subsumption in EL, in: C. Beierle, M. Ragni, F. Stolzenburg, M. Thimm (Eds.), Proceedings of the 7th Workshop on Formal and Cognitive Reasoning co-located with the 44th German Conference on Artificial Intelligence (KI 2021), September 28, 2021, volume 2961 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 9–22. URL: http://ceur-ws.org/Vol-2961/paper_2.pdf.

[4] F. Haifani, P. Koopmann, S. Tourret, Abduction in $\mathcal{EL}$ via translation to FOL, in: R. A. Schmidt, C. Wernhard, Y. Zhao (Eds.), Proceedings of the Second Workshop on Second-Order Quantifier Elimination and Related Topics (SOQE 2021) associated with the 18th International Conference on Principles of Knowledge Representation and Reasoning (KR 2021), Online Event, November 4, 2021, volume 3009 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 46–58. URL: http://ceur-ws.org/Vol-3009/paper2.pdf.

[5] D. L. McGuinness, Explaining Reasoning in Description Logics, Ph.D. thesis, Rutgers University, NJ, USA, 1996. doi:10.7282/t3-q0c6-5305.

[6] A. Borgida, E. Franconi, I. Horrocks, Explaining ALC subsumption, in: W. Horn (Ed.), ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, IOS Press, 2000, pp. 209–213. URL: http://www.frontiersinai.com/ecai/ecai2000/pdf/p0209.pdf.

[7] S. Schlobach, R. Cornet, Non-standard reasoning services for the debugging of description logic terminologies, in: G. Gottlob, T. Walsh (Eds.), IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 2003, pp. 355–362. URL: http://ijcai.org/Proceedings/03/Papers/053.pdf.

[8] F. Baader, R. Peñaloza, B. Suntisrivaraporn, Pinpointing in the description logic EL^+, in: J. Hertzberg, M. Beetz, R. Englert (Eds.), KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI, volume 4667 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 52–67. doi:10.1007/978-3-540-74565-5_7.

[9] M. Horridge, Justification Based Explanation in Ontologies, Ph.D. thesis, University of Manchester, UK, 2011. URL: https://www.research.manchester.ac.uk/portal/files/54511395/FULL_TEXT.PDF.

[10] M. Horridge, B. Parsia, U. Sattler, Justification oriented proofs in OWL, in: P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, B. Glimm (Eds.), The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, volume 6496 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 354–369. doi:10.1007/978-3-642-17746-0_23.

[11] S. Schlobach, Explaining subsumption by optimal interpolation, in: J. J. Alferes, J. A. Leite (Eds.), Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, volume 3229 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 413–425. doi:10.1007/978-3-540-30227-8_35.

[12] C. Alrabbaa, F. Baader, S. Borgwardt, P. Koopmann, A. Kovtunova, Finding small proofs for description logic entailments: Theory and practice, in: E. Albert, L. Kovács (Eds.), LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, volume 73 of *EPiC Series in Computing*, EasyChair, 2020, pp. 32–67. doi:10.29007/nhpp.

[13] Y. Kazakov, M. Krötzsch, F. Simancik, The incredible ELK - from polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies, J. Autom. Reason. 53 (2014) 1–61. doi:10.1007/s10817-013-9296-3.

[14] M. A. Musen, The Protégé project: A look back and a look forward, AI Matters 1 (2015) 4–12. doi:10.1145/2757001.2757003.

[15] Y. Kazakov, P. Klinov, A. Stupnikov, Towards reusable explanation services in protege, in: A. Artale, B. Glimm, R. Kontchakov (Eds.), Proceedings of the 30th International Workshop on Description Logics, volume 1879 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2017. URL: http://ceur-ws.org/Vol-1879/paper31.pdf.

[16] C. Alrabbaa, F. Baader, S. Borgwardt, P. Koopmann, A. Kovtunova, On the complexity of finding good proofs for description logic entailments, in: S. Borgwardt, T. Meyer (Eds.), Proceedings of the 33rd International Workshop on Description Logics (DL 2020), volume 2663 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020. URL: http://ceur-ws.org/Vol-2663/paper-1.pdf.

[17] C. Alrabbaa, F. Baader, S. Borgwardt, P. Koopmann, A. Kovtunova, Finding good proofs for description logic entailments using recursive quality measures, in: A. Platzer, G. Sutcliffe (Eds.), Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, volume 12699 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 291–308. doi:10.1007/978-3-030-79876-5_17.

[18] C. Alrabbaa, F. Baader, S. Borgwardt, R. Dachselt, P. Koopmann, J. Méndez, Evonne: Interactive proof visualization for description logics (system description), in: Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Lecture Notes in Computer Science, Springer, 2022. URL: https://lat.inf.tu-dresden.de/research/papers/2022/ALBABODAKOME-IJCAR22.pdf, to appear.

[19] P. Koopmann, LETHE: Forgetting and uniform interpolation for expressive description logics, Künstliche Intell. 34 (2020) 381–387. doi:10.1007/s13218-020-00655-w.

[20] Y. Zhao, R. A. Schmidt, FAME: An automated tool for semantic forgetting in expressive description logics, in: D. Galmiche, S. Schulz, R. Sebastiani (Eds.), Automated Reasoning - 9th International Joint Conference, IJCAR 2018, volume 10900 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 19–27. doi:10.1007/978-3-319-94205-6_2.

[21] C. Alrabbaa, F. Baader, R. Dachselt, T. Flemisch, P. Koopmann, Visualising proofs and the modular structure of ontologies to support ontology repair, in: S. Borgwardt, T. Meyer (Eds.), Proceedings of the 33rd International Workshop on Description Logics (DL 2020), volume 2663 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020. URL: http://ceur-ws.org/Vol-2663/paper-2.pdf.

[22] T. Flemisch, R. Langner, C. Alrabbaa, R. Dachselt, Towards designing a tool for understanding proofs in ontologies through combined node-link diagrams, in: V. Ivanova, P. Lambrix, C. Pesquita, V. Wiens (Eds.), Proceedings of the Fifth International Workshop on Visualization and Interaction for Ontologies and Linked Data (VOILA 2020),

volume 2778 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 28–40. URL: http://ceur-ws.org/Vol-2778/paper3.pdf.

[23] C. Lutz, F. Wolter, Foundations for uniform interpolation and forgetting in expressive description logics, in: T. Walsh (Ed.), IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI/AAAI, 2011, pp. 989–995. doi:10.5591/978-1-57735-516-8/IJCAI11-170.

[24] N. Nikitina, S. Rudolph, (non-)succinctness of uniform interpolants of general terminologies in the description logic $\mathcal{EL}$, Artif. Intell. 215 (2014) 120–140. doi:10.1016/j.artint.2014.06.005.

[25] M. Horridge, B. Parsia, U. Sattler, Explaining inconsistencies in OWL ontologies, in: L. Godo, A. Pugliese (Eds.), Scalable Uncertainty Management, Third International Conference, SUM 2009, volume 5785 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 124–137. doi:10.1007/978-3-642-04388-8_11.

[26] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, HermiT: An OWL 2 reasoner, J. Autom. Reason. 53 (2014) 245–269. URL: https://doi.org/10.1007/s10817-014-9305-1.

[27] P. Koopmann, R. A. Schmidt, Forgetting concept and role symbols in $\mathcal{ALCH}$-ontologies, in: K. L. McMillan, A. Middeldorp, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, volume 8312 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 552–567. doi:10.1007/978-3-642-45221-5_37.

[28] N. Matentzoglu, B. Parsia, Bioportal snapshot 30.03.2017, 2017. doi:10.5281/zenodo.439510.

[29] M. Horridge, S. Bail, B. Parsia, U. Sattler, Toward cognitive support for OWL justifications, Knowl. Based Syst. 53 (2013) 66–79. doi:10.1016/j.knosys.2013.08.021.

[30] C. Alrabbaa, S. Borgwardt, T. Friese, P. Koopmann, J. Méndez, A. Popovič, On the Eve of True Explainability for OWL Ontologies: Description Logic Proofs with Evee and Evonne (Extended Version), 2022. doi:10.48550/ARXIV.2206.07711.

[31] A. Bate, B. Motik, B. C. Grau, F. Simancik, I. Horrocks, Extending consequence-based reasoning to $\mathcal{SRIQ}$, in: C. Baral, J. P. Delgrande, F. Wolter (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, AAAI Press, 2016, pp. 187–196. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12882.