Visualizing Proofs and the Modular Structure of Ontologies to Support Ontology Repair

Christian Alrabbaa[1], Franz Baader[1], Raimund Dachselt[2], Tamara Flemisch[2], and Patrick Koopmann[1]

[1] Institute of Theoretical Computer Science, TU Dresden, Germany
[2] Interactive Media Lab, TU Dresden, Germany

Abstract. The classical approach for repairing a Description Logic (DL) ontology in the sense of removing an unwanted consequence is to delete a minimal number of axioms from the ontology such that the resulting ontology no longer has the consequence. While there are automated tools for computing all possible such repairs, the user still needs to decide by hand which of the (potentially exponentially many) repairs to choose. In this paper, we argue that exploring a proof of the unwanted consequence may help us to locate other erroneous consequences within the proof, and thus allows us to make a more informed decision on which axioms to remove. In addition, we suggest that looking at the so-called atomic decomposition, which describes the modular structure of the ontology, enables us to judge the impact that removing a certain axiom has. Since both proofs and atomic decompositions of ontologies may be large, visual support for inspecting them is required. We describe a prototypical system that can visualize proofs and the atomic decomposition in an integrated visualization tool to support ontology debugging.

1 Introduction

We report here on first steps in a project whose goal it is to visualize various aspects of ontologies, with the purpose of supporting design, debugging, maintenance, and comprehension of ontologies. In a first prototype of our system, we concentrate on the visualization of proofs of consequences computed by a DL reasoner and the visualization of the modular structure of the ontology, and use ontology repair as an application scenario to guide our design decisions.

As is the case with all software artifacts, creating large ontologies is a difficult and error-prone process. However, the reasoning facilities provided by DL systems allow designers and users of DL-based ontologies to detect errors by finding incorrect consequences (called *defects* in the following). Such defects can be the inconsistency of the whole ontology, the unsatisfiability of a concept, or a derived subsumption relationship that obviously does not hold in the application domain (like amputation of finger being a subconcept of amputation of

hand [5]). The classical method for repairing an ontology in the sense of removing a given defect employs Reiter's approach for model-based diagnosis [28]. First, one computes all justifications for the defect, i.e., all minimal subsets of the ontology that have the defect as a consequence [31,4,16,15]. In order to get rid of the defect as a consequence of the whole ontology, it is then sufficient to remove from the ontology a hitting set of the justifications, i.e., a set of axioms that intersects with every justification [17,35,25]. Following [28,25], we call such a hitting set a *diagnosis* and the ontology obtained by removing it a *repair*.

Example 1. Let $\mathcal{T} = \{A \sqsubseteq B \sqcap C, B \sqsubseteq D, C \sqsubseteq D\}$ be a TBox , where $A \sqsubseteq D$ is an undesired consequence, i.e., a defect. The sets $\{A \sqsubseteq B \sqcap C, B \sqsubseteq D\}$ and $\{A \sqsubseteq B \sqcap C, C \sqsubseteq D\}$ are all justifications of the defect w.r.t. $\mathcal{T}$; and the sets $\{A \sqsubseteq B \sqcap C\}$ and $\{B \sqsubseteq D, C \sqsubseteq D\}$ are all subset-minimal diagnoses.

While all justifications and diagnoses of a given defect can be computed automatically, deciding which of the diagnoses to choose for constructing the actual repair requires human interaction. There are, however, some systems that support the user in making this choice, based on the impact that removing a certain diagnosis has on the ontology. One possibility to evaluate this impact is to count the number of subsumptions between concept names that are lost in this repair [23], but there are also other criteria for measuring the impact [17,26,27,34,35].

In the present paper, we propose to use not just the justifications of a given defect α when trying to repair it, but also proofs of α from the justifications. Basically, the idea is that, while navigating through such a proof, the user may find another defect β that is "closer" to the ontology axioms in the proof, and thus may pinpoint the "real" reason for the observed problem in a more precise way. Instead of repairing α, the idea is then to repair β first (possibly using the same approach recursively). If this also repairs α, then we are done. Otherwise, we can continue by looking at another proof of α from a new justification w.r.t. the new ontology.

It may happen, of course, that the user does not notice a defect other than the original one in the proof. For this case, we propose to use the modular structure of the ontology as described by the *atomic decomposition* [36] for judging the impact of a diagnosis. Basically, the atomic decomposition is a graph structure whose nodes are so-called $\top\bot^*$-modules [11] and whose edges describe dependencies between the modules. The idea is now to visualize the impact of a given diagnosis by showing which modules are affected by the removal of its axioms. The user can then decide, based on her knowledge of the modules, which diagnosis to prefer.

The realization of the ontology debugging approach sketched above requires a tool that can visualize proofs and atomic decompositions in an appropriate way. Whereas proofs are trees, atomic decompositions are graphs. Many graph visualization [13,12] and tree visualization techniques [32,33] as well as their combination [10] have been proposed in the literature, on which we could base our approach. For our application scenario, we had to select, adapt, and combine appropriate visualization techniques and to design the interaction with them. Our

prototype is designed for a *dual monitor* setup, and consists of two main components: *Defects Comprehension*, which provides an interactive view for exploring proofs of defects; and *Diagnoses Comprehension*, which provides an interactive view for displaying diagnoses, and their impact on the modular structures of the given ontology. The dual monitor setup allows for a seamless interaction of the two components, while providing sufficient space for displaying proofs and atomic decompositions in a comprehensible way.

2 Diagnoses, Repairs, Proofs, and Modular Structure

In this section we discuss our suggestion of a new workflow for repairing DL-based ontologies. In particular, we describe in more detail how proofs of defects and the modular structure of the ontology can support the repair process. The system that provides us with visual support for this approach is described in the next section.

In the following, we do not fix a particular ontology language. We only assume that the language can be used to formulate *axioms* (e.g., concept inclusions and assertions written in some DL). An *ontology* is a finite set of axioms. In addition, we assume that there is a monotonic *consequence relation* between ontologies and axioms, and write $\mathcal{O} \models \alpha$ to indicate that axiom α is a *consequence* of the ontology $\mathcal{O}$. In case the user thinks that the inferred consequence α actually does not hold in the application domain, we call α a *defect*. Under the assumption that the reasoning process that has produced the consequence is sound, the existence of a defect means that the ontology contains incorrect axioms, and thus needs to be repaired. We say that $\mathcal{O}' \subset \mathcal{O}$ is a *repair* of $\mathcal{O}$ w.r.t. the defect α if $\mathcal{O}' \not\models \alpha$.

Classical Repair The classical method for repairing an ontology is to remove some of its axioms, as defined above. However, given a detected defect α, it may not be obvious to the user which axioms in $\mathcal{O}$ are actually the culprits. In the amputation example mentioned in the introduction, while it is clear that "amputation of finger" should not be a subconcept of "amputation of hand," finding the responsible axioms is not easy since this requires a detailed understanding of the intricacies of the so-called SEP-triplet encoding employed by the modelers of the medical ontology SNOMED CT (see Fig. 1 in [5]).

The first step towards finding a possible repair automatically is to compute all *justifications* of the defect α, i.e., all sets $\mathcal{J} \subseteq \mathcal{O}$ such that $\mathcal{J} \models \alpha$, but $\mathcal{J}' \not\models \alpha$ for all strict subsets $\mathcal{J}' \subset \mathcal{J}$. In the worst case, α may have an exponential number of justifications (in the cardinality of $\mathcal{O}$). There is a large body of work on how to compute justifications for DL-based ontologies (some of which was cited in the introduction). In our prototype, we currently compute justifications by employing the functionalities provided by the Java-based Proof Utility Library PULi [22], which enumerates justifications using resolution.

In order to get rid of the defect α, it is then sufficient to remove (at least) one axiom from every justification. In fact, it is an obvious consequence of the

minimality of justifications that every subset of the ontology that has the consequence α must contain a justification. More formally, let $\mathcal{J}_1, \ldots, \mathcal{J}_n$ be all justifications of α. A *diagnosis* of α in $\mathcal{O}$ is a set $\mathcal{D} \subseteq \mathcal{O}$ that is a hitting set of $\mathcal{J}_1, \ldots, \mathcal{J}_n$, i.e., satisfies $\mathcal{D} \cap \mathcal{J}_i \neq \emptyset$ for $i = 1, \ldots, n$. As already shown by Reiter [28], if $\mathcal{D}$ is a diagnosis of α, then $\mathcal{O} \setminus \mathcal{D}$ is a repair of α, and every repair of α can be obtained in this way. In addition, there is also a 1–1 relationship between minimal diagnoses and maximal repairs. In the worst case, a defect may have an exponential number of (minimal) diagnoses, and thus an exponential number of (maximal) repairs. In our prototype, diagnoses are computed using a modified version of a tool for navigating answer-set programs, called INCA [2].

Proofs What amounts to a proof of an entailment $\mathcal{O} \models \alpha$ depends on the employed ontology language and formal proof system. Here we abstract from the specific proof system, and assume that a *proof of α from $\mathcal{O}$* is a tree whose nodes are labeled with axioms such that

1. the root has label α,
2. the leaves are labeled with elements of $\mathcal{O}$ or axioms β satisfying $\emptyset \models \beta$,
3. if a node with label β has as $n \geq 1$ children with labels $\beta_1, \ldots, \beta_n$, then $\{\beta_1, \ldots, \beta_n\} \models \beta$.

An example of such a proof, as displayed by our prototype, is given in Fig. 2 in the next section. It is a proof of the defect $\mathsf{SpicyIceCream} \sqsubseteq \bot$ entailed by a modified version of the *Pizza Ontology*.[1] This proof is based on the classification rules of the DL reasoner Elk [21], and its visualization contains auxiliary nodes that show names of the employed rules and indicate whether a leaf corresponds to an element of the ontology or to a rule application with an empty set of premises. There has been some work in the DL community on how to generate proofs of consequences [7,19,20,1], but usually with explanation of the proved consequences as use case [24,9,30]. In our prototype, we use proofs generated by the proof service available in the Elk reasoner [21,19,20], but minimize the proofs using the techniques described in [1].

In this paper, we propose to use proofs in the context of ontology repair. Assume that the user has found a defect α. As a first step, we compute a justification $\mathcal{J}$ of this defect, and then show a proof of the entailment $\mathcal{J} \models \alpha$ to the user. By exploring this proof, the user may notice that the proof contains another axiom β derived from $\mathcal{J}$ that is also a defect. Instead of repairing the defect α directly, we can now switch to repairing β. While this switch may not always be advantageous, we believe that it will often be, though this still needs to be investigated empirically. On the one hand, β may be derivable from a strict subset of $\mathcal{J}$, and then there are less axioms to choose from when removing an element from $\mathcal{J}$. On the other hand, the defect β may be more fundamental than α. For instance, consider the amputation example. The erroneous version of Snomed CT also had the consequence that "amputation of finger" is a subconcept of "amputation of arm." If the proof of this consequence contains the

[1] Available at https://lat.inf.tu-dresden.de/Evonne/PizzaOntology/

axioms stating that "amputation of finger" is "amputation of hand" and "amputation of hand" is "amputation of arm," then it is sensible to repair first one of these more specific defects.

The proof of the defect $\alpha = \mathsf{SpicyIceCream} \sqsubseteq \bot$ depicted in Fig. 2 provides us with another illustration of this idea. This proof contains the axiom $\beta = \mathsf{SpicyIceCream} \sqsubseteq \mathsf{Pizza}$, which appears to be the real reason for the observed problem, and thus should be repaired first. Also note that any repair of β also repairs α, but not vice versa. Thus, repairing β fixes the overall problem, whereas repairing α would have just dealt with a symptom of it.

At some point, the user will not find another defect to switch to in a proof, and thus the classical repair approach must be applied to the current defect. We propose to use the modular structure of the ontology to support the decision of which diagnosis to choose for repairing this defect.

The Modular Structure From a formal point of view, an ontology is just a flat set of axioms. In practice, however, ontologies usually consist of different components dealing with different topics, though this structure may not have been made explicit when defining the ontology. For instance, in the pizza ontology, there are axioms specifying fundamental aspects (such as: pizzas always have toppings), axioms defining different types of pizzas, axioms concerned with dietary issues, etc. In case the ontology at hand has not been structured into different such components in the design phase, one can use automated module extraction techniques to compute such a structure for a DL-based ontology. Intuitively, given an ontology $\mathcal{O}$ written in some DL and a set Σ of concept and role names (called *signature*), a module $\mathcal{M} \subseteq \mathcal{O}$ for Σ in $\mathcal{O}$ contains all axioms from $\mathcal{O}$ that are "relevant" for the meaning of the names in Σ.

In the DL literature, there is a large body of work defining different notions of modules, as well as algorithms for computing modules for some of these notions. In this paper, we focus on a specific type of modules called $\top\bot^*$-modules [11]. Such modules have the following useful properties:

1. For each signature Σ and ontology $\mathcal{O}$, there exists a unique $\top\bot^*$-module.
2. The $\top\bot^*$-module $\mathcal{M}$ for Σ in $\mathcal{O}$ preserves all Σ-entailments, i.e., for any axiom α that uses only names from Σ, it holds that $\mathcal{M} \models \alpha$ iff $\mathcal{O} \models \alpha$.
3. Each $\top\bot^*$-module $\mathcal{M}$ is *self-contained* in that it is also a module of the (possible larger) set of all concept and role names occurring in $\mathcal{M}$.
4. If we have $\Sigma_1 \subseteq \Sigma_2$ for two signatures, then also $\mathcal{M}_1 \subseteq \mathcal{M}_2$ for the corresponding modules.

The last two properties imply a hierarchical structure between all possible $\top\bot^*$-modules of $\mathcal{O}$, which can be represented in a compact way by the atomic decomposition [36]. For an ontology $\mathcal{O}$, the *atomic decomposition* is a pair $(\mathfrak{A}, \succeq)$, where $\mathfrak{A}$ is a partitioning of $\mathcal{O}$ into *atoms*, and $\succeq \subseteq \mathfrak{A} \times \mathfrak{A}$ is the *dependency relation*, which satisfies the following property: if an atom $\mathfrak{a}_1$ is a subset of some $\top\bot^*$-module $\mathcal{M}$ and $\mathfrak{a}_1 \succeq \mathfrak{a}_2$ (meaning $\mathfrak{a}_1$ depends on $\mathfrak{a}_2$), then also $\mathfrak{a}_2 \subseteq \mathcal{M}$. This means that an atom represents the module that consists of the union of

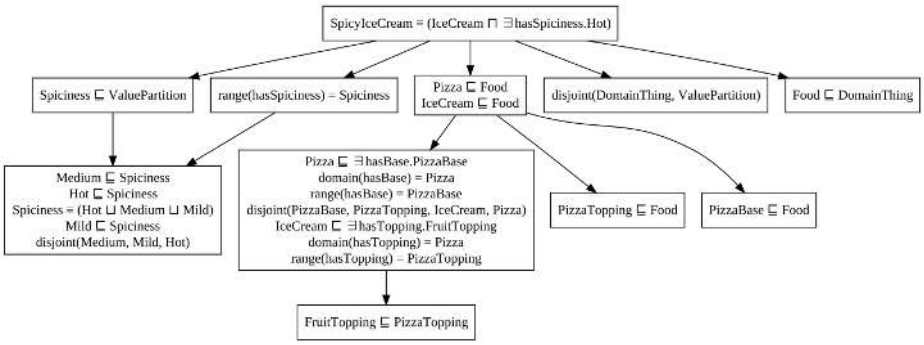

Fig. 1. Atomic decomposition of the $\top\bot^*$-module for the signature $\{\mathsf{SpicyIceCream}\}$ of a variant of the pizza ontology.

itself with all atoms it depends on. Since all $\top\bot^*$-modules can be obtained as the union of such atomic modules, the atomic decomposition indeed provides us with a compact representation of all $\top\bot^*$-modules. Atoms that are not in a dependency relation to each other can indeed be seen as being independent of each other: if we remove an atom and all atoms that depend on it, then the remaining modules are not impacted, that is, all entailments over their signatures are preserved.

An example of an atomic decomposition is shown in Fig. 1 for the subset of our modified pizza ontology that is the $\top\bot^*$-module of the signature $\{\mathsf{SpicyIceCream}\}$. The figure shows the Hasse-diagram of the partial order $\succeq$, i.e., $\succeq$ is the transitive closure of the relation $\rightarrow$ depicted there. To compute the atomic decomposition, we implemented the algorithm described in [36]. For extracting the $\top\bot^*$-modules, we used the tool provided by the OWL API [14].

The atomic decomposition can be used to support the user in choosing a diagnosis, and thus a repair, as follows. Given a diagnosis $\mathcal{D}$, we can show to which of the atoms its axioms belong. By going upward in the hierarchy, this allows us to see which other atoms (and thus $\top\bot^*$-modules) may be impacted by removing these axioms. However, minimizing the number of affected modules is only one possible criterion for making the decision. The ontology engineer might trust some modules more than others, either because of her knowledge about who wrote these axioms or because she knows this topic well enough to be certain that the axioms are correct. Thus, she may look only at diagnoses that concern other parts of the ontology. Also, it may be reasonable to assume that an axiom that interacts with many other axioms in the ontology is less likely to be erroneous, in the case that not many defects have been observed.

Example 2. Let us revisit our example concerned with defects in the pizza ontology. After inspecting the proof of the defect α, we have switched to repairing the more fundamental defect $\beta = \mathsf{SpicyIceCream} \sqsubseteq \mathsf{Pizza}$. It turns out that this defect has a single justification, consisting of three axioms, and thus there are three diagnoses, each consisting of one of the axioms:

- $\mathcal{D}_1 = \{\mathsf{domain}(\mathsf{hasTopping}) = \mathsf{Pizza}\}$
- $\mathcal{D}_2 = \{\mathsf{SpicyIceCream} \equiv \mathsf{IceCream} \sqcap \exists \mathsf{hasSpiciness.Hot}\}$
- $\mathcal{D}_3 = \{\mathsf{IceCream} \sqsubseteq \exists \mathsf{hasTopping.FruitTopping}\}$

Removing the diagnosis $\mathcal{D}_2$ would affect the least number of $\top\bot^*$-modules, since no other atom depends on the one consisting of this axiom. However, removing it would remove all information about SpicyIceCream from the ontology, and thus does not appear to be a good idea. The other two axioms belong to the same atom, and thus the atomic decomposition cannot help us choosing between them. Actually, while it is clear that it does not make sense to have both in the ontology, one could either allow things other than pizzas to have toppings or use another role to describe what is put on top of ice cream.

3 Visual Support for Ontology Debugging

Our tool, called Evonne (Enhanced visual ontology navigation and emendation), is a prototypical web application for ontology debugging of unwanted consequences. It visualizes proofs of defects occurring in ontologies as well as the impact of computed diagnoses based on the atomic decomposition. Currently, Evonne supports the lightweight ontology language OWL 2 EL. It is designed for a *dual monitor* setup, and consists of two main components: *Defects Comprehension*, which provides an interactive view for explaining defects through proofs exploration; and *Diagnoses Comprehension*, which provides an interactive view for showing diagnoses of defects and their impact on the modular structures of ontologies. As a central design goal we wanted to seamlessly integrate both views into a coherent tool with appropriate interactive functionality.

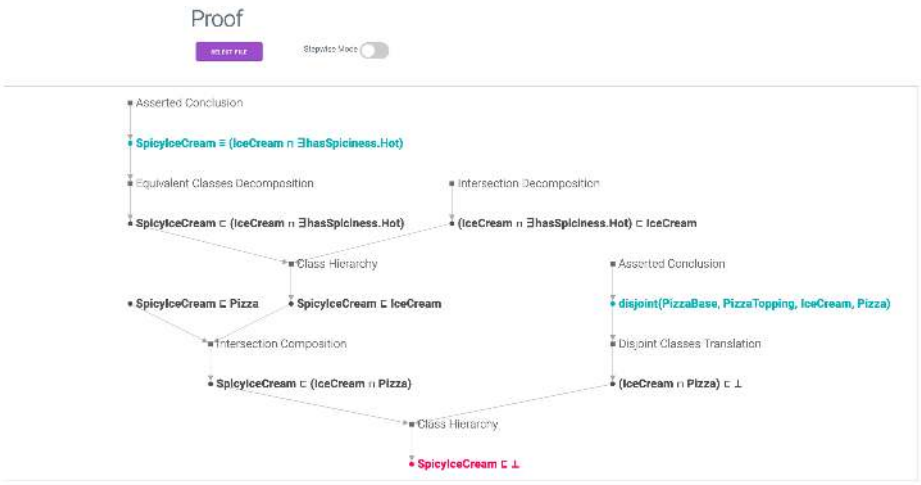

Fig. 2. Screenshot of Evonne showing the *Defects Comprehension* component

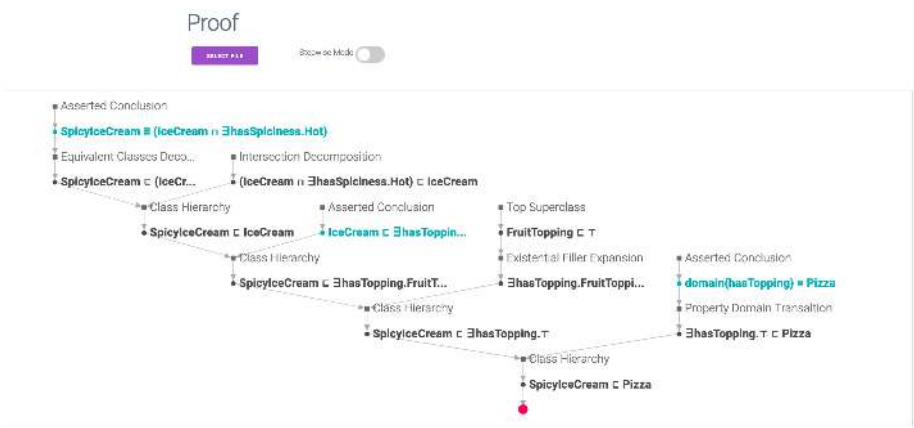

Fig. 3. Screenshot showing the collapsed proof of SpicyIceCream $\sqsubseteq$ Pizza in Fig. 2

The Defects Comprehension Component This component offers an interactive view for understanding defects through exploring and interacting with proofs (see Fig. 2). Its core element is a representation of the proof itself, which in our example shows unsatisfiability of SpicyIceCream. As argued in the previous section, by exploring and interacting with this proof, the user can find a more specific defect within the proof, i.e., SpicyIceCream $\sqsubseteq$ Pizza.

When visualizing proofs, the main problem is that they are usually very large, which makes it hard to display them in a sufficiently compact yet comprehensive manner. Protégé, for instance, contains an explanation plug-in that displays proofs provided by the ELK reasoner [20] as indentation lists. It shows all proofs for a consequence at once, which can potentially lead to visual clutter and a high cognitive load for the user. For our purposes, it is sufficient to display a single proof from a single justification, rather than multiple ones at the same time. In addition, the proofs shown in Evonne are minimal tree proofs, computed using the approach described in [1].

We visualize proofs as node-link diagrams since this encoding emphasizes the connection between nodes, their depth level, and the topological structure of the tree [33]. This representation ensures that (1) the premise of inference steps is localized, which makes it easier to focus on individual inferences; and (2) it puts emphasis on the different paths that lead to the final conclusion. Additionally, we use an axes-oriented layout for visualizing trees since it is extremely common and most users are familiar with its representation [32].

Since the minimal tree proofs displayed by Evonne can still be quite large, the tool is equipped with interactive elements, which provide users with multiple *navigation functionalities* that make large proofs easier to digest.

The button at the top of the component (see Fig. 2) allows the user to load a proof in GraphML format, which is then displayed within the component and can be explored. Selecting axioms by clicking on them reveals buttons (see

Fig. 4) with either navigation functionalities (discussed now), or communication functionalities (discussed later). The navigation buttons as well as toggling the *Stepwise Mode* switch allow the user to explore and traverse the proof in both directions, top-down and bottom-up. In our case, top-down and bottom-up refer to the tree structure and not to the position of nodes, i.e., top is the tree's root node whereas bottom means the tree's leaf nodes.

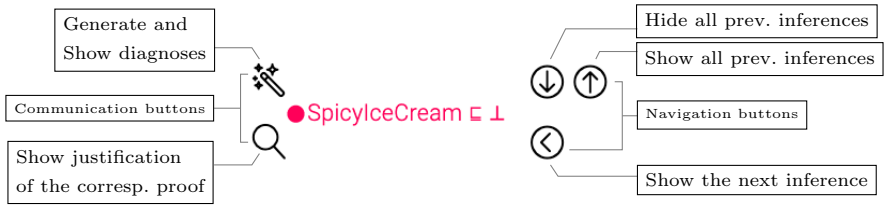

Fig. 4. Buttons associated with axioms in proofs

The top-down approach starts with showing only the final conclusion, and previous inferences can be revealed step-wise. This helps users to steer the exploration process in a way that focuses on specific paths that they deem important to understand the entailment. Thereby, the next inference is only revealed if the current visible part of the proof is understood. In contrast, when exploring the proof in a bottom-up manner, that is, starting from the premises, users can mark the parts of the proof they have already understood by collapsing them, and thereby decreasing the size of the proof. Again, this reduces the amount of displayed information while allowing users to focus on the next part of the proof during traversal. Users can adjust and traverse the proof according to their own preferences. At any stage, collapsed parts can be revisited.

In case a user finds a certain part of a proof particularly hard to comprehend, he can take this *sub-proof* and display it in isolation by clicking on the "delink" button on the connection to the following inference (see Fig. 5). This provides a localized view of all inferences leading to the chosen link, to be inspected separately and without distractions.

The large size of proofs is not the only factor that can make them hard to understand. Even a single application of an instance of a rule may be puzzling, either because the user is not familiar with the employed calculus or since the large size of the involved concept descriptions makes it hard to see why the concrete inference is an instance of a certain inference rule. To support comprehension of inference steps, Evonne is equipped with a tooltip that can be invoked by clicking on a specific rule (see Fig. 6) and provides (1) a display of the abstract rule using meta-variables for concepts, (2) a display of the currently considered instance below the abstract rule, (3) a color coding that clarifies how the instance was obtained.

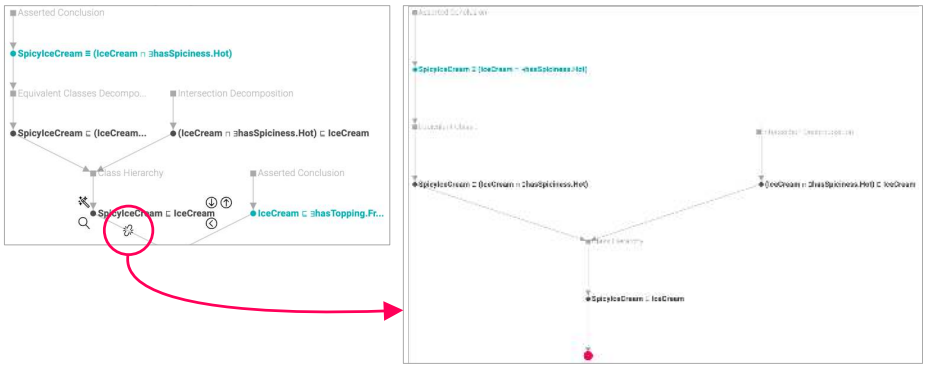

Fig. 5. Clicking on the *delink* icon next to the link in the left image isolates the sub-proof starting from this connection, which can be seen in the right image.

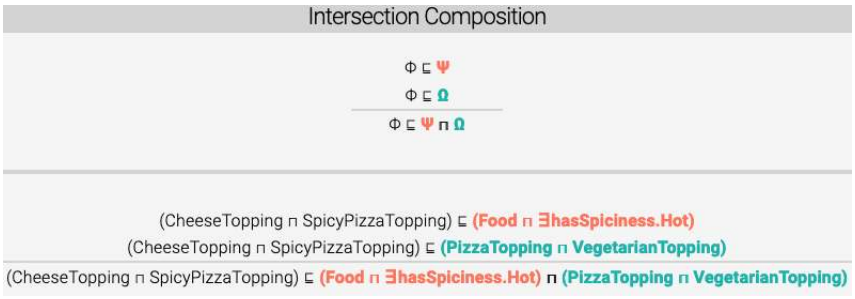

Fig. 6. Explanation of an instance of `Intersection Composition` displayed by Evonne.

The Diagnoses Comprehension Component This component is responsible for computing all diagnoses of defects and for showing their impact through the atomic decomposition. As shown in Fig. 7, the view of this component consists of two parts: the atomic decomposition (ontology) and the diagnoses part.

For the atomic decomposition, users can either employ the default layout provided by Evonne, which is based on the force-directed layout algorithm [18]; or they can rearrange the nodes into a more suitable layout, which can be saved for later use. Users can choose between two types of labels for the nodes in the atomic decomposition. The default labeling scheme uses axioms occurring in the corresponding atoms, while the other option is to label nodes with the signature of the corresponding atoms.

Diagnoses are shown in a *collapsed side menu*, grouped into *collapsed panels*, based on their size, to minimize their number on display. Hovering over a diagnosis triggers a color change of the corresponding axioms in the atomic decomposition. This Brushing and Linking [6] is a common interaction technique to explore relations between data [29]. It also changes the color of all nodes con-

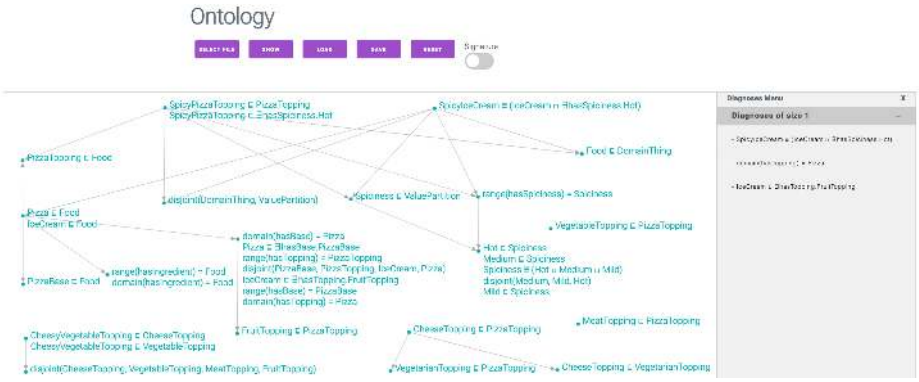

Fig. 7. Screenshot of Evonne showing the *Diagnoses Comprehension* component

taining these axioms, as well as of their predecessors, thereby highlighting the impact of a diagnosis on the $\top\bot^*$-modules of the ontology.

In our pizza example, after locating $\mathsf{SpicyIceCream} \sqsubseteq \mathsf{Pizza}$ as the more specific defect to be repaired, Evonne computes all diagnoses of this defect, i.e., $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$ (see Example 2), and displays them together with the atomic decomposition in the *Diagnoses Comprehension* view. Fig. 8 depicts how the colors of axioms and nodes in the atomic decomposition change when hovering over $\mathcal{D}_1$ (left) or $\mathcal{D}_2$ (right). This shows the impact of $\mathcal{D}_1$ to be more significant than the impact of $\mathcal{D}_2$, since more atoms are affected. Based on the observed impact, the atomic decomposition can also be used to determine which parts of the ontology might need to be adapted once a repair based on this diagnosis is generated.

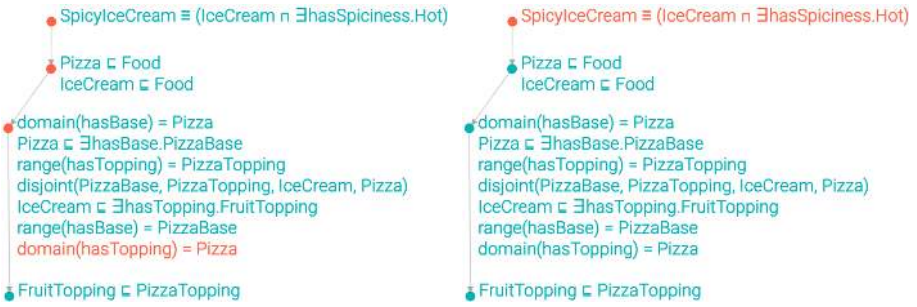

Fig. 8. Highlighted axioms and atoms for diagnoses $\mathcal{D}_1$ (left) and $\mathcal{D}_2$ (right).

We have designed two techniques for the interplay between the two main components of Evonne. While both are triggered in the *Defects Comprehension* view, by using the *communication buttons* shown in Fig. 4, the effects are shown in the *Diagnoses Comprehension* view. The first technique is *diagnoses highlight-*

ing. The user can select the final conclusion, or any entailment appearing in a proof, and ask Evonne to compute all diagnoses of this entailment. The second is *justification highlighting*. For any axiom β occurring in the proof, the justification that corresponds to the proof (i.e., the ontology axioms used in the proof to entail β) can be highlighted in the ontology view. This changes the color of the axioms occurring in the justification and the nodes containing them in the atomic decomposition – an important feature for repairing since it helps users to understand which part of the ontology, causing the defect, is currently being investigated.

4 Conclusion

We presented the interactive tool prototype Evonne that visualizes proofs of consequences and the modular structures of ontologies as described by the atomic decomposition. In this paper, we concentrated on ontology debugging as possible use case for our system, but the visual support it provides can also be employed in other settings, such as explaining why a correct consequence holds rather than repairing an incorrect one. To evaluate the usefulness of the debugging workflow sketched in this paper, we intend to perform a user study, which hopefully will also provide us with interesting new ideas for how to improve Evonne.

In the current version of Evonne, proofs and the atomic decomposition are precomputed separately and then provided as an input for the system. In the future, we want to seamlessly integrate these computations into our tool. There are, of course, many other improvements of Evonne that we intend to make, both regarding improved or additional functionality and how proofs and ontologies are displayed. In the context of repair, it would be usefull to be able to declare certain atoms or modules to be strict, in the sense that their axioms cannot be removed, and then compute only diagnoses that respect these declarations. In addition, we intend to support not only classical repairs (which remove axioms), but also more gentle kinds of repairs that weaken axioms [15,23,8,3]. It will be interesting to see how proofs can help locating parts of an axiom that need to be changed.

Acknowledgements. This work was partially supported by DFG grant 389792660 as part of TRR 248 (`https://perspicuous-computing.science`), and the DFG Research Training Group QuantLA, GRK 1763 (`https://lat.inf.tu-dresden.de/quantla`).

References

1. Alrabbaa, C., Baader, F., Borgwardt, S., Koopmann, P., Kovtunova, A.: Finding small proofs for description logic entailments: Theory and practice. In: LPAR-23. 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning. EPiC Series in Computing, vol. 73, pp. 32–67. EasyChair (2020)

2. Alrabbaa, C., Rudolph, S., Schweizer, L.: Faceted answer-set navigation. In: Rules and Reasoning - Second International Joint Conference, RuleML+RR 2018, Luxembourg, September 18-21, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11092, pp. 211–225. Springer (2018). https://doi.org/10.1007/978-3-319-99906-7_14
3. Baader, F., Kriegel, F., Nuradiansyah, A., Peñaloza, R.: Making repairs in description logics more gentle. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference, KR 2018, Tempe, Arizona, 30 October - 2 November 2018. pp. 319–328. AAAI Press (2018), https://aaai.org/ocs/index.php/KR/KR18/paper/view/18056
4. Baader, F., Peñaloza, R., Suntisrivaraporn, B.: Pinpointing in the description logic $\mathcal{EL}^+$. In: Hertzberg, J., Beetz, M., Englert, R. (eds.) KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI, KI 2007, Osnabrück, Germany, September 10-13, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4667, pp. 52–67. Springer (2007). https://doi.org/10.1007/978-3-540-74565-5_7, https://doi.org/10.1007/978-3-540-74565-5_7
5. Baader, F., Suntisrivaraporn, B.: Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In: Cornet, R., Spackman, K.A. (eds.) Proceedings of the Third International Conference on Knowledge Representation in Medicine, Phoenix, Arizona, USA, May 31st - June 2nd, 2008. CEUR Workshop Proceedings, vol. 410. CEUR-WS.org (2008), http://ceur-ws.org/Vol-410/Paper01.pdf
6. Becker, R.A., Cleveland, W.S.: Brushing Scatterplots. Technometrics **29**(2), 127–142 (May 1987). https://doi.org/10.1080/00401706.1987.10488204, http://amstat.tandfonline.com/doi/abs/10.1080/00401706.1987.10488204
7. Borgida, A., Franconi, E., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F.: Explaining $\mathcal{ALC}$ subsumption. In: Horn, W. (ed.) Proc. of the 14th Eur. Conf. on Artificial Intelligence (ECAI 2000). pp. 209–213. IOS Press (2000)
8. Du, J., Qi, G., Fu, X.: A practical fine-grained approach to resolving incoherent OWL 2 DL terminologies. In: Proc. of the 23rd ACM Int. Conf. on Information and Knowledge Management, (CIKM'14). pp. 919–928 (2014). https://doi.org/10.1145/2661829.2662046, http://doi.acm.org/10.1145/2661829.2662046
9. Engström, F., Nizamani, A.R., Strannegård, C.: Generating comprehensible explanations in description logic. In: Bienvenu, M., Ortiz, M., Rosati, R., Simkus, M. (eds.) Informal Proceedings of the 27th International Workshop on Description Logics, Vienna, Austria, July 17-20, 2014. CEUR Workshop Proceedings, vol. 1193, pp. 530–542. CEUR-WS.org (2014), http://ceur-ws.org/Vol-1193/paper_17.pdf
10. Graham, M., Kennedy, J.: A survey of multiple tree visualisation. Information Visualization **9**(4), 235–252 (Dec 2010). https://doi.org/10.1057/ivs.2009.29, https://doi.org/10.1057/ivs.2009.29
11. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: Theory and practice. J. Artif. Intell. Res. **31**, 273–318 (2008). https://doi.org/10.1613/jair.2375, https://doi.org/10.1613/jair.2375
12. Hadlak, S., Schumann, H., Schulz, H.J.: A Survey of Multi-faceted Graph Visualization. In: Borgo, R., Ganovelli, F., Viola, I. (eds.) Eurographics Conference on Visualization (EuroVis) - STARs. The Eurographics Association (2015). https://doi.org/10.2312/eurovisstar.20151109
13. Herman, I., Melancon, G., Marshall, M.S.: Graph visualization and navigation in information visualization: A survey. IEEE Transactions on Visualization and Computer Graphics **6**(1), 24–43 (2000)

14. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. J. Semant. Web **2**(1), 11–21 (2011)
15. Horridge, M., Parsia, B., Sattler, U.: Laconic and precise justifications in OWL. In: The Semantic Web - ISWC 2008, 7th International Semantic Web Conference, ISWC 2008, Karlsruhe, Germany, October 26-30, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5318, pp. 323–338. Springer (2008). https://doi.org/10.1007/978-3-540-88564-1_21
16. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Lecture Notes in Computer Science, vol. 4825, pp. 267–280. Springer (2007). https://doi.org/10.1007/978-3-540-76298-0_20
17. Kalyanpur, A., Parsia, B., Sirin, E., Grau, B.C.: Repairing unsatisfiable concepts in OWL ontologies. In: The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, ESWC 2006, Budva, Montenegro, June 11-14, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4011, pp. 170–184. Springer (2006). https://doi.org/10.1007/11762256_15
18. Kamada, T., Kawai, S.: An algorithm for drawing general undirected graphs. Information Processing Letters **31**(1), 7–15 (1989). https://doi.org/https://doi.org/10.1016/0020-0190(89)90102-6
19. Kazakov, Y., Klinov, P.: 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.T., Noy, N.F., Janowicz, K., Goble, C.A. (eds.) Proc. of the 13th International Semantic Web Conference (ISWC 2014). Lecture Notes in Computer Science, vol. 8797, pp. 196–211. Springer (2014)
20. Kazakov, Y., Klinov, P., Stupnikov, A.: Towards reusable explanation services in protege. In: Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, July 18-21, 2017. CEUR Workshop Proceedings, vol. 1879. CEUR-WS.org (2017), http://ceur-ws.org/Vol-1879/paper31.pdf
21. Kazakov, Y., Krötzsch, M., Simancik, F.: The incredible ELK – from polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. J. Autom. Reasoning **53**(1), 1–61 (2014). https://doi.org/10.1007/s10817-013-9296-3
22. Kazakov, Y., Skocovský, P.: Enumerating justifications using resolution. In: Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10900, pp. 609–626. Springer (2018). https://doi.org/10.1007/978-3-319-94205-6_40, https://doi.org/10.1007/978-3-319-94205-6_40
23. Lam, J.S.C., Sleeman, D.H., Pan, J.Z., Vasconcelos, W.W.: A fine-grained approach to resolving unsatisfiable ontologies. J. Data Semantics **10**, 62–95 (2008). https://doi.org/10.1007/978-3-540-77688-8_3, https://doi.org/10.1007/978-3-540-77688-8_3
24. McGuinness, D.L., Borgida, A.: Explaining subsumption in description logics. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes. pp. 816–821. Morgan Kaufmann (1995), http://ijcai.org/Proceedings/95-1/Papers/105.pdf
25. Moodley, K., Meyer, T., Varzinczak, I.J.: Root justifications for ontology repair. In: Rudolph, S., Gutiérrez, C. (eds.) Web Reasoning and Rule Systems - 5th International Conference, RR 2011, Galway, Ireland, August 29-30, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6902, pp.

275–280. Springer (2011). https://doi.org/10.1007/978-3-642-23580-1_24, https://doi.org/10.1007/978-3-642-23580-1_24
26. Nikitina, N., Rudolph, S., Glimm, B.: Interactive ontology revision. J. Web Semant. **12**, 118–130 (2012). https://doi.org/10.1016/j.websem.2011.12.002, https://doi.org/10.1016/j.websem.2011.12.002
27. Parsia, B., Sirin, E., Kalyanpur, A.: Debugging OWL ontologies. In: Proceedings of the 14th International Conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005. pp. 633–640. ACM (2005). https://doi.org/10.1145/1060745.1060837
28. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence **32**(1), 57–95 (1987)
29. Roberts, J.C.: State of the Art: Coordinated Multiple Views in Exploratory Visualization. In: Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV 2007). pp. 61–71 (Jul 2007). https://doi.org/10.1109/CMV.2007.20
30. Schiller, M.R.G., Schiller, F., Glimm, B.: Testing the adequacy of automated explanations of $\mathcal{EL}$ subsumptions. In: Artale, A., Glimm, B., Kontchakov, R. (eds.) Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, July 18-21, 2017. CEUR Workshop Proceedings, vol. 1879. CEUR-WS.org (2017), http://ceur-ws.org/Vol-1879/paper43.pdf
31. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Gottlob, G., Walsh, T. (eds.) IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003. pp. 355–362. Morgan Kaufmann (2003), http://ijcai.org/Proceedings/03/Papers/053.pdf
32. Schulz, H.: Treevis.net: A tree visualization reference. IEEE Computer Graphics and Applications **31**(6), 11–15 (Nov 2011). https://doi.org/10.1109/MCG.2011.103
33. Schulz, H., Hadlak, S., Schumann, H.: The design space of implicit hierarchy visualization: A survey. IEEE Transactions on Visualization and Computer Graphics **17**(4), 393–411 (April 2011). https://doi.org/10.1109/TVCG.2010.79
34. Shchekotykhin, K.M., Friedrich, G., Fleiss, P., Rodler, P.: Interactive ontology debugging: Two query strategies for efficient fault localization. J. Web Semant. **12**, 88–103 (2012). https://doi.org/10.1016/j.websem.2011.12.006, https://doi.org/10.1016/j.websem.2011.12.006
35. Thomas, E., Sleeman, D.H., Pan, J.Z., Reul, Q., Lam, J.S.C.: The Aberdeen University ontology reuse stack. In: Symbiotic Relationships between Semantic Web and Knowledge Engineering, Papers from the 2008 AAAI Spring Symposium, Technical Report SS-08-07, Stanford, California, USA, March 26-28, 2008. p. 83. AAAI (2008), http://www.aaai.org/Library/Symposia/Spring/2008/ss08-07-013.php
36. Vescovo, C.D., Parsia, B., Sattler, U., Schneider, T.: The modular structure of an ontology: Atomic decomposition. In: IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011. pp. 2232–2237. IJCAI/AAAI (2011). https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-372