Towards Visual Decision Support in Interactive Repair

Christian Alrabbaa[1], Franz Baader[1], Raimund Dachselt[2], Pratistha Kansakar[1], Julián Méndez[2] and Afshin Zanganeh[2]

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

Abstract

Whether carefully engineered or automatically learned, ontologies may still contain modeling errors. The problem of repairing Description Logic (DL) ontologies has been extensively studied, with a primary focus on minimizing information loss. A given ontology may have many different repairs that minimize information loss, and thus domain knowledge is needed to find the one that is correct in the given application domain. Consequently, interactive approaches have been proposed that involve users in the repair process, with the aim of reducing the number of user decisions required to obtain a valid repair. These methods usually assume that users are always able to make definitive decisions. However, users may face situations where the semantics of axioms alone are insufficient for them to determine whether an axiom should be retained or removed. In this work, we introduce an interactive repair approach that not only guides users through the exponential space of possible repairs while minimizing the decisions required, but also provides support in cases of uncertainty. Our approach augments the repair process with several *decision-support features*, including information about *important entailments*, *distance* to reachable repairs, and *structural differences* in the class hierarchy. We have implemented this approach both as a standalone command-line tool and as a web-based interactive visualization prototype that provides analytical capabilities not easily achievable otherwise.

Keywords

Ontology Repair, Decision Support, Visualization, Explanations

1. Introduction

Even carefully hand-crafted ontologies may contain errors, and this problem is exacerbated for ontologies built (semi)automatically using machine learning or information retrieval tools. For example, in earlier versions of the large medical ontology SNOMED CT,[1] which at that time was a large hand-crafted TBox written in the DL $\mathcal{EL}$, the concept "amputation of finger" was classified as a subconcept of "amputation of hand", which should clearly not be the case [1, 2]. As in the case of this example, errors in ontologies are usually noticed when reasoning produces a consequence that follows from the ontology, but does not hold in the application domain modeled by it. Getting rid of such an unwanted consequence by removing a minimal amount of information from the KB is usually called repair in the DL literature [3, 4, 5, 6, 7, 8, 9], where the repair approaches differ with respect to how "removing a minimal amount of information" is interpreted. Optimal classical repairs (in the terminology of [7]) are maximal subsets of the ontology that do not have the unwanted consequence. Following Reiter's approach for model-based diagnosis [10], such repairs can be produced as follows. First, compute all minimal subsets of the ontology that still have the unwanted consequence, usually called justifications in the DL literature [3, 11, 12, 13]. Then compute all minimal hitting sets of the collection of all justifications, i.e., sets that contain at least one element of each justification, called diagnoses. The optimal classical repairs are obtained by removing the elements of one of the diagnoses from the ontology.

DL 2026: 39th International Workshop on Description Logics, July 17–19, 2026, Lisbon, Portugal
christian.alrabbaa@tu-dresden.de (C. Alrabbaa); franz.baader@tu-dresden.de (F. Baader); raimund.dachselt@tu-dresden.de (R. Dachselt); pratistha.kansakar@mailbox.tu-dresden.de (P. Kansakar); julian.mendez2@tu-dresden.de (J. Méndez); afshin.zanganeh@tu-dresden.de (A. Zanganeh)
0000-0002-2925-1765 (C. Alrabbaa); 0000-0002-4049-221X (F. Baader); 0000-0002-2176-876X (R. Dachselt); 0009-0007-0538-839X (P. Kansakar); 0000-0003-1029-7656 (J. Méndez); 0009-0003-8678-5821 (A. Zanganeh)

[1]https://www.snomed.org/

This and also the other cited repair approaches may in the worst case compute exponentially many repairs in the size of the original ontology, of which some will not even make sense in the application domain. It is not a good idea to compute all these repairs first, and then ask the ontology engineer building the ontology or the domain expert that has spotted the error (both called users in the following) to choose, among potentially exponentially many computed repairs, one that does make sense. For this reason, interactive repair approaches have been developed [14, 15, 16, 17, 18], where the user makes certain decisions (e.g., whether an axiom in a justification should be part of the diagnosis or not), which prune the search space for an appropriate application-conforming repair.

The order in which the user is asked to make the relevant decisions influences how many decisions need to be made overall before a repair is reached. For this reason, several of the interactive repair approaches cited above are concerned with defining appropriate orders, and this issue is also addressed by our repair tools. For example, when computing a hitting set, it makes sense to favor axioms that occur in many justifications, though this heuristic does not guarantee that a *minimal* hitting set is found. This kind of frequency sorting of axioms was, e.g., used in [4], and is also employed in our repair tools. In addition, we consider an order that uses the number of occurrences of axioms in diagnoses to compute an entropy score. As shown in [19], ordering axioms by entropy score helps to reduce the number of questions the user needs to answer.

Another problem that requires attention is that users may need help in deciding which decision (keep the axiom or remove it) to make. We had already observed this issue when working with our interactive ontology completion tool based on formal concept analysis [20], where users sometimes made wrong decisions, which had to be retracted in an appropriate way [21]. To address this problem, our interactive repair tools are equipped with several decision-support features, which provide the user with additional information that they cannot obtain from simply looking at the axiom in question. On the one hand, we allow the user to define certain important entailments that should be retained as far as possible. The user is then informed of the effect of their decision on these entailments: for each such entailment, our system computes the number of still reachable repairs in which the entailment is preserved after the decision. On the other hand, we provide information on how much the concept hierarchy or the ontology itself changes based on the decision to remove or keep an axiom. For the hierarchy, the information refers to which parts of the hierarchy are affected by the current choice and how these parts change. For the ontology itself, it is measured how much the still reachable best repair differs from the the current ontology w.r.t. Hamming distance, where "best" is based on the number of retained axioms.

We have implemented these ideas in two interactive repair tools. The first is a standalone command-line tool, which displays the additional information generated by our decision-support features in a text-based way. The second is a web-based interactive prototype that visually supports the user along the repair process through a number of views: (1) a list of justification axioms to decide upon, (2) a decision tree that can be expanded on demand, (3) multivariate views that visually encode the rates of preserving user-defined important entailments, (4) a class hierarchy diff view detailing the changes resulting from removing an axiom, and (5) comparative bars and shortcuts to the discussed distances and corresponding best repairs. Together, these views provide analytical capabilities not easily achievable otherwise. We provide static examples (Hospital, Pizza) on a live version of our visualization prototype at https://imldresden.github.io/vds-repair.

2. Ontology Repair with Decision Support

In this section, we present our interactive repair approach by first providing an overview and then describing the individual components in detail. Although the approach itself is not dependent on a specific DL, we restrict our attention to $\mathcal{EL}^+_\bot$ to obtain a tractable implementation using ELK. Formally, we focus only on repairing the terminological part of an ontology; however, for readability, we occasionally use the terms TBox and ontology interchangeably.

Algorithm 1: Interactive Ontology Repair with Decision Support – Basic Algorithm

Input: $\mathcal{T}, \delta, \mathcal{E}, sortMethod$
Output: $\mathcal{R}$
$\mathfrak{J} \coloneqq \text{compAllJustifications}(\mathcal{T}, \delta)$; $\mathfrak{D} \coloneqq \text{compAllMinDiagnoses}(\mathfrak{J})$; $toRemove \coloneqq \emptyset$;
$\mathfrak{R} \coloneqq \text{compAllMaxRepairs}(\mathcal{T}, \mathfrak{D})$; $\mathcal{M} \coloneqq \text{compSortedAxioms}(\mathfrak{J}, \mathfrak{D}, sortMethod)$;
$\mathcal{R} \coloneqq \mathcal{T}$;
foreach $\alpha \in \mathcal{M}$ **do**
  $ans \coloneqq \text{getUserAnswer}(\alpha)$;
  while $ans \notin \{\mathsf{Yes}, \mathsf{No}\}$ **do**
    $\mathcal{F} \coloneqq \text{getUserSelection}(\{\mathsf{Rate}, \mathsf{Diff}, \mathsf{Dist}\})$;
    if $\mathsf{Rate} \in \mathcal{F}$ **then**
      $\text{display}(\text{compRates}(\mathcal{R}, \mathfrak{R}, \mathcal{E}, \alpha))$;
    if $\mathsf{Diff} \in \mathcal{F}$ **then**
      $\text{display}(\text{compDifferences}(\mathcal{R}, \alpha))$;
    if $\mathsf{Dist} \in \mathcal{F}$ **then**
      $\text{display}(\text{compDistances}(\mathcal{R}, \text{getBestRepairs}(\mathcal{R}, \mathfrak{R}, \alpha)))$;
    $ans \coloneqq \text{getUserAnswer}(\alpha)$;
  if $ans = \mathsf{No}$ **then**
    $toRemove \coloneqq toRemove \cup \{\alpha\}$;
  $\mathcal{R} \coloneqq \mathcal{T} \setminus toRemove$;
  if $\exists \mathcal{R}' \in \mathfrak{R}$ *such that* $\mathcal{R} \subseteq \mathcal{R}'$ **then**
    return $\mathcal{R}$;
return $\emptyset$;

2.1. Repair Workflow

We assume that we are given a TBox $\mathcal{T}$ and an undesired consequence δ (defect) of $\mathcal{T}$, i.e., $\mathcal{T} \models \delta$. In addition, we assume that the user specifies a set of important entailments $\mathcal{E}$ that are to be preserved as much as possible. As shown in Algorithm 1, we start by computing the set $\mathfrak{J}$ of all *justifications* [3, 11, 12, 22] of $\mathcal{T} \models \delta$. To construct a repair $\mathcal{R}$, the user must decide which axioms from the original ontology should be removed ($toRemove$), in particular those that contribute to the entailment of the defect. These axioms are precisely those that occur in the justifications. However, it is not always necessary to consider all of them. For instance, if $\mathfrak{J}$ contains a single justification, removing any one of its axioms is sufficient to obtain a repair. Therefore, we aim to minimize the number of questions posed to the user while ensuring that all justifications are hit. However, this is not trivial. For now, assume that the $sortMethod$ allows us to compute a *sorted set* $\mathcal{M}$ of all axioms occurring in the justifications, ordered such that decisions about axioms that appear earlier increases the likelihood of reaching a repair sooner.

For every justification axiom $\alpha \in \mathcal{M}$, the user can decide to either keep or remove it. In cases where this decision is unclear, the user may instead select a combination of *decision-support features* $\mathcal{F}$ to assist them. The first feature provides information for each important entailment in $\mathcal{E}$. Specifically, it shows the *percentage* of repairs in which the entailment holds when α is retained, compared to the percentage of repairs in which it holds when α is removed. The purpose of this feature is to illustrate how the user's decision affects the preservation of important entailments. The second feature highlights the effect of the user's decision on the *class hierarchy* by showing how removing α changes those parts of the hierarchy whose entailments depend on α. The last feature informs the user about the best outcome they can achieve with respect to their decision. This is conveyed by showing how much the best repairs—one corresponding to retaining α and one to removing it—differ from the current TBox. Here, a best repair is defined as a repair that preserves the largest number of axioms from the original TBox, and this difference is quantified using a normalized *Hamming distance* over TBox axioms. Lastly,

if the user's decisions preserve too many axioms of $\mathcal{T}$, thereby preventing a repair from being reached, the algorithm returns the empty set.

In Algorithm 1 we compute the set $\mathfrak{D}$ of all *minimal diagnoses* [10, 23], from which we obtain the set $\mathfrak{R}$ of all *maximal repairs*. Although all diagnoses, and consequently all repairs, are computed, the user never inspects them explicitly. Instead, as we show in the following sections, information from these sets is used to assist the user in constructing repairs interactively.

2.2. Axioms Sorting

The goal of sorting the axioms occurring in the justifications is to minimize the number of questions the user needs to answer. In principle, defects can be any unwanted consequences, such as concept unsatisfiability. In such a case, the ordering of axioms could, for instance, be guided by *anti-patterns* [24], so that the user first makes decisions about the potentially faulty axioms. However, since we consider defects in a broader sense, we require a more general criterion. Specifically, we consider two ways of sorting: *frequency* and *entropy* sorting.

Both methods rely on the complete set of justifications, since the corresponding frequency or entropy scores of the axioms can only be computed once all justifications have been obtained. As the computation of all justifications can be expensive [13], waiting for the complete set before presenting any axioms to the user may significantly delay the start of the repair process. Therefore, we provide a *live* mode in which sorting is performed incrementally. Instead of waiting for all justifications to be computed, the system periodically takes a snapshot of the justifications computed so far and derives a temporary ordering of the axioms based on this partial information. This process is repeated at fixed intervals, continuously refining the ordering as additional justifications become available. Once all justifications have been computed, the final ordering is determined. The underlying intuition is that axioms appearing near the beginning of the ordering are expected to have a greater influence on the repair process. Consequently, presenting these axioms to the user first helps navigate the space of diagnoses and repairs more efficiently, potentially reducing the number of decisions required to reach a repair.

2.2.1. Frequency Sorting

If an axiom appears in m justifications, then removing it breaks m justifications [4]. This means that prioritizing such axioms can reduce the number of user decisions required compared to first considering axioms that occur in only one or two justifications. Computing the frequency is inexpensive, since justifications must be computed in any case. However, there is no guarantee that such a strategy will always yield a decision-minimizing ordering.

2.2.2. Entropy Sorting

Instead of considering axioms through justifications, another perspective is to group them into diagnoses. Recall that a diagnosis $\mathcal{D} \in \mathfrak{D}$ of δ w.r.t. $\mathcal{T}$ is a set of axioms such that $\mathcal{T} \setminus \mathcal{D} \not\models \delta$. Therefore, prioritizing axioms that occur in roughly half of the diagnoses is beneficial, since a decision about such axioms tends to partition the diagnoses more evenly than a decision about axioms that occur either very frequently or very rarely. As diagnoses are filtered out, the corresponding set of possible repairs is reduced, allowing a repair to be reached more quickly. In our approach, we achieve this behavior by sorting axioms according to their *entropy score*. A smaller score for an axiom α_1 indicates that its occurrence is more evenly distributed across the diagnoses in $\mathfrak{D}$ than the occurrence of an axiom α_2 with a higher score. Consequently, a decision about α_1 partitions the diagnoses (and repairs) search space more evenly than a decision about α_2. Similar to the entropy score function in [19], we define:

$$\text{entropy}_v(\alpha) = \sum_{v \in \{\mathsf{In},\mathsf{Out}\}} [\text{prob}_v(\alpha) * \log_2(\text{prob}_v(\alpha))] + 1$$

where, $\mathrm{prob}_{\mathsf{In}}(\alpha)$ denotes the probability that α occurs in a diagnosis in $\mathfrak{D}$, and similarly $\mathrm{prob}_{\mathsf{Out}}(\alpha)$. Formally, $\mathrm{prob}_v(\alpha)$ is defined as:

$$\mathrm{prob}_v(\alpha) = \frac{|\mathfrak{D}^v|}{|\mathfrak{D}|} \quad \text{where} \quad \mathfrak{D}^v = \begin{cases} \{\mathcal{D} \in \mathfrak{D} \mid \alpha \in \mathcal{D}\} & \text{if } v = \mathsf{In} \\ \{\mathcal{D} \in \mathfrak{D} \mid \alpha \notin \mathcal{D}\} & \text{if } v = \mathsf{Out} \end{cases}$$

Unlike frequency sorting, this method is more costly, as the set of all minimal diagnoses must first be computed. However, it has been shown in [19] that using the entropy score helps reducing the number of questions the user needs to answer.

2.3. Decision-Support features

As mentioned earlier, the user may not always be able to make a definitive decision about an axiom α based solely on its semantics. In such cases, a mechanism for providing decision support is required. In this section, we describe the three decision-support features we consider.

2.3.1. Entailment Preservation Rate

When repairing a TBox $\mathcal{T}$, the user can provide an additional set $\mathcal{E}$ containing important entailments. Note that these entailments are not necessarily part of $\mathcal{T}$, but should be consequences of it. The purpose of this is to allow the user to incorporate domain knowledge into the repair process, which can then be used to provide additional insights. Specifically, whenever the user is unsure whether to keep or remove an axiom α, they can request the important *entailments preservation rate*. For each $\eta \in \mathcal{E}$, this feature provides two values: the percentage of reachable repairs in which η remains entailed when α is retained, and when α is removed. This feature allows the user to better understand how their decision affects the set of consequences they consider important.

2.3.2. Hierarchy Difference

When repairing $\mathcal{T}$, changes to its structure are inevitable. For instance, removing a subsumption axiom between concept names may alter the class hierarchy by modifying the set of *direct* super-concept relations between concept names, causing some direct relations to disappear while previously indirect ones become direct as a consequence. Therefore, the purpose of this feature is to leverage these changes to provide additional information about the role of α in the hierarchical structure of $\mathcal{T}$. When the user is uncertain whether to keep or remove α, they are presented with two DAGs representing the class hierarchies of two possibilities: $\mathcal{T}_{\mathsf{In}} = \mathcal{T} \setminus toRemove$ and $\mathcal{T}_{\mathsf{Out}} = \mathcal{T} \setminus (toRemove \cup \{\alpha\})$, where vertices represent concept names and edges represent subsumption relations between them. The differences between the two structures are highlighted as follows: If an edge appears in the graph of $\mathcal{T}_{\mathsf{In}}$ but not in the graph of $\mathcal{T}_{\mathsf{Out}}$, it is marked as a removed edge. Conversely, if an edge appears in the graph of $\mathcal{T}_{\mathsf{Out}}$ but not in that of $\mathcal{T}_{\mathsf{In}}$, it is marked as an added edge. In essence, this feature allows the user to localize the effect of their decision on the structure of $\mathcal{T}$.

2.3.3. Dissimilarity Measure

Although a repair that alters the original TBox $\mathcal{T}$ as little as possible may still not be the correct repair for the user, it is nevertheless desirable to obtain a repair that does not differ significantly from $\mathcal{T}$. The goal of this feature is to provide the user with additional information that helps them compare the effect of their decision (whether to keep or remove α) based on the best reachable repairs in each case, where the best repairs are those most similar to $\mathcal{T}$. This is achieved using the Hamming distance, which quantifies the dissimilarity between TBoxes based on their axioms. We denote by $\mathcal{T}_c$ the current TBox obtained by applying the user's decisions so far to $\mathcal{T}$. The user is provided with two distance values: one between the current state TBox $\mathcal{T}_c$ and one of the best repairs reachable when α is retained, and the other between $\mathcal{T}_c$ and one of the best repairs reachable when α is removed. Note that there may

be multiple reachable repairs in each case that remove the same minimal number of axioms from $\mathcal{T}$. Therefore, we randomly select one of them as a reference for computing the distance. We lift the similarity measure between concepts defined in [25, 26] to the level of axioms, since our repair approach considers only the removal of axioms rather than their modification. The Hamming distance between two TBoxes $\mathcal{T}_c$ and $\mathcal{T}'$ is defined based on this similarity measure as follows:

$$\text{hamming}(\mathcal{T}_c, \mathcal{T}') = 1 - \frac{|\mathcal{T}_c \cap \mathcal{T}'|}{|\mathcal{T}_c \cup \mathcal{T}'|}$$

where the distance ranges between 0 (no changes at all) to 1 (no similarities at all).

Lastly, regardless of which combination of decision-support features the user chooses, they are always notified whenever retaining α prevents a repair from being reached.

3. Implementation

Our implementation of the interactive repair approach with decision support consists of two main components: a text-based command-line tool and a web-based prototype that extends the command-line tool with additional visual features. Algorithm 1 provides only a high-level description of the approach, intentionally abstracting from the implementation details to simplify the presentation of the main idea. In this chapter, we describe the concrete implementation in more detail.

3.1. Command-line tool

We implement our repair approach as an extension of ELExplicator [2], which is primarily a command-line tool for explaining consequences and non-consequences of $\mathcal{EL}$ ontologies. Our implementation is mainly written in Java, using the OWL API [27] and the reasoner Elk. In addition, we use Python and a modified version of Inca [28]—a tool for navigating stable models of Answer Set Programs (ASP) using the solver Clingo [29]—to compute the diagnoses.

The input consists of the following: two OWL files—one containing the defective ontology and another containing the set of important entailments—the defect axiom in *Manchester OWL Syntax*, and the sorting method. As the computation of all justifications, and consequently all diagnoses, can take a long time, the tool provides an option to parallelize justification computation, sorting, and the display of axioms, which can be enabled using the flag `-live`. By default, the entire process is performed synchronously, corresponding to the flag `-full`. During the repair process, the user can access the decision-support features by entering the keyword *not sure*.

3.1.1. Computing justifications and minimal diagnoses

The computation of all justifications of the given defect is performed using Elk and the OWL API. Then, using Inca, we compute the set of all minimal diagnoses, which in turn yields all maximal repairs and enables the decision-support features provided to the user. Since ASP is not the main topic of this paper, we provide only an intuition of how Inca is used to compute the diagnoses. For further details, we refer the reader to [28].

Answer Set Programming is a declarative rule-based programming paradigm that allows problem specifications to be described by means of constraints in logic programs [30]. In order to use Inca, we translate certain aspects of the repair process into a logic program whose solutions, called *answer sets* or *stable models*, represent diagnoses. For example, let δ be a defect and $\mathfrak{J}$ be the set of all its justifications. Let N be the number of distinct axioms occurring in $\mathfrak{J}$. We assign a unique identifier α_j, where $1 \leq j \leq N$, to each such axiom and maintain the mapping between these identifiers and the original ontology axioms throughout the repair process. An atom α_j in an answer set indicates that the

[2]The source code and instructions for using the tool are available at https://github.com/de-tu-dresden-inf-lat/elexplicator.

corresponding axiom is retained. For every $\mathcal{J} \in \mathfrak{J}$, where $\mathcal{J} = \{\alpha_{j_1}, \alpha_{j_2}, \ldots, \alpha_{j_n}\}$, we create a *rule* of the form

$$\delta \text{ :- } \alpha_{j_1},\ \alpha_{j_2},\ \ldots,\ \alpha_{j_n}.$$

These rules encode all the ways in which the defect can be inferred. Additionally, the following rules are added to the logic program:

1. For every axiom identifier α_j, the rule

   $$\text{remove}(j) \text{ :- not } \alpha_j.$$

   explicitly records that the corresponding axiom is removed.

2. The *integrity constraint*

   $$\text{:- } \delta.$$

   prevents answer sets in which the defect is entailed.

To enumerate diagnoses in increasing order of cardinality, INCA iteratively modifies and solves the logic program while constraining the number of removed axioms. Specifically, at iteration i, the following constraint is temporarily added:

$$\text{:- not \#count}\{X : \text{remove}(X)\} \ = \ i.$$

This restricts the search to answer sets containing exactly i atoms of the form $\text{remove}(X)$. Since each such atom corresponds to one removed justification axiom, these answer sets represent diagnoses of size i. Initially, i is set to 1 and is subsequently increased up to N.

Whenever a diagnosis represented by $\{\text{remove}(k_1), \text{remove}(k_2), \ldots, \text{remove}(k_m)\}$ is found, the following integrity constraint is added to the program:

$$\text{:- not } \alpha_{k_1},\ \text{not } \alpha_{k_2}, \ldots, \text{not } \alpha_{k_m}.$$

This constraint eliminates the discovered diagnosis and all its supersets from subsequent iterations by preventing any answer set in which all axioms belonging to the discovered diagnosis are removed. Since diagnoses are considered in increasing order of cardinality, every diagnosis found at iteration i is minimal. Indeed, any smaller diagnosis would already have been discovered, and all its supersets would therefore have been eliminated. After all diagnoses of size i have been computed and added to the program as constraints, INCA checks whether any diagnoses of greater cardinality remain. To this end, it solves the program without the cardinality constraint while retaining all constraints corresponding to previously discovered diagnoses. Since all diagnoses of cardinality at most i have already been considered, the *unsatisfiability* of this program indicates that no additional diagnoses remain and that all minimal diagnoses have been enumerated. If the program is *satisfiable*, then the cardinality bound is increased to $i + 1$, and the process is repeated.

Once the process terminates, the *remove* atoms contained in the answer sets are translated back into diagnoses over the original justification axioms using the mapping established earlier. These diagnoses are then used to derive the corresponding maximal repairs, which underpin the decision-support features of the interactive repair process.

3.1.2. Axiom Sorting, Display and User Decisions

The justification axioms are sorted before they are displayed to the user in order to improve the efficiency of the repair process. We use a hash map to store the score of each justification axiom, where the score corresponds to either its frequency or its entropy, depending on the selected sorting method. If the repair process is initiated using the `-full` flag, the system waits for all the justifications to be computed before sorting the axioms. In contrast, if the `-live` flag is enabled the system periodically checks for newly computed justifications and updates the map accordingly. Then it creates a temporary ordered

list of axioms to be shown to the user. In the *live* mode, the axioms are displayed to the user as the sorted list is populated, whereas in the *full* mode, they are displayed only after the complete set of justification axioms has been fully sorted.

For each justification axiom, the user is asked whether it should be kept or removed, and must answer Yes or No to proceed with the repair process. Two sets of axioms, $toKeep$ and $toRemove$, are used to record the user's decisions throughout the process. To reflect these decisions in the computation of minimal diagnoses, the following constraints are added:

- `:-` not α., for every $\alpha \in toKeep$, ensuring that the diagnoses do not contain these axioms and they are therefore preserved in the repair.
- `:-` α., for every $\alpha \in toRemove$, enforcing the removal of these axioms by requiring them to occur in every computed diagnosis.

3.1.3. Repair-Reached Check

After all minimal diagnoses have been computed, the system checks after each user decision whether the decisions made so far can still lead to a repair. Instead of actually computing the set of all repairs, it is sufficient to check whether there exists a diagnosis that is a subset of $toRemove$. If this is the case, then a repair has been reached and the user is notified accordingly. They are then given the option to save the repair or to continue answering the remaining justification axioms. If the user chooses to continue, the repair-reached check is no longer performed for the remaining axioms.

3.1.4. Saving the Repair Ontology

The user can choose to save the current state of the ontology at any point during the repair process, either after all justification axioms have been answered or at an intermediate stage, using the *save* command. The resulting ontology is then saved as an OWL/XML file in an output directory with a file name specified by the user. Different scenarios, and consequently different program flows, may arise when the user invokes the *save* command, depending on whether the current ontology is already a repair.

When saving is initiated by the user, the user's selections are applied to create a new ontology by retaining only those axioms from the original ontology that have not been chosen for removal, i.e., those not in the $toRemove$ set. The system then checks whether this modified ontology is a repair by testing whether the defect axiom is still entailed. Similar to the *repair-reached check*, this is achieved by verifying whether the $toRemove$ list is a superset of any of the minimal diagnoses. Afterwards, depending on the outcome, the saving process proceeds accordingly.

If the modified ontology still entails the defect axiom, it is not a repair, and the user is notified accordingly. They can then choose either to continue saving it or cancel the save process using the commands *continue* or *cancel*, respectively. Canceling the process returns the user to the state before the save process was initiated. If the modified ontology no longer entails the defect axiom, it is a repair. However, it is possible that the user has chosen to remove more axioms than necessary, resulting in a non-maximal repair. Since we prioritize minimal information loss, the system checks whether the repair is maximal and gives the user the option either to save a feasible maximal repair or to save the repair as it is.

In order to check the maximality of the repair, the system compares the set of removed axioms with the set of minimal diagnoses computed from the original ontology. If the $toRemove$ set matches one of the minimal diagnoses, the current repair is already maximal. Otherwise, all the minimal diagnoses that are fully contained in the $toRemove$ set become candidates for refining the repair further into a maximal one. The user can then use the *max* command to refine the repair or *continue* to save the repair as it is. If multiple candidate minimal diagnoses exist, the user is presented with all of them and can select one to compute and save the corresponding maximal repair. If only one such candidate exists, the maximal repair is computed and saved automatically.

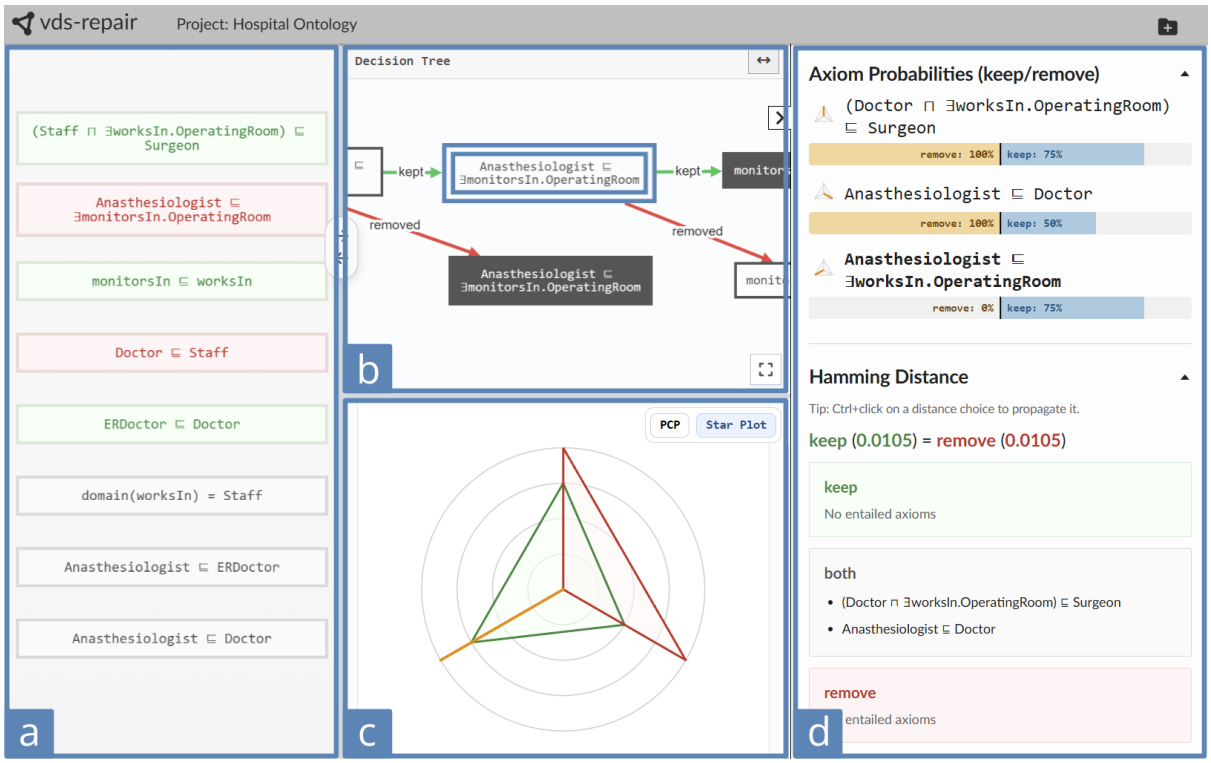

Figure 1: Screenshot of our prototype for visual decision support in interactive repair. The *process view* (a) shows the axioms for which the user must make a decision (green = **kept**, red = **removed**, gray = **undecided**). The *decision tree* (b) explicitly shows the possible repair paths depending on the binary decisions for each axiom. The *starplot* (c) compares the preservation rates of important entailments, supported by the interactive panel (d), where the least dissimilar minimal-change repairs (measured by Hamming distance) can also be propagated to the other views.

3.2. Interactive Prototype for Visual Decision Support

In order to provide *visual decision support*, we designed and developed an interactive prototype that displays the information illustrated thus far along the repair process in tailored views (see Figure 1). The axioms for which the user must make a decision are listed in a *process view*, while the decision-support features are spread along the rest of its views. For instance, consider the example of a *Hospital Management System* where the defect is $\mathtt{Anasthesiologist} \sqsubseteq \mathtt{Surgeon}$, and the set of the important entailments are:

(IA1) $\mathtt{Anasthesiologist} \sqsubseteq \mathtt{Doctor}$

(IA2) $\mathtt{Doctor} \sqcap \exists \mathtt{worksIn}.\mathtt{OperatingRoom} \sqsubseteq \mathtt{Surgeon}$

(IA3) $\mathtt{Anasthesiologist} \sqsubseteq \exists \mathtt{worksIn}.\mathtt{OperatingRoom}$

This is the example illustrated in Figure 1. The *process view* provides a compact representation of the repair by listing the axioms to decide upon and encoding whether they have been **kept**, **removed**, or are **still undecided**. However, the user may be interested in alternative paths within the full *decision tree* related to the repair. Considering the exponential size of the decision trees (depending on the number of axioms to decide upon) we employ degree-of-interest trees [31], a focus+context technique that reveals only paths of interest to the user, on demand. The *decision tree* and *process view* are equivalent. That is, every path of the decision tree can be shown in the process view, and every state of the process view corresponds to a set of paths in the decision tree (all axioms undecided in the process view correspond to the entire decision tree). Therefore, via the context menu, we allow transferring a path of the decision tree to the process view, or to open at once all paths of the decision tree that correspond to the state of the process view.

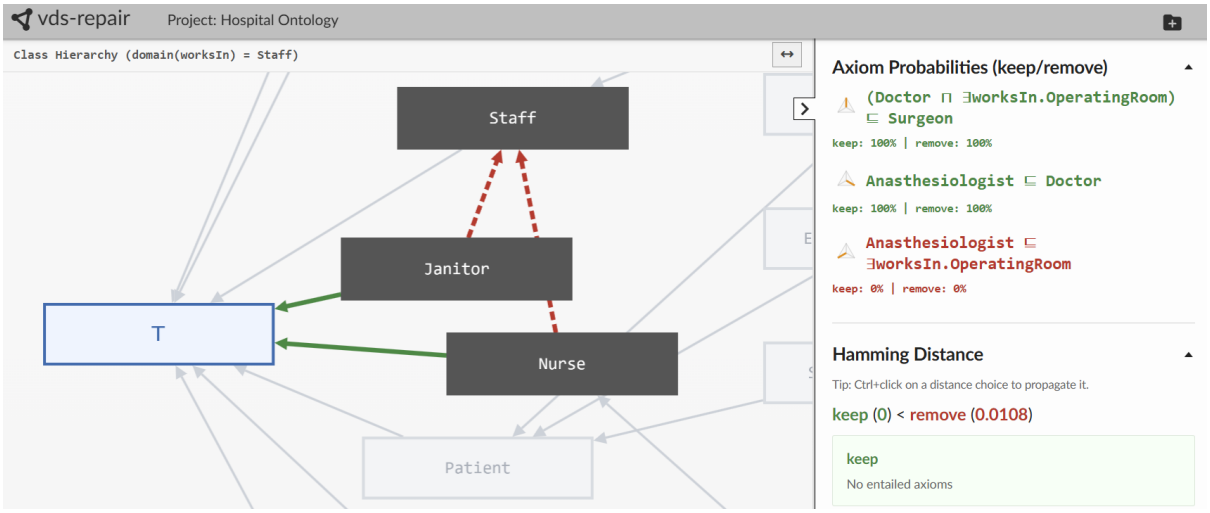

Figure 2: Screenshot of class hierarchy *diff view* in our interactive prototype. The diff shows the result of removing `domain(worksIn) = Staff` from the ontology. **Removed relations** are denoted with red and **new ones** with green. Note how the right-hand-side panel on Hamming distance shows that keeping this axiom at this point corresponds to a minimal-change repair (distance 0).

For each node in the decision tree, we provide a comparative view that employs starplots (superimposed or juxtaposed [32]) or parallel coordinates plots [33]. In both these techniques, axes correspond to the rates of preserving the important entailments. Furthermore, more than one axiom from the decision tree can be loaded into these views, for deeper analysis of their impact towards the important entailments at different points of the repair process. This is exemplified in Figure 3, where the view has been set to use juxtaposed (i.e., side-by-side) pairs of starplots. In our example, consider the decision for the axiom $\mathtt{Anasthesiologist} \sqsubseteq \exists \mathtt{monitorsIn}.\mathtt{OperatingRoom}$. If **removed**, the rate of entailing **(IA3)** in the resulting repair becomes 0. If **kept**, the chances of entailing **(IA3)** are higher (see corresponding triangles in Figure 1c or the top-right pair of Figure 3). This information is also laid out in the right-hand-side panel with comparative bars (Figure 1d). In this panel, the user is also provided the *hamming distances* to the closest minimal-change repairs, which can be temporarily inspected (on hover) or propagated to the other views (ctrl+click). Lastly, we provide a simple diff view of the class hierarchy when an axiom is removed. In Figure 2, the selected axiom for this view is $\mathtt{domain(worksIn)} = \mathtt{Staff}$, which results in two subsumption relations changing from $\mathtt{Staff}$ to $\top$.

Our work is free and open source, hosted at github.com/imldresden/vds-repair/[3]. The prototype was implemented as a web-application using *Cytoscape.js* [34] and *D3.js* [35]. The complete integration with the command-line tool is not yet finished, though the file format is already matched between applications. For reproducibility, the GitHub tag `DL26` corresponds to the version of the tool and the example discussed in this paper.

4. Conclusion

Revising and repairing ontologies is an essential part of the ontology development life cycle. In this work, we have presented a novel approach for interactive ontology repair with decision-support features that not only helps users navigate the large space of possible repairs, but also assists them in making decisions when they are uncertain. Our approach addresses several key aspects of ontology repair: (1) it provides methods for reducing the number of questions the user needs to answer through different sorting strategies; (2) it introduces three decision-support criteria—entailment preservation rate, hierarchy difference, and dissimilarity measure—together with a concrete instantiation of each; and (3) we present two tools implementing the approach: a command-line system and a web-based prototype that offers

[3]The latest version of our tool is live at imldresden.github.io/vds-repair.

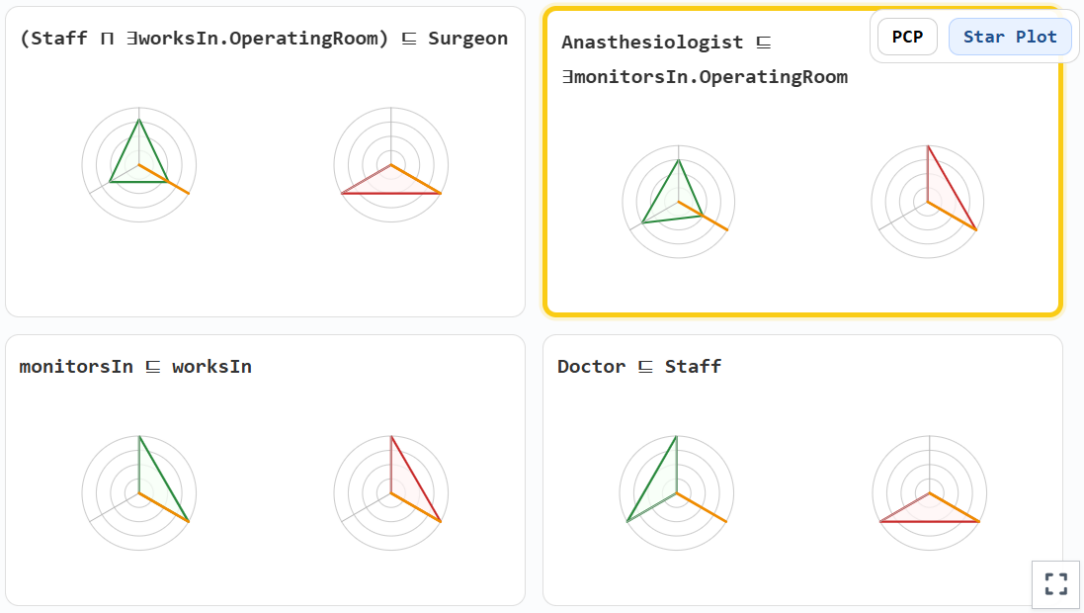

Figure 3: Screenshot of the side-by-side starplots view in our prototype. Hovering over a pair of starplots higlights the corresponding axiom in the decision tree in yellow. Likewise, by hovering over an axis of the starplot, the important entailment is highlighted in all starplots (here **(IA1)**). **(IA2)** is the top axis, and **(IA3)** the left one. On the bottom left pair, we see that keeping and removing the axiom affects the important entailments equally. On the top right pair, we see that removing this axiom means **(IA3)** will be lost. On the bottom right, we see that keeping $\texttt{Doctor} \sqsubseteq \texttt{Staff}$ leads to the same fate for **(IA3)**.

additional analytical features, allowing users to compare multiple decisions at different stages of the repair process.

A quantitative evaluation of the entire approach is a necessary next step. As future work, we plan to conduct multiple experiments to benchmark the performance of our tools, and evaluate the different decision-support features through a comparative analysis. We hope that this will provide insights into which repair scenarios benefit most from which decision-support features. Additionally, we plan to conduct a user study to evaluate the design of our interactive visual solution, as well as its usability and usefulness. Lastly, we will continue improving our tools and integrate this repair approach into Evonne [36], our tool for explaining DL reasoning and debugging ontologies.

Acknowledgments

This work is funded by Deutsche Forschungsgemeinschaft (DFG) under Germany's Excellence Strategy: EXC 2050/2, 390696704 – "Centre for Tactile Internet with Human-in-the-Loop" (CeTI); by DFG grant 389792660 as part of TRR 248 – CPEC, see https://perspicuous-computing.science; and by Bundesministerium für Bildung und Forschung (BMBF) and Saxon State Ministry for Science, Culture and Tourism (SMWK) in Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI, SCADS22B).

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT 5.3 in order to: Grammar and spelling check. After using this tool the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] B. Suntisrivaraporn, F. Baader, S. Schulz, K. A. Spackman, Replacing SEP-triplets in SNOMED CT using tractable description logic operators, in: Artificial Intelligence in Medicine, 11th Conference on Artificial Intelligence in Medicine, AIME 2007, Amsterdam, The Netherlands, July 7-11, 2007, Proceedings, Lecture Notes in Computer Science, Springer, 2007, pp. 287–291. doi:10.1007/978-3-540-73599-1_38.

[2] F. Baader, B. Suntisrivaraporn, Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$, in: Proceedings of the Third International Conference on Knowledge Representation in Medicine, Phoenix, Arizona, USA, May 31st - June 2nd, 2008, CEUR Workshop Proceedings, CEUR-WS.org, 2008. URL: https://ceur-ws.org/Vol-410/Paper01.pdf.

[3] S. Schlobach, R. Cornet, Non-standard reasoning services for the debugging of description logic terminologies, in: IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003, Morgan Kaufmann, 2003, pp. 355–362. URL: http://ijcai.org/Proceedings/03/Papers/053.pdf.

[4] A. Kalyanpur, B. Parsia, E. Sirin, B. C. Grau, Repairing unsatisfiable concepts in OWL ontologies, in: Y. Sure, J. Domingue (Eds.), 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, Springer, 2006, pp. 170–184. doi:10.1007/11762256_15.

[5] S. Schlobach, Z. Huang, R. Cornet, F. van Harmelen, Debugging incoherent terminologies, J. Autom. Reason. 39 (2007) 317–349. doi:10.1007/S10817-007-9076-Z.

[6] J. S. C. Lam, D. H. Sleeman, J. Z. Pan, W. W. Vasconcelos, A fine-grained approach to resolving unsatisfiable ontologies, Journal on Data Semantics 10 (2008) 62–95. doi:10.1007/978-3-540-77688-8_3.

[7] F. Baader, F. Kriegel, A. Nuradiansyah, R. Peñaloza, 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, AAAI Press, 2018, pp. 319–328. URL: https://aaai.org/papers/36-18056-making-repairs-in-description-logics-more-gentle/.

[8] F. Baader, P. Koopmann, F. Kriegel, Optimal repairs in the description logic $\mathcal{EL}$ revisited, in: Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings, Lecture Notes in Computer Science, Springer, 2023, pp. 11–34. doi:10.1007/978-3-031-43619-2_2.

[9] C. Alrabbaa, F. Baader, R. Dachselt, T. Flemisch, P. Koopmann, Visualising proofs and the modular structure of ontologies to support ontology repair, in: Proceedings of the 33rd International Workshop on Description Logics (DL 2020) co-located with the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020), Online Event [Rhodes, Greece], September 12th to 14th, 2020, CEUR Workshop Proceedings, CEUR-WS.org, 2020. URL: https://ceur-ws.org/Vol-2663/paper-2.pdf.

[10] R. Reiter, A theory of diagnosis from first principles, Artif. Intell. 32 (1987) 57–95. doi:10.1016/0004-3702(87)90062-2.

[11] M. Horridge, B. Parsia, U. Sattler, 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, Springer, 2008, pp. 323–338. doi:10.1007/978-3-540-88564-1_21.

[12] A. Kalyanpur, B. Parsia, M. Horridge, E. Sirin, 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, Springer, 2007, pp. 267–280. doi:10.1007/978-3-540-76298-0_20.

[13] F. Baader, R. Peñaloza, B. Suntisrivaraporn, Pinpointing in the description logic $\mathcal{EL}^+$, in: KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI, KI 2007, Proceedings, Lecture Notes in Computer Science, Springer, 2007, pp. 52–67. doi:10.1007/

978-3-540-74565-5_7.
[14] N. Nikitina, S. Rudolph, B. Glimm, Interactive ontology revision, J. Web Semant. 12 (2012) 118–130. doi:10.1016/J.WEBSEM.2011.12.002.
[15] Y. Li, P. Lambrix, A system for repairing $\mathcal{EL}$ ontologies using weakening and completing, in: The Semantic Web: ESWC 2023 Satellite Events - Hersonissos, Crete, Greece, May 28 - June 1, 2023, Proceedings, Lecture Notes in Computer Science, Springer, 2023, pp. 101–105. doi:10.1007/978-3-031-43458-7_19.
[16] T. Zendron, R. Peñaloza, Guiding interactive ontology repair through prolific and relevant axioms, in: Proceedings of the Workshop on Foundations and Future of Change in Artificial Intelligence (FCAI 2025), volume 4069, CEUR-WS, 2025, pp. 8–26. URL: https://ceur-ws.org/Vol-4069/paper2.pdf.
[17] F. Baader, F. Kriegel, Pushing optimal ABox repair from $\mathcal{EL}$ towards more expressive horn-dls, in: Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel, July 31 - August 5, 2022, 2022. doi:10.24963/kr.2022/3.
[18] F. Kriegel, Beyond optimal: Interactive identification of better-than-optimal repairs, in: Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing, SAC 2025, ACM, 2025, pp. 1019–1026. doi:10.1145/3672608.3707750.
[19] K. Shchekotykhin, G. Friedrich, P. Fleiss, P. Rodler, Interactive ontology debugging: Two query strategies for efficient fault localization, Journal of Web Semantics 12 (2012) 88–103. doi:10.1016/j.websem.2011.12.006.
[20] F. Baader, B. Ganter, B. Sertkaya, U. Sattler, Completing description logic knowledge bases using formal concept analysis, in: IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, 2007, pp. 230–235. URL: http://ijcai.org/Proceedings/07/Papers/035.pdf.
[21] F. Baader, B. Sertkaya, Usability issues in description logic knowledge base completion, in: Formal Concept Analysis, 7th International Conference, ICFCA 2009, Proceedings, Lecture Notes in Computer Science, Springer, 2009, pp. 1–21. doi:10.1007/978-3-642-01815-2_1.
[22] Y. Kazakov, M. Krötzsch, F. Simancík, 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.
[23] K. Moodley, T. Meyer, I. J. Varzinczak, Root justifications for ontology repair, in: Web Reasoning and Rule Systems - 5th International Conference, RR 2011, Galway, Ireland, August 29-30, 2011. Proceedings, Lecture Notes in Computer Science, Springer, 2011, pp. 275–280. doi:10.1007/978-3-642-23580-1_24.
[24] O. Corcho, C. Roussey, L. M. Vilches Blazquez, Catalogue of Anti-Patterns for formal Ontology debugging, in: Conférence francophone sur l'apprentissage automatique (AFIA 2009)Atelier Construction d ontologies : vers un guide des bonnes pratiques,, Hammamet, Tunisia, 2009, p. 11. URL: https://hal.science/hal-01437682.
[25] J. David, J. Euzenat, Comparison between ontology distances (preliminary results), 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, Springer, 2008, pp. 245–260. doi:10.1007/978-3-540-88564-1_16.
[26] A. Maedche, S. Staab, Measuring similarity between ontologies, in: International Conference on Knowledge Engineering and Knowledge Management, Springer, 2002, pp. 251–263.
[27] 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.
[28] C. Alrabbaa, S. Rudolph, L. Schweizer, 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, Springer, 2018, pp. 211–225. doi:10.1007/978-3-319-99906-7_14.
[29] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Multi-shot ASP solving with clingo, Theory Pract. Log. Program. 19 (2019) 27–82. doi:10.1017/S1471068418000054.

[30] T. Eiter, G. Ianni, T. Krennwallner, Answer set programming: A primer, in: Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures, Lecture Notes in Computer Science, Springer, 2009, pp. 40–110. doi:10.1007/978-3-642-03754-2_2.

[31] S. K. Card, D. Nation, Degree-of-interest trees: a component of an attention-reactive user interface, in: Proceedings of the Working Conference on Advanced Visual Interfaces, AVI '02, Association for Computing Machinery, 2002, p. 231–245. doi:10.1145/1556262.1556300.

[32] W. Javed, N. Elmqvist, Exploring the design space of composite visualization, in: 2012 IEEE Pacific Visualization Symposium, 2012, pp. 1–8. doi:10.1109/PacificVis.2012.6183556.

[33] A. Inselberg, The plane with parallel coordinates, The Visual Computer 1 (1985) 69–91. doi:10.1007/BF01901341.

[34] M. Franz, C. T. Lopes, D. Fong, M. Kucera, M. Cheung, M. C. Siper, G. Huck, Y. Dong, O. Sumer, G. D. Bader, Cytoscape.js 2023 update: a graph theory library for visualization and analysis, Bioinformatics 39 (2023). doi:10/g92hft.

[35] M. Bostock, V. Ogievetsky, J. Heer, D3: Data-Driven Documents, IEEE Transactions on Visualization and Computer Graphics 17 (2011) 2301–2309. doi:10/b7bhhf.

[36] J. Méndez, C. Alrabbaa, P. Koopmann, R. Langner, F. Baader, R. Dachselt, Evonne: A visual tool for explaining reasoning with OWL ontologies and supporting interactive debugging, Comput. Graph. Forum 42 (2023). doi:10.1111/CGF.14730.