Union and Intersection of all Justifications[⋆]

Jieying Chen[1], Yue Ma[2], Rafael Peñaloza[3], and Hui Yang[2]

[1] SIRIUS, Department of Information, University of Oslo, Oslo, Norway
`jieyingc@ifi.uio.no`
[2] LISN, Univ. Paris-Sud, CNRS, Université Paris-Saclay, Orsay, France
`{ma, yang}@lri.fr`
[3] University of Milano-Bicocca, Milan, Italy
`rafael.penaloza@unimib.it`

Abstract. We present new algorithms for computing the union and intersection of all justifications for a given ontological consequence without first computing the set of all justifications. Our goal is to use these sets to explain the consequences and, if needed, repair them. Through an empirical evaluation, we show that our approach behaves well in practice for expressive description logics. In particular, the union of all justifications can be computed much faster than with existing justification-enumeration approaches.

Keywords: Justifications, Axiom Pinpointing, Ontology Repairs.

1 Introduction

It is well known that ontology engineering is a delicate and error-prone task, which requires automated tools to avoid introducing unexpected or unwanted consequences. Indeed, as an ontology grows in size it becomes difficult to predict *a priori* what effect would introducing or modifying an axiom have over the represented notions. In these settings, it is not rare for a knowledge engineer to encounter unexpected consequences from the explicitly stated knowledge. In this case, the knowledge engineer should try to understand why is this a consequence, and perhaps how to get rid of it. To achieve this, it is helpful to focus exclusively on the axioms that are relevant for this consequence.

Axiom pinpointing [35] is the task of identifying the axioms in an ontology that are required for a consequence to follow. Primarily, its focus is on computing the class of all *justifications*: subset-minimal subontologies that entail the consequence. A dual notion is that of a *repair*: a subset-maximal subontology which does not entail the consequence. Justifications provide a way to understand the causes for a consequence, while repairs suggest a way to get rid of it. Axiom pinpointing methods can be separated into two main classes, commonly known as *black-box* and *glass-box*.

⋆ This work is partially funded by the Norwegian Research Council via the SIRIUS centre (Grant Nr.237898) and the BPI-France (PSPC AIDA: 2019-PSPC-09).

Black-box approaches [24, 25, 34] use existing reasoners as an oracle, and require no further modification of the reasoning method. Therefore, these approaches work for ontologies written in any monotonic logical language (including expressive DLs such as $\mathcal{SROIQ}$ [21]), as long as a reasoner supporting it exists. In their most naïve form, black-box methods check all possible subsets of the ontology for the desired entailment and compute the justifications from these results. In reality, many optimisations have been developed to reduce the number of calls needed, and avoid irrelevant work. Glass-box approaches, on the other hand, modify a reasoning algorithm to output one or all justifications directly, from only one call. While the theory for developing glass-box methods has been developed for different reasoners [6–9, 32], in practice not many of these methods have been implemented, as they require new coding efforts and (often) deactivating the optimisation techniques that make reasoners practical. A promising approach, first proposed in [42] is to reduce, through a reasoning simulation, the axiom pinpointing problem to a related enumeration problem from a propositional formula, and use state-of-the-art SAT-solving methods to enumerate all the justifications. This idea has led to effective axiom pinpointing systems developed primarily for the lightweight DL $\mathcal{EL}$ [4]; see [1–3, 26, 30, 31].

The interest of axiom pinpointing goes beyond enumerating justifications. When the scope is to get rid of an unwanted consequence, one may be interested in the repairs, or their complement, commonly called *diagnoses*. Diagnoses can be derived from justifications via a hitting set computation, and viceversa [8]. Moreover, as there might exist exponentially many justifications (or repairs, or diagnoses) for a given entailment w.r.t. an ontology, even for ontologies in an inexpressive language such as the DL $\mathcal{EL}$, finding all justifications is not feasible in general. To alleviate this issue, one may approximate the information through the union and the intersection of all justifications. Every element in the intersection is a diagnosis by itself. From the union, a knowledge engineer has a more precise view on the problematic instances, and can make a detailed analysis.

Although much work has focused on methods for computing one or all justifications efficiently, to the best of our knowledge there is little work on computing their intersection or union without enumerating them first, beyond the approximations presented in [36, 37]. Here, we first propose an algorithm for computing the intersection of all justifications. This algorithm has the same worst-case behaviour as the black-box algorithm of computing one justification, avoiding the worst-case exponential enumeration. Additionally, we present two approaches to compute the union of all justifications: one is based on the black-box algorithm for finding all justifications and the other approach uses the SAT-based tool *cmMUS*. An extended abstract of our paper was published on [16].

The paper is structured as follows. In Section 2 we recall relevant definitions of description logics and propositional logic. Section 3 presents the algorithm for computing the intersection of all justifications without computing any single justification, followed by two methods of computing the union of all justifications in Section 4. Afterwards, we explain how to use the union and intersection of all justifications to repair ontologies. Before concluding, an evaluation of our methods on real-world ontologies is presented in Section 6.

2 Preliminaries

We briefly recall the description logic $\mathcal{ALCH}$ [5] and the notions of justifications, repairs and ontology modules.

Let N_C and N_R disjoint sets of *concept-*, and *role names* respectively. The set of $\mathcal{ALC}$-concepts is built through the grammar rule

$$C ::= \top \mid \bot \mid A \mid C \sqcap C \mid C \sqcup C \mid \neg C \mid \exists r.C \mid \forall r.C,$$

where $A \in N_C$ and $r \in N_R$. An $\mathcal{ALCH}$ *TBox* $\mathcal{T}$ is a finite set of general concept inclusions (*GCIs*) of the form $C \sqsubseteq D$ and *role inclusions* $r \sqsubseteq s$, where C and D are $\mathcal{ALC}$ concepts and $r, s \in N_R$. From now on, we will call the TBox an *ontology*, and its elements (GCIs and role inclusions) will be called *axioms* in general. The DL $\mathcal{EL}$ is the restriction of $\mathcal{ALC}$ that does not allow for bottom $\bot$, negations $\neg$, nor value restrictions $\forall$.

The semantics of this logic is defined in terms of interpretations. An *interpretation* is a pair $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ where $\Delta^\mathcal{I}$ is a non-empty *domain*, and $\cdot^\mathcal{I}$ is the *interpretation function*, which maps each concept name $A \in N_C$ to a subset $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$, and each role name $r \in N_R$ to a binary relation $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. The interpretation function is extended to $\mathcal{ALC}$-concepts as usual: $(\top)^\mathcal{I} := \Delta^\mathcal{I}$, $(\bot)^\mathcal{I} := \emptyset$, $(\neg C)^\mathcal{I} := \Delta^\mathcal{I} \backslash C^\mathcal{I}$, $(C \sqcap D)^\mathcal{I} := C^\mathcal{I} \cap D^\mathcal{I}$, $(C \sqcup D)^\mathcal{I} := C^\mathcal{I} \cup D^\mathcal{I}$, $(\exists r.C)^\mathcal{I} := \{x \in \Delta^\mathcal{I} \mid \exists y \in C^\mathcal{I} : (x, y) \in r^\mathcal{I}\}$, and $(\forall r.C)^\mathcal{I} := \{x \in \Delta^\mathcal{I} \mid \forall y \in \Delta^\mathcal{I} : (x, y) \in r^\mathcal{I} \Rightarrow y \in C^\mathcal{I}\}$.

The interpretation $\mathcal{I}$ *satisfies* $C \sqsubseteq D$ iff $C^\mathcal{I} \subseteq D^\mathcal{I}$ and it *satisfies* $r \sqsubseteq s$ iff $r^\mathcal{I} \subseteq s^\mathcal{I}$. We write $\mathcal{I} \models \alpha$ if $\mathcal{I}$ satisfies the axiom α. The interpretation $\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ if $\mathcal{I}$ satisfies all axioms in $\mathcal{O}$. An axiom α *is entailed by* $\mathcal{O}$, denoted as $\mathcal{O} \models \alpha$, if $\mathcal{I} \models \alpha$ for all models $\mathcal{I}$ of $\mathcal{O}$. We use $|\mathcal{O}|$ to denote the size of $\mathcal{O}$, i.e., the number of axioms in $\mathcal{O}$. $\mathsf{sig}(\alpha)$ is a function that extracts a set of concept and role names that occur in the GCI α.

We are interested in the notions of *justification* and *repair*.

Definition 1 (Justification, repair). *Let $\mathcal{O}$ be an ontology and α a GCI.*

- *A* justification *for $\mathcal{O} \models \alpha$ is a subset $\mathcal{M} \subseteq \mathcal{O}$ such that $\mathcal{M} \models \alpha$ and for any $\mathcal{M}' \subsetneq \mathcal{M}$, $\mathcal{M}' \not\models \alpha$.* Just$(\mathcal{O}, \alpha)$ *denotes the set of all justifications of α w.r.t. $\mathcal{O}$.*
- *A* repair *for $\mathcal{O} \models \alpha$ is a subontology $\mathcal{R} \subseteq \mathcal{O}$ such that $\mathcal{R} \not\models \alpha$, but $\mathcal{O}' \models \alpha$ for any $\mathcal{R} \subsetneq \mathcal{O}' \subseteq \mathcal{O}$. We denote the set of all repairs as* Rep$(\mathcal{O}, \alpha)$.

Briefly, a justification is a minimal subset of an ontology that preserves the conclusion, while a repair is a maximal sub-ontology that does not entail the consequence.

In the context of error-tolerant reasoning, where the goal is to derive meaningful consequences while avoiding known errors from the ontology, three main entailment relations have been considered.

Definition 2 (Brave, cautious and IAR entailments). *Let α be a consequence of $\mathcal{O}$ and $\mathrm{Rep}(\mathcal{O}, \alpha)$ be the set of all repairs. A consequence β is:*

- bravely *entailed by $\mathcal{O}$ w.r.t. α if $\mathcal{R} \models \beta$ for some $\mathcal{R} \in \mathrm{Rep}(\mathcal{O}, \alpha)$;*
- cautiously *entailed by $\mathcal{O}$ w.r.t. α if $\mathcal{R} \models \beta$ for all $\mathcal{R} \in \mathrm{Rep}(\mathcal{O}, \alpha)$;*
- IAR *entailed by by $\mathcal{O}$ w.r.t. α if $\bigcap_{R \in \mathrm{Rep}(\mathcal{O},\alpha)} \mathcal{R} \models \beta$.*

In other words, a brave entailment is one which could still hold after repairing the ontology. Cautious entailment is a stronger notion, requiring that the consequence holds regardless of how the ontology is repaired; thus every cautious entailment is also a brave one, but the converse is not true. IAR entailments are those that follow from the *intersection* of all repairs. Importantly, note that the intersection of all repairs and the union of all justifications complement each other. Hence, studying one yields the results for the other.

When dealing with large ontologies, it is useful to consider only a subset of axioms, which preserves all information about the entailments under consideration. A *module* is a sub-ontology that preserves some syntactic or semantic properties w.r.t. a restricted signature Σ. In general, it is hard to compute minimal modules for expressive ontologies. Still, there exists the notion of a *syntactic locality module* which can be computed efficiently, even for very expressive DLs [40]. *Lean kernels* [27,37] for the DL $\mathcal{ALC}$, and *minimal deductive modules* for the DL $\mathcal{EL}$ [13,14] usually define a smaller module. An important property of those modules (syntactic locality modules, lean kernels and minimal deductive modules) is that they are *justification-preserving*; i.e., all the justifications for a given consequence are contained in them. Due to this property, when computing justifications, their union, and their intersection, we will first compute such a module. As it is better understood, and well defined for expressive DLs, in the following we consider computing only locality-based modules. However, it should be considered that any justification-preserving module would suffice.

We now consider a propositional language with a finite set of propositional *variables* $L = \{p_1, p_2, \cdots, p_n\}$. A *literal* is a variable p_i or its negation $\neg p_i$. A *clause* $l_1 \vee l_2 \vee \cdots \vee l_k$ is a disjunction of literals, denoted by ω [11]. A Boolean formula in conjunctive normal form (CNF) is a conjunction of clauses. A CNF formula ϕ is *satisfiable* iff there exists a *truth assignment* $\mu_L : L \to \{0, 1\}$ such that μ_L satisfies all clauses in ϕ. We can also consider a CNF formula as a set of clauses. A subformula $\phi' \subseteq \phi$ is a *minimally unsatisfiable subformula (MUS)* iff ϕ' is unsatisfiable, but for every $\phi'_1 \subsetneq \phi'$ is satisfiable. Note that the notion of a MUS corresponds to that of a justification where the ontology language is propositional logic and the consequence under consideration is unsatisfiability.

3 Computing the Intersection of all Justifications

We first study the problem of computing the intersection of all justifications, which we call the *core*. Algorithm 1 provides a method for finding this core.

The algorithm is inspired by the known black-box approach for finding justifications [9, 23]. Starting from a justification-preserving module $\mathcal{M}$ (in this case, the locality-based module, Line 2), we try to remove one axiom β (Line 4). If the removal of the axiom β removes the entailment (Line 5), then β belongs to all justifications (β is *sine qua non* required for the entailment within $\mathcal{M}$), and is thus added to the core $\mathcal{C}$ (Line 6). It can be shown that Algorithm 1 correctly computes the intersection of all justifications.

Theorem 3. *Let $\mathcal{O}$ be an ontology and α a GCI. Algorithm 1 computes the intersection of all justifications (core) of $\mathcal{O}$ w.r.t. α.*

Proof. Let $\mathcal{S}$ be the set computed by Algorithm 1, so $\mathcal{S} = \{\beta \mid \mathcal{O} \setminus \{\beta\} \not\models \alpha\}$ (Line 5-6). Additionally, let $\mathcal{C}$ be the core. First, we prove that $\mathcal{S} \subseteq \mathcal{C}$. We assume towards a contradiction that there exists an axiom β such that $\beta \in \mathcal{S}$, but $\beta \notin \mathcal{C}$. As $\beta \notin \mathcal{C}$, there exists a justification $\mathcal{J}$ of $\mathcal{O}$ w.r.t. α such that $\beta \notin \mathcal{J}$. According to Definition 1, $\mathcal{O} \setminus \{\beta\} \models \alpha$, which contradicts to our assumption that $\beta \in \mathcal{S}$. The other direction $\mathcal{C} \subseteq \mathcal{S}$ is analogous. Therefore, $\mathcal{C} = \mathcal{S}$, that is, Algorithm 1 computes the core of all justifications of $\mathcal{O}$ w.r.t. α.

Algorithm 2, on the other hand, generalises the known algorithm for computing a single justification, by considering a (fixed) set $\mathcal{C}$ that is known to be contained in all justifications. If $\mathcal{C} = \emptyset$, the approach works as usual; otherwise, the algorithm avoids trying to remove any axiom from $\mathcal{C}$. This reduces the number of calls to the black-box reasoner, potentially decreasing the overall execution time.

The choice for a locality-based module in these algorithms is arbitrary, and any justification-preserving module would suffice. In particular, we could compute lean kernel [27, 37] for $\mathcal{ALC}$ ontologies, and minimal subsumption modules [12, 15] for $\mathcal{EL}$ ontologies instead, which is typically smaller thus reducing the number of iterations within the algorithms. However, as it could be quite expensive to compute such modules, it might not be worthwhile in some cases.

Algorithm 1, like all black-box methods for computing justifications, calls a standard reasoner $|\mathcal{M}|$ times. In terms of computational complexity, computing the core requires as many computational resources as computing a single justification. However, computing one justification might be faster in practice, as the size of $\mathcal{M}$ decreases throughout the execution of Algorithm 2. Clearly, if the core coincides with one justification $\mathcal{M}$, then $\mathcal{M}$ is the *only* justification.

Algorithm 1 Computing the intersection of all justifications of $\mathcal{O}$ w.r.t. α

```
INPUT: an Ontology $\mathcal{O}$, a conclusion $\alpha$
1: function COMPUTE-JUSTIFICATION-CORE($\mathcal{O}, \alpha$)
2:     $\mathcal{C} := \emptyset$
3:     $\mathcal{M}$ := COMPUTE-LOCALITY-BASED-MODULE($\mathcal{O}, \mathsf{sig}(\alpha)$)
4:     for every axiom $\beta \in \mathcal{M}$ do
5:         if $\mathcal{M} \setminus \{\beta\} \not\models \alpha$ then
6:             $\mathcal{C} := \mathcal{C} \cup \{\beta\}$
7:     return $\mathcal{C}$
```

Algorithm 2 Using core to compute a single justification of $\mathcal{O}$ *w.r.t.* an conclusion

```
INPUT: an ontology $\mathcal{O}$, a conclusion $\alpha$, the core $\mathcal{C}$
1: function SINGLE-JUSTIFICATION($\mathcal{O}, \alpha, \mathcal{C}$)
2:     $\mathcal{M} := $ COMPUTE-LOCALITY-BASED-MODULE($\mathcal{O}, \mathsf{sig}(\alpha)$)
3:     for every axiom $\beta \in \mathcal{M}$ and $\beta \notin \mathcal{C}$ do
4:         if $\mathcal{M} \setminus \{\beta\} \models \alpha$ then
5:             $\mathcal{M} := \mathcal{M} \setminus \{\beta\}$
6:     return $\mathcal{M}$
```

Corollary 4. *Let $\mathcal{O}$ be an ontology, α a GCI; and let $\mathcal{C}$ be the core and $\mathcal{J}$ a justification for $\mathcal{O} \models \alpha$. If $\mathcal{C} = \mathcal{J}$, $\mathcal{J}$ is the only justification for $\mathcal{O} \models \alpha$.*

4 Computing the Union of all Justifications

We now present two algorithms of computing the union of all justifications. The first algorithm follows a black-box approach that calls a standard reasoner as oracle using the core of justifications computed in the previous section. Importantly, it is known that no black-box method for computing the union of all justifications can call an oracle only a polynomial number of times, unless P = NP [38]. Our method is inspired by Reiter's Hitting Set Tree algorithm [39] and partially follows the approach originally developed in [23,45] for enumerating all justifications. For the second algorithm, we reduce the problem of computing the union of all justifications to the problem of computing the union of MUSes of a propositional formula. Note that the second algorithm works only for $\mathcal{ALCH}$ ontologies, while the first algorithm can be applied to ontologies with any expressivity, as long as a reasoner is available.

4.1 Black-box Algorithm

The black-box algorithm for computing all justifications from [45] was inspired by the algorithm of computing all minimal hitting sets [39]. Some of the improvements to prune the search space were already proposed in [39]. Our method for computing the union of all justifications (Algorithm 3) works in a similar manner, but with a few key differences. To avoid computing all justifications, we prune the search space when all remaining justifications are fully contained in the union computed so far. In addition, we use the core to speed the search. As the axioms in the core must appear in every justification, we can reduce the number of calls made to the reasoner, and optimise the single justification computation. Finally, when we organise our search space, we do not need to consider the axioms in the core.

We now explain Algorithm 3 in detail. Given an ontology $\mathcal{O}$, a conclusion α, and the core $\mathcal{C} \subseteq \mathcal{O}$ of $\mathcal{O}$ w.r.t. α as input, a justification-preserving module $\mathcal{M}$ of $\mathcal{O}$ w.r.t. α is extracted from $\mathcal{O}$ (Line 2). The justification search tree Ψ is a four-tuple $(\mathcal{V}, \mathcal{E}, \mathcal{L}, \rho)$, where $\mathcal{V}$ is a finite set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges, $\mathcal{L}$ is

Algorithm 3 Computing the Union of All Justifications w.r.t. a Conclusion α

```
INPUT: an Ontology $\mathcal{O}$, a conclusion $\alpha$, the intersection of all Justifications $\mathcal{C} \subseteq \mathcal{O}$
 1: function UNION-OF-ALL-JUSTIFICATIONS($\mathcal{O}, \alpha, \mathcal{C}$)
 2:     $\mathcal{M} := $ COMPUTE-LOCALITY-BASED-MODULE($\mathcal{O}, \mathsf{sig}(\alpha)$)
 3:     $\mathcal{U} := \mathcal{C}$; $\varPsi := (\{\rho\}, \emptyset, \emptyset, \rho)$; $\mathbb{Q} := [\rho]$; $\mathbb{P} := \emptyset$; $\mathbb{M} := \{\emptyset\}$
 4:     while $\mathbb{Q} \neq [\,]$ do
 5:         $v :=$ HEAD($\mathbb{Q}$), $\mathbb{Q} :=$ REMOVEFIRSTELEMENT($\mathbb{Q}$), $\mathbb{P} := \mathbb{P} \cup \{v\}$
 6:         $\mathcal{M}_{\text{ex}} :=$ LABELS(PATH($\varPsi, \rho, v$))
 7:         if IS-PATH-REDUNDANT($\varPsi, \rho, \mathcal{M}_{\text{ex}}, \mathbb{P}$) then
 8:             continue
 9:         if $\mathcal{M} \setminus \mathcal{M}_{\text{ex}} \not\models \alpha$ then
10:             continue
11:         if $\mathcal{M} \setminus \mathcal{M}_{\text{ex}} \subseteq \mathcal{U}$ then
12:             continue
13:         $\mathcal{M} := \emptyset$
14:         if there exists $\mathcal{M}' \in \mathbb{M}$ such that $\mathcal{M}_{\text{ex}} \cap \mathcal{M}' = \emptyset$ then
15:             $\mathcal{M} := \mathcal{M}'$
16:         else
17:             $\mathcal{M} :=$ SINGLE-JUSTIFICATION($\mathcal{M} \setminus \mathcal{M}_{\text{ex}}, \alpha, \mathcal{C}$)
18:             if $\mathcal{M} = \mathcal{C}$ then
19:                 return $\{\mathcal{C}\}$
20:             $\mathbb{M} := \mathbb{M} \cup \{\mathcal{M}\}$
21:             $\mathcal{U} := \mathcal{U} \cup \mathcal{M}$
22:         for every $\beta \in \mathcal{M} \setminus \mathcal{C}$ do
23:             $v_\beta :=$ ADDCHILD($\varPsi, v, \beta$)
24:             $\mathbb{Q} := v_\beta :: \mathbb{Q}$
25:     return $\mathcal{U}$
```

an edge labelling function, mapping every edge to an axiom $\alpha \in \mathcal{M}$, and $\rho \in \mathcal{V}$ is the root node. We initialise the variable $\varPsi$ to represent a justification search tree for $\mathcal{O}$ having only root node ρ. Besides, the variables $\mathbb{M} \subseteq 2^{\mathcal{M}}$, containing the justifications that have been computed so far, and $\mathbb{P} \subseteq \mathcal{V}$, containing the already explored nodes of $\varPsi$, are both initialised with the empty set. The queue $\mathbb{Q}$ of nodes in $\varPsi$ that still has to be explored is also set to contain the node ρ as its only element.

The algorithm then enters a loop (Lines 4–24) that runs while $\mathbb{Q}$ is not empty. The loop extracts the first element v from $\mathbb{Q}$ and adds it to $\mathbb{P}$ (Line 5). The axioms that label the edges of the path π_v from ρ to v in $\varPsi$ are collected in the set $\mathcal{M}_{\text{ex}}$ (Line 6). After that, the algorithm checks whether π_v is redundant via the function IS-PATH-REDUNDANT($\varPsi, \rho, \mathcal{O}_{\text{ex}}, \mathbb{P}$). The path π_v is *redundant* iff there exists an explored node $w \in \mathbb{P}$ such that (a) the axioms in $\mathcal{O}_{\text{ex}}$ are exactly the axioms labelling the edges of the path π_w from ρ to w in $\varPsi$, or (b) w is a leaf node of $\varPsi$ and the edges of π_w are only labelled with axioms from $\mathcal{O}_{\text{ex}}$. Case (a) corresponds to *early path termination* in [23,39]: the existence of π_w implies that all possible extensions of π_v have already been considered. Case (b) implies that the axioms labelling the edges of π_w lead to the fact that α can not be entailed be the remaining TBox when removed from $\mathcal{M}$. Therefore, by monotonicity of $\models$, we infer that removing $\mathcal{O}_{\text{ex}}$ from $\mathcal{M}$ also has the same consequence implying that we do not need to explore π_v and all its extensions.

If $\mathcal{M} \setminus \mathcal{M}_{\text{ex}} \not\models \alpha$ (Lines 9–10), the current iteration can be terminated immediately as no subset of $\mathcal{M} \setminus \mathcal{M}_{\text{ex}}$ can be a justification of $\mathcal{M}$ w.r.t. α. In contrast to other black-box algorithms for computing justifications, we additionally check whether $\mathcal{M} \setminus \mathcal{M}_{\text{ex}}$ is a subset of $\mathcal{U}$. If it is the case, no new axioms belonging to the union of all justifications can appear in this sub-tree. Hence, the algorithm does not need to explore it any further. Subsequently, the variable $\mathcal{M}$ that will hold a justification of $\mathcal{M} \setminus \mathcal{M}_{\text{ex}}$ is initialised with $\emptyset$. At this point we can check if a justification $\mathcal{M}' \in \mathbb{M}$ has already been computed for which $\mathcal{O}_{\text{ex}} \cap \mathcal{M}' = \emptyset$ (Lines 14–15) holds, in which case we set $\mathcal{M}$ to $\mathcal{M}'$. This optimisation step can also be found in [23, 39] and it allows us to avoid a costly call to the Single-Justification procedure. Otherwise, in Line 17 we call Single-Justification on $\mathcal{M} \setminus \mathcal{O}_{\text{ex}}$ to obtain a justification of α w.r.t. $\mathcal{M} \setminus \mathcal{O}_{\text{ex}}$. We then check whether $\mathcal{M}$ is equal to $\mathcal{C}$ (Lines 18–19), in which case the search for additional justifications can be terminated (recall Corollary 4). Otherwise, the justification $\mathcal{M}$ is added to $\mathbb{M}$ in Line 20 and the union of all justifications is updated in Line 21. Finally, for every $\beta \in \mathcal{M} \setminus \mathcal{C}$, the algorithm extends the tree Ψ in Lines 22–24 by adding a child v_α to v, connected by an edge labelled with β. Note that it is sufficient to take $\beta \notin \mathcal{C}$ as a set $\mathcal{M}$ with $\mathcal{C} \not\subseteq \mathcal{M}$ cannot be a justification of $\mathcal{O}$ w.r.t. α. The procedure finishes by returning the set $\mathcal{U}$.

Note that this algorithm only adds justifications to $\mathbb{M}$. For completeness, one can show that the locality-based module $\mathcal{M}$ of $\mathcal{O}$ w.r.t. $\mathsf{sig}(\alpha)$ contains all justifications of $\mathcal{O}$ w.r.t. α. Moreover, it is easy to see that the proposed optimisations do not lead to a justification not being computed. Overall, we obtain the following result.

Theorem 5. *Let $\mathcal{O}$ be an ontology, α a GCI, and $\mathcal{C} \subseteq \mathcal{O}$ the core of α w.r.t. $\mathcal{O}$. The procedure* Union-of-All-Justifications *computes the union of all justifications of $\mathcal{O}$ w.r.t. α.*

Algorithm 3 terminates on any input as the paths in the module search tree Ψ for $\mathcal{O}$ constructed during the execution represent all the permutations of the axioms in $\mathcal{O}$ that are relevant for finding all minimal modules. It is easy to see that the procedure Union-of-All-Justifications runs in exponential time in size of $\mathcal{O}$ in the worst case.

4.2 MUS Membership Algorithm (MUS-MEM)

It has been well-investigated that one can encode the problem of computing justifications to the problem of computing MUSes of CNF formula. One first needs to transfer all axioms and a given conclusion to CNF formulae and then uses a SAT-solver to compute a MUS. Finally, the corresponding axioms of MUS is a justification for a given conclusion. For a general overview on how this process works see [30].

$$\mathbf{R}_{\mathbf{A}}^{+}\ \frac{}{H \sqsubseteq A} : A \sqsubseteq H \qquad\qquad \mathbf{R}_{\mathbf{A}}^{-}\ \frac{H \sqsubseteq N \sqcup A}{H \sqsubseteq N} : \neg A \sqsubseteq H$$

$$\mathbf{R}_{\sqcap}^{\mathbf{n}}\ \frac{\{H \sqsubseteq N_i \sqcup A_i\}_{i=1}^{n}}{H \sqsubseteq \sqcup_{i=1}^{n} N_i \sqcup M} : \sqcap_{i=1}^{n} A_i \sqsubseteq M \in \mathcal{O}$$

$$\mathbf{R}_{\exists}^{+}\ \frac{H \sqsubseteq N \sqcup A}{H \sqsubseteq N \sqcup \exists r.B} : A \sqsubseteq \exists r.B \in \mathcal{O}$$

$$\mathbf{R}_{\exists}^{-}\ \frac{H \sqsubseteq M \sqcup \exists r.K, K \sqsubseteq N \sqcup A}{H \sqsubseteq M \sqcup B \sqcup \exists r.(K \sqcap \neg A)} : \exists s.A \sqsubseteq B \in \mathcal{O}, \mathcal{O} \models r \sqsubseteq s$$

$$\mathbf{R}_{\exists}^{\perp}\ \frac{H \sqsubseteq M \sqcup \exists r.K, K \sqsubseteq \perp}{H \sqsubseteq M}$$

$$\mathbf{R}_{\forall}\ \frac{H \sqsubseteq M \sqcup \exists r.K, H \sqsubseteq N \sqcup A}{H \sqsubseteq M \sqcup B \sqcup \exists r.(K \sqcap B)} : A \sqsubseteq \forall s.B \in \mathcal{O}, \mathcal{O} \models r \sqsubseteq s$$

Table 1: Inference rules of *condor*

We now show how to compute the union of all justifications of a GCI by a membership approach. The idea is to check the membership of each axiom, i.e., whether it is a member of some justification. We further encode it to the problem of checking each CNF-formula whether it is a member of some MUS.

The MUS-MEM approach first, as a pre-processing step, computes a CNF formula ϕ using the consequence-based reasoner *condor* [43].[4] Afterwards, it computes the union of all justifications of $\alpha \in \mathcal{O}$ by checking the membership for each axiom using the SAT-tool *cmMUS* [22] over ϕ. The two steps of our method is detailed below:

1. **Compute CNF formula ϕ.** Let H, K denote (possibly empty) conjunctions of concepts, and M, N (possibly empty) disjunctions of concepts; *condor* classifies the TBox through the inference rules in Table 1. Each inference rule can be rewritten as a clause. For example, the $\mathbf{R}_{\exists}^{\perp}$ rule can be rewritten to the clause $\neg p_1 \vee \neg p_2 \vee p_3$ if we denote the GCIs $H \sqsubseteq M \sqcup \exists r.K, K \sqsubseteq \perp$, and $H \sqsubseteq M$ as literals p_1, p_2, p_3, respectively. The CNF formula ϕ is the conjunction of all the clauses corresponding to all the applied inference rules during the classification process [37, 42].
2. **Check membership of each axiom using *cmMUS*.** Given a CNF formula ϕ and a subformula $\phi' \subseteq \phi$, the algorithm *cmMUS* is used to decide whether there is a MUS $\phi'' \subseteq \phi$ with $\phi' \cap \phi'' \neq \emptyset$. We set $cmMUS(\phi, \phi') = 1$ if there exists such MUS ϕ'' and 0 otherwise. To check membership, we need to define two objects:
   (a) a CNF-formula $\phi_{\mathcal{O}} = \wedge_{\beta \in \mathcal{O}} p_\beta$, where each literal p_β corresponds to an axiom $\beta \in \mathcal{O}$, and $\phi_\alpha = \neg p_\alpha$, where α is the given conclusion;
   (b) $\psi_\alpha = \phi \wedge \phi_{\mathcal{O}} \wedge \phi_\alpha$.

[4] We restrict to $\mathcal{ALCH}$ in this section as *condor* only accepts $\mathcal{ALCH}$-TBoxes.

Then we can see the following facts hold: Firstly, ψ_α is unsatisfiable. Moreover, each MUS $\psi' \subseteq \psi_\alpha$ corresponds to a justification of α. Finally, $\forall \beta \in \mathcal{O}$, $\mathit{cmMUS}(\psi_\alpha, p_\beta) = 1$ iff β belongs to some justifications of α.

An important optimization is based on the fact that only a small number of clauses in ϕ are related to the derivation of α. In practice, (i) $\phi' \subseteq \phi$ is the subformula contributing to the derivation of α obtained by tracing back from α, (ii) $\phi'_{\mathcal{O}} \subseteq \phi_{\mathcal{O}}$ is the subformula including only $\beta \in \mathcal{O}$ that appears in ϕ'. Using $\psi'_\alpha = \phi' \wedge \phi'_{\mathcal{O}} \wedge \phi_\alpha$ instead of ψ_α as the input of algorithm *cmMUS* can significantly accelerate the *cmMUS* algorithm.

Theorem 6. *Let $\mathcal{O}$ be an ontology and α a GCI. The procedure* MUS-MEM *algorithm computes the union of all justifications of $\mathcal{O}$ w.r.t. α.*

Regarding to the computational complexity of the MUS-MEM algorithm, in general, the classification of an $\mathcal{ALC}$ TBox requires exponential time. Since the MUS-membership problem is Σ_2^P-complete [28], it follows that this method runs in exponential time overall.

5 Repairing Ontologies

Similar to justifications, it is common to have multiple repairs for an unwanted consequence. Instead of treating all the repairs equally, in this section we propose a notion of optimal repair and provide a method for computing all such optimal repairs. Therefore, knowledge engineers can be better guided while repairing erroneous conclusions.

Definition 7 (Optimal Repair). *Let $\mathcal{O}$ be an ontology, α be a GCI, and* $\mathrm{Rep}(\mathcal{O}, \alpha)$ *be the set of all repairs for $\mathcal{O} \models \alpha$. $\mathcal{R} \in \mathrm{Rep}(\mathcal{O}, \alpha)$ is an* optimal repair *for $\mathcal{O} \models \alpha$, if $|\mathcal{R}| \geq |\mathcal{R}'|$ holds for every $\mathcal{R}' \in \mathrm{Rep}(\mathcal{O}, \alpha)$.*

That is, an optimal repair is a repair with the largest cardinality. It is also important to recall the notion of a hitting set. Given a set of sets $\mathbb{S}$, $\mathcal{S}$ is a minimal *hitting set* for $\mathbb{S}$ if $\mathcal{S} \cap s \neq \emptyset$ for every $s \in \mathbb{S}$. $\mathcal{S}$ is a smallest minimal hitting set if it is of minimal cardinality among all hitting sets. We can compute the set of all optimal repairs through a hitting set computation [8, 29, 41].

Proposition 8. *Let* $\mathrm{Just}(\mathcal{O}, \alpha)$ *be the set of all justifications for the GCI α w.r.t. the ontology $\mathcal{O}$. If $\mathbb{S}$ is the set of all smallest minimal hitting sets for* $\mathrm{Just}(\mathcal{O}, \alpha)$*, then $\{\mathcal{O} \setminus \mathcal{S} \mid \mathcal{S} \in \mathbb{S}\}$ is the set of all optimal repairs for $\mathcal{O} \models \alpha$.*

When the core is not empty, a set consisting of single axiom from this core is a smallest hitting set for all justifications. We get the following corollary of Proposition 8, stating how to compute all optimal repairs faster in this case.

Corollary 9. *Let $\mathcal{O}$ be an ontology, α be a GCI and $\mathcal{C}$ be the core for $\mathcal{O} \models \alpha$. If $\mathcal{C} \neq \emptyset$, then $\{\mathcal{O} \setminus \{\beta\} \mid \beta \in \mathcal{C}\}$ is the set of all optimal repairs for $\mathcal{O} \models \alpha$.*

It is easy to see that removing the union of all justifications from the given ontology results in the intersection of all repairs. Therefore, the union of all justifications can be used as a step towards deducing IAR entailments [36].

6 Evaluation

To evaluate the performance of our algorithms in real-world ontologies, we built a prototypical implementation.[5] The black-box algorithm is implemented in Java and uses the OWLAPI [20] to access ontologies and HermiT [19] as a standard reasoner. The MUS-membership algorithm (MUS-MEM) is implemented in Python and calls cmMUS [22] to detect whether a clause is a member of MUSes. The ontologies used in the evaluation come from the classification task at the 2014 ORE competition [33]. We selected the ontologies that have less than 10,000 axioms, for a total of 95 ontologies. In the experiments, we computed a single justification, the core, and the union of all justifications for all atomic concept inclusions that are entailed by the ontologies.An atomic concept inclusion is the inclusion of the form of $A \sqsubseteq B$, with $A, B \in \mathsf{N_C}$. All experiments ran on two processors Intel® Xeon® E5-2609v2 2.5GHz, 8 cores, 64Go, Ubuntu 18.04.

Computation time of the core vs. a single justification. In terms of computational complexity, as discussed in Section 3, computing the core and a single justification are equally hard problems. But the size of the remaining ontology reduces during the latter process. Intuitively, if $\mathcal{O}' \subseteq \mathcal{O}$, checking whether a subsumption is satisfied by $\mathcal{O}'$ would be faster than checking it on $\mathcal{O}$. Therefore, theoretically, the computation time of the core and a single justification should be comparable and computing the core should be slightly easier than computing a single justification. In practice, our evaluation justifies it. Table 2 provides some basic statistics for comparing the time to compute the core against computing a single justification. We can see from Table 2 that the mean computation time of the core and a single justification are very similar, around 0.4s. Generally speaking, computing the core is usually faster than computing one justification as expected.

	$\mathcal{J}$	$\mathcal{C}$
mean	0.400	0.456
std	3.649	4.273
min	0.001	0.001
25%	0.004	0.001
50%	0.009	0.002
75%	0.023	0.005
max	226.608	341.560

Table 2: Statistics of computation time (s) of core ($\mathcal{C}$) vs. single justification ($\mathcal{J}$)

To best of our knowledge, there is no existing tools to compute the intersection of all justifications directly. A naïve algorithm is to compute the intersection of finding all justifications. Thus, the naïve algorithm is as hard as computing all justifications, i.e., runs in exponential time in size of $\mathcal{O}$ in the worst case. However, our algorithm runs only in polynomial time in $|\mathcal{O}|$ in all cases. Therefore, we do not compare our algorithm with the naïve algorithm in the evaluation.

Computation time of the union of all justifications. As a benchmark, we compute all justifications and their union via the OWL API. As the MUS-MEM algorithm can compute the union of all justifications only for $\mathcal{ALCH}$ ontologies,

[5] The implementation is vailable at `https://github.com/JieyingChenChen/IntersectionAndUnionOfAllJust`.

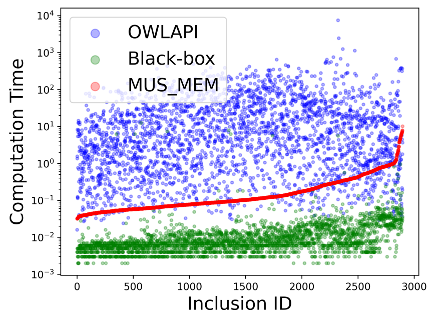

Fig. 1: Computation time (s) of the union for $\mathcal{ALCH}$-ontologies when there exist several justifications

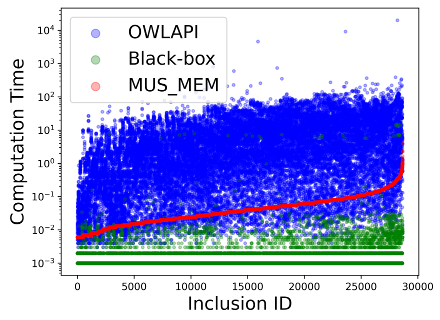

Fig. 2: Computation time (s) of the union for $\mathcal{ALCH}$-ontologies when there exists one justification

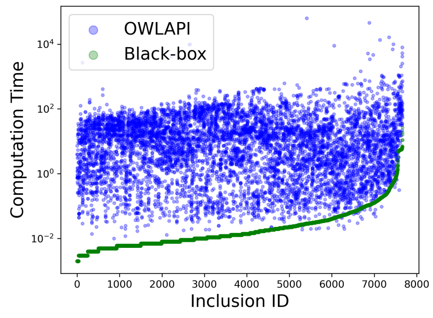

Fig. 3: Computation time (s) of the union for more expressive ontologies when there exist several justifications

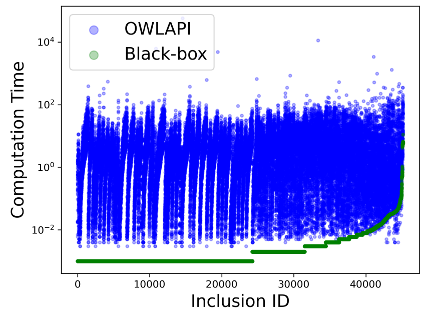

Fig. 4: Computation time (s) of the union for more expressive ontologies when there exists one justification

we divide the ontologies into two categories: (i) $\mathcal{ALCH}$ ontologies and (ii) with expressivity beyond $\mathcal{ALCH}$. The computation times for the union of all justifications for $\mathcal{ALCH}$ ontologies are shown in Figs. 1 and 2 for $\mathcal{ALCH}$ ontologies with several, or one justification, respectively. Figs. 3 and 4 show the same information for the class of more expressive ontologies. Figs. 1–4 plot the logarithmic computation time (in the Y-axis) of each test instance (in the X-axis). Each dot corresponds to computation time of the union by one of the methods tested. Note that the dots are much denser in Fig. 2 and 4 than Fig. 1 and 3 due to the fact that there exist more cases that have only one justification.

We order the conclusions along the X-axis by increasing order of computation time of MUS-MEM algorithms in Figs. 1 and Fig. 2, and by the black-box performance in the latter two figures. We observe from these plots that, generally, the green spots are located lower than the red and blue spots, which indicates that it took less time for black-box algorithm to compute the union

	Black-box	MUS-MEM	OWL API
mean	0.322	0.261	2.781
std	13.707	0.597	26.312
min	0.002	0.017	0.009
25%	0.004	0.069	0.123
50%	0.007	0.113	0.442
75%	0.016	0.259	1.726
max	970.834	10.255	1628.930

Table 3: Statistics of computation time (s) of the union when there exist several justifications for $\mathcal{ALCH}$ ontologies

	#JUST	$\mid\mathcal{C}\mid$	$\mid\mathcal{J}\mid$	$\mid\mathcal{U}\mid$
mean	3.0	2.3	4.0	4.1
std	3.2	1.7	1.9	3.2
min	2.0	0.0	1.0	1.0
25%	2.0	1.0	1.0	1.0
50%	2.0	2.0	2.0	3.0
75%	3.0	3.0	5.0	6.0
max	92.0	20.0	28.0	32.0

Table 4: Statistics of justification number, size of core ($\mathcal{C}$), a random justification ($\mathcal{J}$), and union of justifications ($\mathcal{U}$)

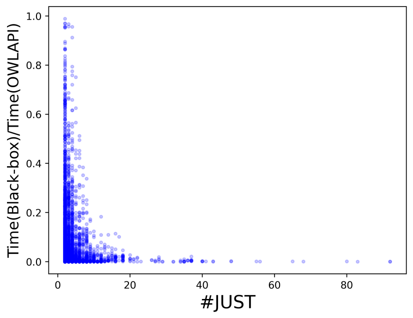

Fig. 5: Relation between #JUST and computation time of black-box/OWLAPI

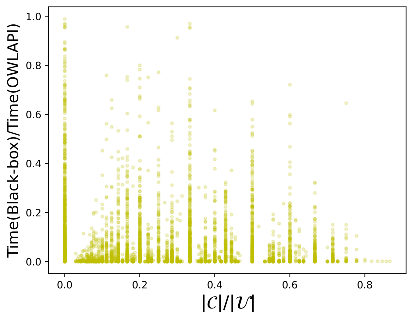

Fig. 6: Relation between $|\mathcal{C}|/|\mathcal{U}|$ and computation time of black-box/OWLAPI

of all justifications. When there exist several justifications, for more than 78% cases, MUS-MEM algorithm is faster than the OWLAPI. Detailed statistics of computation time of the union of all justifications when there exist several justifications for $\mathcal{ALCH}$ ontologies can be found in Table 3. We noticed that relatively large size of CNF-formulae (compared with the size of axioms) were generated in the cases that MUS-MEM algorithm is slower than the OWLAPI. Note that various numbers of CNF-formulae will be generated when we transform an axiom to CNF-formulae. Only less than 2% cases that MUS-MEM algorithm is faster than black-box algorithm.

Interestingly, we can see from Table 3 that the maximum computation time of the MUS-MEM algorithm is only 10.255s; much lower than the black-box algorithm and OWLAPI. Additionally, according to the Table 3, the standard deviation of computation time of the MUS-MEM algorithm for all cases in Fig. 2 is only 0.597, which is much lower than the Black-box algorithm and OWLAPI. When we plot the data and visualize the relationship between the number of all justifications (#JUST) and computation time of the union, we found that it tends to take longer for black-box algorithm, especially OWLAPI to compute the

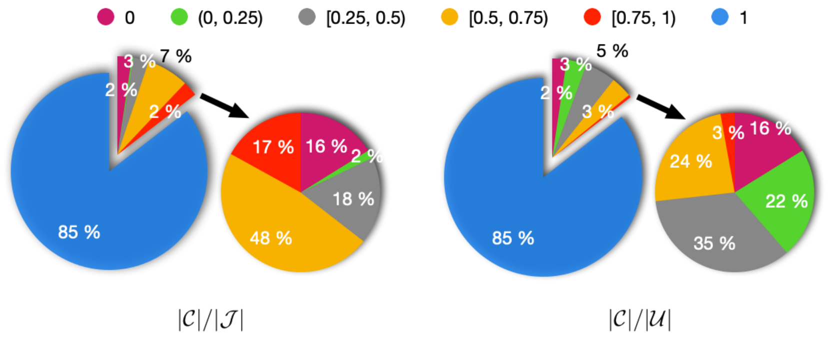

Fig. 7: Ratio of $|\mathcal{C}|$ to a random $|\mathcal{J}|$ (left) and ratio of $|\mathcal{C}|$ to $|\mathcal{U}|$ (right).

union of all justifications when #JUST increases. But the MUS-MEM algorithm seems to be less sensitive to this factor compared with other approaches.

We consider two factors that influence the performance difference of the black-box algorithm and OWLAPI: the number of justifications and the ratio of $|\mathcal{C}|/|\mathcal{U}|$. The main difference between these two approaches is that the black-box algorithm uses several strategies to reduce search space. If there exists a large number of justifications, the search space of OWLAPI approach becomes very large and the black-box approach may have to traverse several uninformative branches. Our black-box algorithm only considers the axioms that are not included in the union and terminates earlier if there are no axioms that will be included in it, which significantly prunes the search space. Additionally, the black-box algorithm also uses core to optimise the algorithm. If the ratio of $|\mathcal{C}|/|\mathcal{U}|$ is larger, there are less axioms left for checking. In order to further investigate it, we plot the relations between these two factors and the ratio of computation time of the black-box algorithm and OWLAPI in Fig. 5 and Fig. 6. Y-axis represents the ratio of computation time of the black-box algorithm and OWLAPI in both figures. X-axis represents the number of justifications in Fig. 5 and, in Fig. 6, it represents the ratio of $|\mathcal{C}|/|\mathcal{U}|$. We can see that when the number of justifications increases, in Fig. 5, the ratio of computation time of the black-box algorithm and OWLAPI decreases, which means that black-box algorithm become much faster than OWLAPI. Similarly, in Fig. 6, the ratio of computation time of the two approaches also reduced when the ratio of $|\mathcal{C}|/|\mathcal{U}|$ increases, which means that the black-box algorithm has better performance than OWLAPI when the intersection of the core and the union is larger.

In general, we can conclude that, when available, MUS-MEM tends to perform better than a direct use of the OWLAPI. Although in most cases in our experiments, the MUS-MEM algorithm is slower than black-box algorithm, its time difference when computing justifications various less. The black-box algorithm can be used for more expressive ontologies and outperforms OWLAPI, especially when there exists a large number of justifications and the size of the core is relatively small compared with the union.

Size comparisons for justifications, cores, and unions of justifications. Fig. 7 illustrates the ratio of the size of the core to the size of a random justification and to the size of the union of all justifications. In our experiments, the core for only 2.35% conclusions is empty (magenta part of the charts), which means that we could use Corollary 9 to compute optimal repairs for 97.65% (100%-2.35%) of the cases. Moreover, for more than 85% cases, the size of a justification ($|\mathcal{J}|$) equals to the size of the core ($|\mathcal{C}|$), which indicates that there exists only one justification (blue part of the charts). When several justifications exist (the second chart from the left of Fig. 7), the ratio of $|\mathcal{C}|$ to a random $|\mathcal{J}|$ falls between 0.5 to 0.75 (yellow part of the charts) for almost half of the cases. The right-most chart displays the distribution of the ratio of $|\mathcal{C}|$ to the size of the union of all justifications $|\mathcal{U}|$ when there exist multiple justifications. The ratio distributes quite evenly between 0 (not including) to 0.75. Additionally, the core is empty for only 16% subsumptions even when several justifications exist. See to Table 4 for the statistics information of the size of the core, the union and a single justification when multiple justifications exist.

7 Conclusions

We presented algorithms for computing the core (that is, the intersection of all justifications) and the union of all justifications for a given DL consequence. Our black-box algorithm is based on repeated calls to a reasoner, and hence apply for ontologies and consequences of any expressivity, as long as a reasoner exists. Whilst our MUS-based approach for computing the union of all justifications depends on the properties of the $\mathcal{ALCH}$ consequence-based method implemented by *condor*. Still, the approach should be generalisable without major problems to any language for which consequence-based reasoning methods exists like, for instance, $\mathcal{SROIQ}$ [17, 18]. As an application of our work, we study how to find optimal repairs effectively, through the information provided by the core and the union of all justifications.

Through an empirical analysis, run over more than 100,000 consequences from almost a hundred ontologies from the 2014 ORE competition we observe that our methods behave better in practice than the usual approach through the OWLAPI. Our experiments also confirm the observation that has already been made for light-weight ontologies [44], and to a smaller degree in the ontologies from the BioPortal corpus [10]; namely, that consequences tend to have one, or only a few, overlapping justifications. We also explored the fact that, in our experiments, the efficient core computation algorithm could find the optimal repairs in more than 97% of the test instances: those with non-empty core, where removing any axiom from it leads to an optimal repair.

It remains to be seen how these results change in the presence of larger ontologies. In particular, the instances considered had a limited number of justifications. We expect that the improvements observed would increase as more justifications are encountered.

References

1. M. F. Arif, C. Mencía, A. Ignatiev, N. Manthey, R. Peñaloza, and J. Marques-Silva. Beacon: An efficient sat-based tool for debugging $\mathcal{EL}^+$ ontologies. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 521–530. Springer, 2016.
2. M. F. Arif, C. Mencía, and J. Marques-Silva. Efficient axiom pinpointing with EL2MCS. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 225–233. Springer, 2015.
3. M. F. Arif, C. Mencía, and J. Marques-Silva. Efficient MUS enumeration of Horn formulae with applications to axiom pinpointing. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 324–342. Springer, 2015.
4. F. Baader, S. Brandt, and C. Lutz. Pushing the EL envelope further. In *In Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*, 2008.
5. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications.* Cambridge University Press, 2 edition, June 2010.
6. F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. *Journal of Automated Reasoning*, 14(1):149–180, 1995.
7. F. Baader and R. Peñaloza. Automata-based axiom pinpointing. *J. Autom. Reason.*, 45(2):91–129, 2010.
8. F. Baader and R. Peñaloza. Axiom pinpointing in general tableaux. *J. Log. Comput.*, 20(1):5–34, 2010.
9. F. Baader, R. Peñaloza, and B. Suntisrivaraporn. Pinpointing in the description logic EL+. In J. Hertzberg, M. Beetz, and R. Englert, editors, *Proceedings of the 30th Annual German Conference on AI, KI 2007*, volume 4667 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 2007.
10. S. P. Bail. *The justificatory structure of OWL ontologies.* PhD thesis, University of Manchester, UK, 2013.
11. C.-L. Chang and R. C.-T. Lee. *Symbolic logic and mechanical theorem proving.* Academic press, 2014.
12. J. Chen, M. Ludwig, Y. Ma, and D. Walther. Zooming in on ontologies: Minimal modules and best excerpts. In *Proc. of ISWC'17, Part I*, volume 10587 of *Lecture Notes in Computer Science*, pages 173–189. Springer, 2017.
13. J. Chen, M. Ludwig, Y. Ma, and D. Walther. Computing minimal projection modules for *ELH*ˆr -terminologies. In F. Calimeri, N. Leone, and M. Manna, editors, *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings*, volume 11468 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2019.
14. J. Chen, M. Ludwig, and D. Walther. On computing minimal el-subsumption modules. In *Proceedings of the Joint Ontology Workshops 2016 Episode 2: The French Summer of Ontology co-located with the 9th International Conference on Formal Ontology in Information Systems (FOIS 2016), Annecy, France, July 6-9, 2016*, volume 1660 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.
15. J. Chen, M. Ludwig, and D. Walther. Computing minimal subsumption modules of ontologies. In *Proc. of GCAI'18*, pages 41–53, 2018.
16. J. Chen, Y. Ma, R. Peñaloza, and H. Yang. Union and intersection of all justifications (extended abstract). In *Proceedings of the 34th International Workshop on Description Logics DL'21*, volume 2954 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2021.

17. D. T. Cucala, B. C. Grau, and I. Horrocks. Consequence-based reasoning for description logics with disjunction, inverse roles, number restrictions, and nominals. In J. Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*, pages 1970–1976. ijcai.org, 2018.
18. D. T. Cucala, B. C. Grau, and I. Horrocks. Sequoia: A consequence based reasoner for SROIQ. In M. Simkus and G. E. Weddell, editors, *Proceedings of the 32nd International Workshop on Description Logics*, volume 2373 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019.
19. B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang. HermiT: an OWL 2 reasoner. *Journal of Automated Reasoning*, 53(3):245–269, 2014.
20. M. Horridge and S. Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.
21. I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning*, pages 57–67. AAAI Press, 2006.
22. M. Janota and J. Marques-Silva. cmMUS: A tool for circumscription-based mus membership testing. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 266–271. Springer, 2011.
23. A. Kalyanpur, B. Parsia, M. Horridge, and E. Sirin. Finding all justifications of OWL DL entailments. In *Proceedings of ISWC 2007 & ASWC 2007*, volume 4825 of *LNCS*, pages 267–280. Springer, 2007.
24. A. Kalyanpur, B. Parsia, E. Sirin, and J. Hendler. Debugging unsatisfiable classes in OWL ontologies. *Journal of Web Semantics*, 3(4):268–293, 2005.
25. A. A. Kalyanpur. *Debugging and repair of OWL ontologies*. PhD thesis, 2006.
26. Y. Kazakov and P. Skočovskỳ. Enumerating justifications using resolution. In *International Joint Conference on Automated Reasoning*, pages 609–626. Springer, 2018.
27. P. Koopmann and J. Chen. Deductive module extraction for expressive description logics. In C. Bessiere, editor, *Proceedings of IJCAI'20*, pages 1636–1643. ijcai.org, 2020.
28. P. Liberatore. Redundancy in logic i: Cnf propositional formulae. *Artificial Intelligence*, 163(2):203–232, 2005.
29. M. H. Liffiton and K. A. Sakallah. On finding all minimally unsatisfiable subformulas. In F. Bacchus and T. Walsh, editors, *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT 2005)*, volume 3569 of *Lecture Notes in Computer Science*, pages 173–186. Springer, 2005.
30. N. Manthey, R. Peñaloza, and S. Rudolph. Efficient axiom pinpointing in EL using sat technology. In *Description Logics*, 2016.
31. N. Manthey, R. Peñaloza, and S. Rudolph. Satpin: Axiom pinpointing for lightweight description logics through incremental SAT. *Künstliche Intell.*, 34(3):389–394, 2020.
32. A. Ozaki and R. Peñaloza. Consequence-based axiom pinpointing. In D. Ciucci, G. Pasi, and B. Vantaggi, editors, *Proceedings of the 12th International Conference on Scalable Uncertainty Management (SUM 2018)*, volume 11142 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2018.
33. B. Parsia, N. Matentzoglu, R. S. Gonçalves, B. Glimm, and A. Steigmiller. The OWL reasoner evaluation (ORE) 2015 competition report. *Journal of Automated Reasoning*, pages 1–28, 2015.
34. B. Parsia, E. Sirin, and A. Kalyanpur. Debugging owl ontologies. In *Proceedings of the 14th international conference on World Wide Web*, pages 633–640, 2005.

35. R. Peñaloza. Axiom pinpointing. In G. Cota, M. Daquino, and G. L. Pozzato, editors, *Applications and Practices in Ontology Design, Extraction, and Reasoning*, volume 49 of *Studies on the Semantic Web*, pages 162–177. IOS Press, 2020.
36. R. Peñaloza. Error-tolerance and error management in lightweight description logics. *Künstliche Intell.*, 34(4):491–500, 2020.
37. R. Peñaloza, C. Mencía, A. Ignatiev, and J. Marques-Silva. Lean kernels in description logics. In E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler, and O. Hartig, editors, *Proceeding of ESWC'17*, volume 10249 of *Lecture Notes in Computer Science*, pages 518–533, 2017.
38. R. Penaloza and B. Sertkaya. Understanding the complexity of axiom pinpointing in lightweight description logics. *Artificial Intelligence*, 250:80–104, 2017.
39. R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.
40. U. Sattler, T. Schneider, and M. Zakharyaschev. Which kind of module should I extract? In *Proceedings of DL'09*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.
41. S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In G. Gottlob and T. Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 355–362. Morgan Kaufmann, 2003.
42. R. Sebastiani and M. Vescovi. Axiom pinpointing in lightweight description logics via Horn-SAT encoding and conflict analysis. In R. A. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction*, volume 5663 of *Lecture Notes in Computer Science*, pages 84–99. Springer, 2009.
43. F. Simancik, Y. Kazakov, and I. Horrocks. Consequence-based reasoning beyond horn ontologies. In T. Walsh, editor, *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 1093–1098. IJCAI/AAAI, 2011.
44. B. Suntisrivaraporn. *Polynomial time reasoning support for design and maintenance of large-scale biomedical ontologies*. PhD thesis, Dresden University of Technology, Germany, 2009.
45. B. Suntisrivaraporn, G. Qi, Q. Ji, and P. Haase. A modularization-based approach to finding all justifications for OWL DL entailments. In J. Domingue and C. Anutariya, editors, *The Semantic Web*, pages 1–15, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.