Enumerating Justifications using Resolution

Yevgeny Kazakov[1] and Peter Skočovský[2]

[1] The University of Ulm, Germany
yevgeny.kazakov@uni-ulm.de
[2] peter.skoco@gmail.com

Abstract. If a conclusion follows from a set of axioms, then its justification is a minimal subset of axioms for which the entailment holds. An entailment can have several justifications. Such justifications are commonly used for the purpose of debugging of incorrect entailments in Description Logic ontologies. Recently a number of SAT-based methods have been proposed that can enumerate all justifications for entailments in light-weight ontologies languages, such as $\mathcal{EL}$. These methods work by encoding $\mathcal{EL}$ inferences in propositional Horn logic, and finding minimal models that correspond to justifications using SAT solvers. In this paper, we propose a new procedure for enumeration of justifications that uses resolution with answer literals instead of SAT solvers. In comparison to SAT-based methods, our procedure can enumerate justifications in any user-defined order that extends the set inclusion relation. The procedure is easy to implement and, like resolution, can be parametrized with ordering and selection strategies. We have implemented this procedure in PULi—a new Java-based Proof Utility Library, and performed an empirical comparison of (several strategies of) our procedure and SAT-based tools on popular $\mathcal{EL}$ ontologies. The experiments show that our procedure provides a comparable, and often better performance than those highly optimized tools. For example, using one of the strategies, we were able for the first time to compute all justifications for all entailed concept subsumptions in one of the largest commonly used medical ontology Snomed CT.

1 Introduction and Motivation

Axiom pinpointing, or computing justifications—minimal subsets of axioms of the ontology that entail a given logical consequence—has been a widely studied research topic in ontology engineering [1–12]. Most of the recent methods focus on the so-called $\mathcal{EL}$ family of Description Logics (DLs), in which logical consequences can be proved by deriving new axioms from existing ones using inference rules. The resulting inferences are usually encoded as propositional (Horn) clauses, and justifications are computed from them using (modifications of) SAT solvers. To ensure correctness, the input inference set must be complete, that is, the inferences are enough to derive the consequence from any subset of the ontology from which it follows.

In this paper, we present a new resolution-based procedure that enumerates all justifications of an entailment given a complete set of inferences. Apart from requiring completeness, the form of inferences can be arbitrary and does not depend on any logic. For example, our method can be used with the inferences provided by existing consequence-based procedures [13–16]. The procedure can enumerate justifications in any given order, provided it extends the proper subset relation on sets of axioms. Performance of

the procedure depends on the strategy it follows while enumerating justifications. We have empirically evaluated three simple strategies and experimentally compared our procedure with other highly optimized justification computation tools.

The paper is organized as follows. In Section 2 we describe related work. Section 3 introduces background on DLs, justifications, and resolution. In Section 4 we present the new procedure, and in Section 5 we describe its implementation and empirical evaluation.

2 Related Work

There are, generally, two kinds of procedures for computing justifications [9] using a DL reasoner. *Black-Box* procedures use a reasoner solely for entailment checking, and thus can be used for any reasoner and DL. *Glass-Box* procedures require additional information from a reasoner, such as inferences that the reasoner has used, and thus can only work with reasoners that can provide such information.

In a nutshell, Black-Box procedures [4, 6, 7, 10] systematically explore subsets of axioms and check using a reasoner, which of these subsets entail the given logical conclusion, and which not. Unnecessary tests are avoided using the monotonicity property of the entailment.

Finding *one* justification is relatively easy. Starting from the set of all axioms that entail the conclusion, one tries to remove axioms one by one. If after the removal the entailment does not hold, the axiom is inserted back. This results in a subset from which no axiom can be further removed without breaking the entailment, i.e., a justification for the entailment. This justification, however, may be not unique as the result depends on the order in which the axioms are considered for removal. In the worst case, there can be exponentially-many different justifications. So, unsurprisingly, there is no polynomial procedure for computing *all* justifications even in languages such as $\mathcal{EL}$, for which entailment checking is polynomially decidable [17]. Further, computing all justifications is even hard in the number of justifications: it is already NP-hard to verify, given a set of justifications, if there exists another justification not in this set [4]. Hence, in practice, one is interested in algorithms for *enumeration* of justifications, i.e., algorithms that can return justifications without necessarily finishing computing all of them.

Most existing algorithms for enumeration of justifications rely, in one way or the other, on the *hitting set duality* that was introduced in the field of Model Based Diagnosis [18, 19] and later adapted for DLs [5, 7]. A *hitting set* for a collection of sets is a set containing at least one element from each set in the collection. A minimal hitting set of all justifications for an entailment is a *repair*—a minimal set of axioms, removal of which breaks the entailment. Dually, every minimal hitting set of all repairs is a justification. Existing justification enumeration algorithms, in fact, also (implicitly) enumerate repairs in addition to justifications.

Suppose that one has computed some justifications and repairs for an entailment. To find a new justification or a repair, it is sufficient to find a set M of axioms that has at least one axiom from each repair and misses at least one axiom from each justification. I.e., M is a hitting set of the computed repairs, and its complement (within the set of all axioms) is a hitting set of the computed justifications. If no such set M exists, then there

are no new justifications or repairs, for otherwise a new justification or the complement of a new repair would satisfy this requirement. Now, if M entails the conclusion, then a justification can be extracted from M by repeatedly removing axioms as described before. This justification will be different from all previously computed justifications because M is not a super-set of any of them. On the other hand, if M does not entail the conclusion, then a new repair can be extracted from the complement of M similarly, by removing axioms until the set is no longer a repair. Likewise, this will be a new repair since the complement of M is not a super-set of any previously computed repairs. By repeating this procedure, one can enumerate all repairs and all justifications.

Finding a suitable set M satisfying the requirements above can be accomplished using a propositional SAT solver. Specifically, for each computed repair, we add a clause consisting of atoms corresponding to the axioms in the repair. Similarly, for each computed justification, we add a clause consisting of the negations of atoms corresponding to the axioms in the justification. Then for every model of these clauses, the set M consisting of the axioms whose atoms are true, satisfies the requirements. SAT solvers can also be used to optimize the entailment tests, which are usually main bottleneck of Black-Box procedures. For example, in the case of $\mathcal{EL}$, all necessary information about entailments from subsets of axioms can be represented by (a polynomial number of) inferences. Every $\mathcal{EL}$ inference can be translated to a propositional (Horn) clause with the negative atoms corresponding to the premises of the inference, and the positive atom corresponding to its conclusion. A conclusion is derivable from axioms using the inferences iff the translation of the inferences entails the (Horn) clause whose negative atoms correspond to axioms and the positive atoms corresponds to the conclusion.

The above Glass-Box procedure was first proposed and implemented in EL+SAT [11, 12], and later improved in EL2MUS [3] and SATPin [8]. These tools differ mainly in the way how they enumerate models corresponding to the candidate sets M, and further optimizations employed. EL+SAT and EL2MUS use two instances of a SAT solver—one for enumeration of candidate models, and another for verifying derivability using inferences—whereas SATPin [8] uses one (modified) SAT solver for both of these tasks. The encoding for finding the candidate set M described above is most close to the implementation of EL2MUS. EL+SAT and SATPin do not explicitly enumerate repairs, but each time a model is found that corresponds to a set M that does not entail the conclusion, a "blocking" clause is added to ensure that such a model is not returned again. However, the number of such blocking clauses is at least as large as the number of repairs. Further differences are that in EL2MUS the entailment checking solver is specialized in Horn clauses, and that EL+SAT and SATPin extract justifications by a deletion-based procedure (as outlined above), while EL2MUS uses an insertion-based procedure. Another tool EL2MCS [2] uses MaxSAT [20, 21] to compute all repairs and extracts justifications from them using the hitting set duality, but it cannot return any justification before all repairs are computed. Further, BEACON [1] is a tool that integrates the justification procedure of EL2MUS.

Up to a few optimizations, the mentioned SAT-based tools use $\mathcal{EL}$ inferences only for the entailment checks. Had they delegated the entailment checks to a separate DL reasoner, they could be regarded as Black-Box. Our approach uses a similar encoding of inferences in propositional logic, however, it relies neither on a SAT solver nor on the

Table 1. The syntax and semantics of $\mathcal{EL}$

	Syntax	Semantics
Roles:		
atomic role	R	$R^{\mathcal{I}}$
Concepts:		
atomic concept	A	$A^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists R.C$	$\{x \mid \exists y \in C^{\mathcal{I}} : \langle x, y\rangle \in R^{\mathcal{I}}\}$
Axioms:		
concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$

$$\mathbf{R_0}\ \frac{}{C \sqsubseteq C} \qquad \mathbf{R}^{-}_{\sqcap}\ \frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2} \qquad \mathbf{R}_{\sqsubseteq}\ \frac{C \sqsubseteq D}{C \sqsubseteq E} : D \sqsubseteq E \in \mathcal{O}$$

$$\mathbf{R}_{\top}\ \frac{}{C \sqsubseteq \top} \qquad \mathbf{R}^{+}_{\sqcap}\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2} \qquad \mathbf{R}_{\exists}\ \frac{C \sqsubseteq \exists R.D \quad D \sqsubseteq E}{C \sqsubseteq \exists R.E}$$

Fig. 1. The inference rules for reasoning in $\mathcal{EL}$

hitting set duality. In particular, our method does not enumerate repairs (explicitly or implicitly). As we operate on inferences directly, a Black-Box version of our procedure would not be possible.

3 Preliminaries

3.1 The Description Logic $\mathcal{EL}$

The syntax of $\mathcal{EL}$ is defined using a vocabulary consisting of countably infinite sets of *(atomic) roles* and *atomic concepts*. Complex *concepts* and *axioms* are defined recursively using Table 1. We use letters R, S for roles, C, D, E for concepts, and A, B for atomic concepts. An *ontology* is a finite set of axioms.

An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a nonempty set $\Delta^{\mathcal{I}}$ called the *domain* of $\mathcal{I}$ and an interpretation function $\cdot^{\mathcal{I}}$ that assigns to each role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to each atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$. This assignment is extended to complex concepts as shown in Table 1. $\mathcal{I}$ *satisfies* an axiom α (written $\mathcal{I} \models \alpha$) if the corresponding condition in Table 1 holds. $\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ (written $\mathcal{I} \models \mathcal{O}$) if $\mathcal{I}$ satisfies all axioms in $\mathcal{O}$. We say that $\mathcal{O}$ *entails* an axiom α (written $\mathcal{O} \models \alpha$), if every model of $\mathcal{O}$ satisfies α. A concept C is *subsumed* by D w.r.t. $\mathcal{O}$ if $\mathcal{O} \models C \sqsubseteq D$. The *ontology classification task* requires to compute all entailed subsumptions between atomic concepts occurring in $\mathcal{O}$.

Reasoning in $\mathcal{EL}$ can be performed by applying inference rules that derive subsumptions between concepts [17]. We use a variant of $\mathcal{EL}$ rules shown in Figure 1 that do not require normalization [14]. As usual, the premises of the rules (if any) are given above the horizontal line, and the conclusions below. Note that rule $\mathbf{R}_{\sqsubseteq}$ can only use $D \sqsubseteq E$ from the ontology $\mathcal{O}$. This *side condition* should be distinguished from the premises

of the rules, where one can use any derived axiom. This restriction has been made for efficiency reasons: if to use $D \sqsubseteq E$ as the second premise of $\mathbf{R}_{\sqsubseteq}$, like for rule $\mathbf{R}_{\exists}$, there would be too many unnecessary inferences.

The rules in Figure 1 are *sound* and *complete* for entailment checking, i.e., the entailment $\mathcal{O} \models \alpha$ holds iff α is derivable from $\mathcal{O}$ using the rules.[3] Furthermore, for deriving α, it is sufficient to use inferences that contain only concepts appearing in $\mathcal{O}$ or α [14]. I.e., it is not necessary to apply $\mathbf{R}^{+}_{\sqcap}$ if $D_1 \sqcap D_2$ does not appear in $\mathcal{O}$ or α. This so-called *subformula property* implies that checking the $\mathcal{EL}$ entailment $\mathcal{O} \models \alpha$ can be performed in polynomial time [17, 14] since there are at most polynomially-many different rule applications that can use only concepts appearing in $\mathcal{O}$ or α.

3.2 Inferences, Support, and Justifications

Although our experimental evaluation is concerned about $\mathcal{EL}$, our method can be used with a large class of inference systems of which the system in Figure 1 is just one example. In general, we assume that the rules manipulate with objects that we call *axioms*, and an *ontology* is any finite set of such axioms. An *inference* is an expression *inf* of the form $\langle \alpha_1, \ldots, \alpha_n \vdash \alpha \rangle$ where $\alpha_1, \ldots, \alpha_n$ is a (possibly empty) sequence of axioms called the *premises* of *inf*, and α is an axiom called the *conclusion* of *inf*.

Let I be a set of inferences. An I-*derivation* from $\mathcal{O}$ is a sequence of inferences $d = \langle \mathit{inf}_1, \ldots, \mathit{inf}_k \rangle$ from I such that for every i with $(1 \leq i \leq k)$, and each premise α of inf_i that is not in $\mathcal{O}$, there exists $j < i$ such that α is the conclusion of inf_j. An axiom α is *derivable* from $\mathcal{O}$ using I (notation: $\mathcal{O} \vdash_{\mathsf{I}} \alpha$) if either $\alpha \in \mathcal{O}$ or there exists an I-derivation $d = \langle \mathit{inf}_1, \ldots, \mathit{inf}_k \rangle$ from $\mathcal{O}$ such that α is the conclusion of inf_k. A *support* for $\mathcal{O} \vdash_{\mathsf{I}} \alpha$ is a subset of axioms $\mathcal{O}' \subseteq \mathcal{O}$ such that $\mathcal{O}' \vdash_{\mathsf{I}} \alpha$. A *justification* for $\mathcal{O} \vdash_{\mathsf{I}} \alpha$ is a subset-minimal support for $\mathcal{O} \vdash_{\mathsf{I}} \alpha$.

Suppose that $\models$ is an entailment relation between ontologies and axioms. A *justification* for $\mathcal{O} \models \alpha$ (also sometimes called a *minimal axiom set* MinA [4]) is a minimal subset $\mathcal{O}' \subseteq \mathcal{O}$ such that $\mathcal{O}' \models \alpha$. An inference $\langle \alpha_1, \ldots, \alpha_n \vdash \alpha \rangle$ is *sound* if $\{\alpha_1, \ldots, \alpha_n\} \models \alpha$. A set of inferences I is *complete* for the entailment $\mathcal{O} \models \alpha$ if $\mathcal{O}' \models \alpha$ implies $\mathcal{O}' \vdash_{\mathsf{I}} \alpha$ for every subset $\mathcal{O}' \subseteq \mathcal{O}$. Note that if I is complete for $\mathcal{O} \models \alpha$ then $\mathcal{O}' \models \alpha$ iff $\mathcal{O}' \vdash_{\mathsf{I}} \alpha$ for every $\mathcal{O}' \subseteq \mathcal{O}$. In particular, justifications for $\mathcal{O} \vdash_{\mathsf{I}} \alpha$ coincide with justifications for $\mathcal{O} \models \alpha$.

Example 1. Consider the following applications of rules $\mathbf{R}^{-}_{\sqcap}$ and $\mathbf{R}^{+}_{\sqcap}$ in Figure 1: $\mathsf{I}_e = \{\langle A \sqsubseteq B \sqcap C \vdash A \sqsubseteq B \rangle,\ \langle A \sqsubseteq B \sqcap C \vdash A \sqsubseteq C \rangle,\ \langle A \sqsubseteq C, A \sqsubseteq B \vdash A \sqsubseteq C \sqcap B \rangle\}$. Thus, I_e is a set of inferences over $\mathcal{EL}$ axioms. Let $\mathcal{O}_e = \{A \sqsubseteq B \sqcap C; A \sqsubseteq B; A \sqsubseteq C\}$ be an $\mathcal{EL}$ ontology and $\alpha_e = A \sqsubseteq C \sqcap B$ an $\mathcal{EL}$ axiom. Note that $\mathcal{O}_e \vdash_{\mathsf{I}_e} \alpha_e$.

It is easy to see that $\mathcal{O}'_e = \{A \sqsubseteq B; A \sqsubseteq C\} \vdash_{\mathsf{I}_e} \alpha_e$ and $\mathcal{O}''_e = \{A \sqsubseteq B \sqcap C\} \vdash_{\mathsf{I}_e} \alpha_e$, but $\{A \sqsubseteq B\} \nvdash_{\mathsf{I}_e} \alpha_e$ and $\{A \sqsubseteq C\} \nvdash_{\mathsf{I}_e} \alpha_e$. Hence, $\mathcal{O}'_e$ and $\mathcal{O}''_e$ are justifications for $\mathcal{O}_e \vdash_{\mathsf{I}_e} \alpha_e$. All inferences in I_e are also sound for the $\mathcal{EL}$ entailment relation $\models$. Since $\mathcal{O}'_e$ and $\mathcal{O}''_e$ are the only two justifications for $\mathcal{O}_e \models \alpha_e$, the inference set I_e is complete for the entailment $\mathcal{O}_e \models \alpha_e$.

[3] Actually, w.l.o.g., one can also assume that the axioms in $\mathcal{O}$ are only used in the side conditions of rule $\mathbf{R}_{\sqsubseteq}$. Indeed, any axiom $D \sqsubseteq E \in \mathcal{O}$ can be derived in this way by first deriving a tautology $D \sqsubseteq D$ by $\mathbf{R_0}$ and then deriving $D \sqsubseteq E$ by $\mathbf{R}_{\sqsubseteq}$ from $D \sqsubseteq D$ using $D \sqsubseteq E \in \mathcal{O}$.

$$\textbf{Resolution}\ \frac{c \vee a \quad \neg a \vee d}{c \vee d} \qquad \textbf{Factoring}\ \frac{c \vee a \vee a}{c \vee a}$$

Fig. 2. Propositional resolution and factoring rules

3.3 Resolution with Answer Literals

Our procedure for enumeration of justifications is based on the *resolution calculus*, which is a popular method for automated theorem proving [22]. We will mainly use resolution for propositional Horn clauses. A (propositional) *literal* is either an atom $l = a$ (*positive literal*) or a negation of atom $l = \neg a$ (*negative literal*). A (propositional) *clause* is a disjunction of literals $c = l_1 \vee \cdots \vee l_n$, $n \geq 0$. As usual, we do not distinguish between the order of literals in clauses, i.e., we associate clauses with multisets of literals. Given two clauses c_1 and c_2, we denote by $c_1 \vee c_2$ the clause consisting of all literals from c_1 plus all literals from c_2. A clause is *Horn* if it has at most one positive literal. The *empty clause* $\square$ is the clause with $n = 0$ literals. The inference rules for the propositional resolution calculus are given in Figure 2. We say that a set of clauses S is *closed* under the resolution rules if S contains every clause derived by the rules in Figure 2 from S. The resolution calculus is *refutationally complete*: every set of clauses S closed under the resolution rules is satisfiable if and only if it does not contain the empty clause. This means that for checking satisfiability of the input set of clauses, it is sufficient to deductively close this set under the resolution rules and check if the empty clause is derived in the closure.

To reduce the number of resolution inferences (and hence the size of the closure) several *refinements* of the resolution calculus were proposed. The rules in Figure 2 can be restricted using orderings and selection functions [22]. In particular, for Horn clauses, it is sufficient to select one (positive or negative) literal in each clause, and require that the resolution inferences are applied only on those (Theorem 7.2 in [22]).[4] This strategy is called *resolution with free selection*. In addition to rule restrictions, one can also use a number of *simplification rules* that can remove or replace clauses in the closure S. We will use two such rules. *Elimination of duplicate literals* removes all duplicate literals from a clause (including duplicate negative literals). *Subsumption deletion* removes a clause c from S if there exists another *sub-clause* c' of c in S, i.e., $c = c' \vee c''$ for some (possibly empty) clause c''. In this case we say that c' *subsumes* c.

Example 2. Consider the set of Horn clauses 1-7 below. We apply resolution with free selection that selects the underlined literals in clauses. Clauses 8-10 are obtained by resolution inferences from clauses shown in angle braces on the right.

1: $\underline{\neg p_1} \vee p_2$	4: $\underline{p_1}$	7: $\underline{\neg p_4}$	8: $\underline{\neg p_3} \vee p_4$	$\langle 3, 5 \rangle$
2: $\underline{\neg p_1} \vee p_3$	5: $\underline{p_2}$		9: $\underline{p_4}$	$\langle 6, 8 \rangle$
3: $\underline{\neg p_2} \vee \neg p_3 \vee p_4$	6: $\underline{p_3}$		10: $\square$	$\langle 7, 9 \rangle$

Note that the resolution rule was not applied, e.g., to clauses 3 and 6 because literal $\neg p_3$ in clause 3 is not selected. Also note that many clauses in the closure above can be derived by several resolution inferences. For example, clause 5 can be obtained by

[4] Note that the factoring rule cannot apply to Horn clauses.

resolving clauses 1 and 4 and clause 6 by resolving 2 and 4. Therefore the empty clause 10 can be derived from several subsets of the original clauses 1-7.

The resolution calculus is mainly used for checking satisfiability of a clause set, and is not directly suitable for finding unsatisfiable subsets of clauses. To solve the latter problem, we use an extension of resolution with so-called answer literals [23]. To determine, which subsets of the input clauses are unsatisfiable, we add to every input clause a fresh positive *answer literal*. Resolution rules can then be applied to the extended clauses on the remaining (ordinary) literals using the usual orderings and selection functions. If some clause with answer literals is derived, then this clause with the answer literals removed, can be derived from the clauses for which the answer literals were introduced. In particular, if a clause containing only answer literals is derived, then the set of clauses that corresponds to these answer literals is unsatisfiable. Completeness of resolution means that all such unsatisfiable sets of clauses can be found in this way. If answer literals are added to some but not all clauses and a clause with only answer literals is derived, then the set of clauses that corresponds to the answer literals plus clauses without answer literals is unsatisfiable.

Example 3. Consider the clauses 1-7 from Example 2. Let us add answer literals a_1-a_3 to clauses 4-6 and apply the resolution rules on the remaining (underlined) literals like in Example 2, eliminating duplicate literals if they appear.

1:	$\underline{\neg p_1} \vee p_2$	8:	$\underline{p_2} \vee a_1$	$\langle 1,4\rangle$	15: $\underline{p_4} \vee a_1$	$\langle 9,11\rangle$
2:	$\underline{\neg p_1} \vee p_3$	9:	$\underline{p_3} \vee a_1$	$\langle 2,4\rangle$	16: $\boxed{\underline{a_2 \vee a_3}}$	$\langle 7,12\rangle$
3:	$\underline{\neg p_2} \vee \neg p_3 \vee p_4$	10:	$\underline{\neg p_3} \vee p_4 \vee a_2$	$\langle 3,5\rangle$	17: $\boxed{\underline{a_1 \vee a_2}}$	$\langle 7,13\rangle$
4:	$\underline{p_1} \vee a_1$	11:	$\underline{\neg p_3} \vee p_4 \vee a_1$	$\langle 3,8\rangle$	18: $\boxed{\underline{a_1 \vee a_3}}$	$\langle 7,14\rangle$
5:	$\underline{p_2} \vee a_2$	12:	$\underline{p_4} \vee a_2 \vee a_3$	$\langle 6,10\rangle$	19: $\boxed{a_1}$	$\langle 7,15\rangle$
6:	$\underline{p_3} \vee a_3$	13:	$\underline{p_4} \vee a_1 \vee a_2$	$\langle 9,10\rangle$		
7:	$\underline{\neg p_4}$	14:	$\underline{p_4} \vee a_1 \vee a_3$	$\langle 6,11\rangle$		

The framed clauses 16-19 contain only answer literals, so the corresponding sets of clauses are unsatisfiable in conjunction with the input clauses without answer literals. For example, clause 16 means that clauses 1-3, 5-7 are unsatisfiable and clause 19 means that clauses 1-4, 7 are also unsatisfiable. Note that clause 19 subsumes clauses 17-18; if subsumed clauses are deleted, we obtain only clauses with answer literals that correspond to *minimal* subsets of clauses 4-6 that are unsatisfiable in conjunction with the remaining input clauses 1-3, 7.

4 Enumerating Justifications using Resolution

In this section, we present a new procedure that, given an ontology $\mathcal{O}$, an inference set I and a goal axiom α_g, enumerates justifications for $\mathcal{O} \vdash_{\mathsf{I}} \alpha_g$. It uses the usual reduction of the derivability problem $\mathcal{O} \vdash_{\mathsf{I}} \alpha_g$ to satisfiability of propositional Horn clauses [11, 12, 2, 8] in combination with the resolution procedure with answer literals .

Given a derivability problem $\mathcal{O} \vdash_{\mathsf{I}} \alpha_g$, we assign to each axiom α_i occurring in I a fresh propositional atom p_{α_i}. Each inference $\langle \alpha_1, \ldots, \alpha_n \vdash \alpha\rangle \in \mathsf{I}$ is then translated to the Horn clause $\neg p_{\alpha_1} \vee \cdots \vee \neg p_{\alpha_n} \vee p_\alpha$. In addition, for each axiom $\alpha \in \mathcal{O}$ that

appears in I, we introduce a (unit) clause p_α. Finally, we add the clause $\neg p_{\alpha_g}$ encoding the assumption that α_g is not derivable. It is easy to see that $\mathcal{O} \vdash_{\mathsf{I}} \alpha_g$ if and only if the resulting set of clauses is unsatisfiable.

We now extend this reduction to find justifications for $\mathcal{O} \vdash_{\mathsf{I}} \alpha_g$. Recall that a subset $\mathcal{O}' \subseteq \mathcal{O}$ is a support for $\mathcal{O} \vdash_{\mathsf{I}} \alpha_g$ if $\mathcal{O}' \vdash_{\mathsf{I}} \alpha_g$. Hence, the subset of clauses p_α for $\alpha \in \mathcal{O}'$ is unsatisfiable in combination with the clauses for the encoding of inferences and $\neg p_{\alpha_g}$. We can find all such minimal subsets (corresponding to justifications) by adding a fresh answer literal to every clause p_α with $\alpha \in \mathcal{O}$, and applying resolution on non-answer literals together with elimination of redundant clauses.

Example 4. Consider the ontology $\mathcal{O}_e$, inferences I_e and axiom α_e from Example 1. To encode the derivability problem $\mathcal{O}_e \vdash_{\mathsf{I}_e} \alpha_e$ we assign atoms p_1–p_4 to the axioms occurring in I_e as follows:

$$p_1 : A \sqsubseteq B \sqcap C, \quad p_2 : A \sqsubseteq B, \quad p_3 : A \sqsubseteq C, \quad p_4 : A \sqsubseteq C \sqcap B.$$

The encoding produces clauses 1-7 from Example 3: the inferences I_e are encoded by clauses 1-3, the axioms in $\mathcal{O}_e$ result in clauses 4-6 with answer literals, and the assumption that α_e is not derivable is encoded by clause 7. The derived clauses 16-19 correspond to supports of $\mathcal{O}_e \vdash_{\mathsf{I}_e} \alpha_e$, and by eliminating redundant clauses 17-18, we obtain clauses 16 and 19 that correspond to justifications $\mathcal{O}'_e$ and $\mathcal{O}''_e$ from Example 1.

One disadvantage of the described procedure is that it requires the closure under the resolution rules to be fully computed before any justification can be found. Indeed, since derived clauses may be subsumed by later clauses, one cannot immediately see whether a clause with only answer literals corresponds to a justification. For example, clause 19 in Example 3 subsumes clauses 17-18 derived before, thus 17-18 do not correspond to justifications. We address this problem by using *non-chronological* application of resolution inferences. Intuitively, instead of applying the rules to clauses in the order in which they are derived, we apply the rules to clauses containing fewer answer literals first. Thus, in Example 3, we apply the rules to clause 15 before clauses 12-14.

The improved procedure can *enumerate* justifications, i.e., return justifications one by one without waiting for the algorithm to terminate. This procedure is described in Algorithm 1. It is a minor variation of the standard saturation-based procedure for computing the closure under (resolution) rules, which uses a *priority queue* to store unprocessed clauses instead of an ordinary queue. Let $\precsim$ be a total preorder on clauses (a transitive reflexive relation for which every two clauses are comparable). As usual, we write $c_1 \prec c_2$ if $c_1 \precsim c_2$ but $c_2 \not\precsim c_1$. We say that $\precsim$ is *admissible* if $c_1 \prec c_2$ whenever the set of answer literals of c_1 is a proper subset of the set of answer literals of c_2. For example, it is required that $\neg p_3 \vee p_4 \vee a_1 \prec p_4 \vee a_1 \vee a_2$, but not necessary that $p_4 \vee a_1 \prec p_4 \vee a_2 \vee a_3$. Note that if c is derived by resolution from clauses c_1 and c_2 then $c_1 \precsim c$ and $c_2 \precsim c$ since c contains the answer literals of both c_1 and c_2.

We say that a clause d (not necessarily occurring in Q) is *minimal* w.r.t. Q if there exists no clause $c \in \mathsf{Q}$ such that $c \prec d$. A *priority queue* based on $\precsim$ is a queue in which the remove operation returns only a *minimal* element w.r.t. Q.[5] Given such a

[5] If there are several minimal elements in the queue, one of them is chosen arbitrarily.

Algorithm 1: Enumeration of justifications using resolution

Enumerate($\mathcal{O} \vdash_{\mathsf{I}} \alpha$, $\precsim$): enumerate justifications for $\mathcal{O} \vdash_{\mathsf{I}} \alpha$
input : $\mathcal{O} \vdash_{\mathsf{I}} \alpha$ – the problem for which to enumerate justifications,
$\precsim$ – an admissible preorder on clauses

```
1  Q ← createEmptyQueue(≾) ; // for unprocessed clauses
2  Q.addAll(encode(O ⊢I α)); // add the clause encoding of the problem
3  S ← createEmptyList() ; // for processed clauses
4  while Q ≠ ∅ do
5      c ← Q.remove(); // take one minimal element out of the queue
6      c ← simplify(c); // remove duplicate literals from c
7      if c is not subsumed by any c' ∈ S then
8          S.add(c);
9          if c contains only answer literals then
10             report decode(c); // a new justification is found
11         else // apply resolution rules to c and clauses in S
12             for c' ∈ resolve(c,S) do
13                 Q.add(c');
```

queue Q, Algorithm 1 initializes it with the translation of the input problem $\mathcal{O} \vdash_{\mathsf{I}} \alpha$ (line 2) and then repeatedly applies resolution between minimal clauses taken out of this queue (loop 4-13) and the clauses in S that were processed before. Specifically, the removed minimal clause c is first simplified by removing duplicate literals (line 6) and then checked if it is subsumed by any previously processed clauses in S (in particular, if c was processed before). If c is subsumed by some $c' \in \mathsf{S}$, it is ignored and the next (minimal) clause is taken from the queue Q. Otherwise, c is added to S (line 8). If c contains only answer literals, then it corresponds to a justification (as we show next), which is then reported by the algorithm (line 10). Otherwise, resolution inferences are then applied on the selected non-answer literal in c (line 12). The new clauses derived by resolution are then added to Q (line 13) and the loop continues until Q is empty.

We now prove that Algorithm 1 in line 10 always returns a (new) justification. It is easy to see that if a clause d was minimal w.r.t. Q in the beginning of the while loop (line 4) then it remains minimal w.r.t. Q at the end of the loop (line 13). Indeed, for the clause c taken from the queue (line 5), we have $c \not\prec d$. For all clauses c' obtained by resolving c with clauses from S (line 12) we have $c \precsim c'$. Hence $c' \not\prec d$ for all c' added to Q (line 13) (for otherwise, $c \precsim c' \prec d$). This, in particular, implies that each clause in S is always minimal w.r.t. Q and, consequently, if c_1 was added to S before c_2 then $c_1 \precsim c_2$ (for otherwise $c_2 \prec c_1$ and c_1 would not be minimal w.r.t. Q when $c_2 \in \mathsf{Q}$). Hence, there cannot be two clauses c_1 and c_2 in S that contain only answer literals such that c_1 is a proper sub-clause of c_2 since in this case $c_1 \prec c_2$, thus c_2 must have been added to S after c_1, but then c_2 would be subsumed by c_1 (see line 7). Hence each result returned in line 10 is a (new) justification.

Since clauses are added to S in the order defined by $\precsim$, the justifications are also returned according to this order. Hence Algorithm 1 can return justifications in any

user-defined order $\precsim$ on subsets of axioms as long as $s_1 \subsetneq s_2$ implies $s_1 \prec s_2$. Indeed, any such an order $\precsim$ can be lifted to an admissible order on clauses by comparing the sets of answer literals of clauses like the corresponding sets of axioms. For example, one can define $s_1 \precsim s_2$ by $\|s_1\| \leq \|s_2\|$ where $\|s\|$ is the cardinality of s. Instead of $\|s\|$ one can use any other measure $m(s)$ that is monotonic over the proper subset relation (i.e., $s_1 \subsetneq s_2$ implies $m(s_1) < m(s_2)$), for example, the *length* of s—the total number of symbols needed to write down all axioms in s.

5 Implementation and Evaluation

We have implemented Algorithm 1 as a part of the new Java-based Proof Utility Library (PULi).[6] In our implementation, we used the standard Java priority queue for Q, and employed a few optimisations to improve the performance of the algorithm.

First, we have noticed that our implementation spends over 95% of time on checking subsumptions in line 7. To improve subsumption checks, we developed a new datastructure for storing sets of elements and checking if a given set is a superset of some stored set. In a nutshell, we index the sets by 128 *bit vectors*, represented as a pair of 64 bit integers, where each set element assigns 1 to one position of the bit vector based on its hash value. This idea is reminiscent of Bloom filters[7] and can be also seen as a simple version of a feature vector indexing [24]. We store the sets in a trie[8] with the bit vector as the key, and use bitwise operations to determine if one vector has all bits of the other vector, which gives us a necessary condition for set inclusion. Using this datastructure, we were able to significantly improve the subsumption tests.

We have also noticed that the queue Q often contains about 10 times more elements than the closure S. To improve the memory consumption, we do not create the resolvents c' immediately (see line 12), but instead store in the queue Q the pairs of clauses (from S) from which these resolvents were obtained. This does not reduce the number of elements in the queue, but reduces the memory consumed by each element to essentially a few pointers plus an integer for determining the priority of the element.

We have evaluated our implementation on inferences computed for entailed axioms in some large $\mathcal{EL}$ ontologies, and compared performance with SAT-based tools for enumeration of justifications EL2MUS [3], EL2MCS [2] and SATPin [8]. The inferences were extracted using EL+SAT [11] (in the following called sat inferences) and ELK reasoner [25] (in the following called elk inferences). Both are capable of computing small inference sets that derive particular entailed axioms and are complete for these entailments (see Section 3.2).

For our evaluation, we chose ontologies GO-PLUS, GALEN and SNOMED, which contain (mostly) $\mathcal{EL}$ axioms. GO-PLUS is a recent version of Gene Ontology,[9] which imports a number of other ontologies. The provided distribution included subsumption axioms that were inferred (annotated with `is_inferred`), which we have removed.

[6] `https://github.com/liveontologies/puli`
[7] `https://en.wikipedia.org/wiki/Bloom_filter`
[8] `https://en.wikipedia.org/wiki/Trie`
[9] `http://geneontology.org/page/download-ontology`

Table 2. Summary of the input ontologies

	GO-PLUS	GALEN	SNOMED
# axioms	105557	44475	315521
# concepts	57173	28482	315510
# roles	157	964	77
# queries	90443	91332	468478

Table 3. Summary of sizes of inference sets

		GO-PLUS	GALEN	SNOMED
sat	average	470.3	59140.0	997.8
	median	39.0	110290.0	1.0
	max	15915.0	152802.0	39381.0
elk	average	166.9	3602.0	110.3
	median	43.0	3648.0	8.0
	max	7919.0	81501.0	1958.0

GALEN is the version 7 of OpenGALEN.[10] We did not use the more recent version 8, because the other tools were running out of memory. SNOMED is the 2015-01-31 version of Snomed CT.[11] From the first two ontologies we removed non-$\mathcal{EL}$ axioms, such as functional property axioms, and axioms that contain inverse property expressions and disjunctions. We have also adapted the input ontologies, so that they could be processed by (the reasoner of) EL+SAT. We removed disjointness axioms and replaced property equivalences with pairs of property inclusions. Duplicate axioms were removed by loading and saving the ontologies with OWL API.[12] With these ontologies, we have computed justifications for the entailed direct subsumptions between atomic concepts (in the following called *the queries*) using various tools. Table 2 shows the numbers of axioms, atomic concepts, atomic roles, and queries of each input ontology, and Table 3 the statistics about the sizes of inference sets obtained for these queries. All queries were processed by tools in a fixed random order to achieve a fair distribution of easy and hard problems. We used a *global timeout* of one hour for each tool and a *local timeout* of one minute per query.[13] To run the experiments we used a PC with Intel Core i5 2.5 GHz processor and 8 GiB RAM operated under 64-bit OS Ubuntu 16.04. For Java tools, we used OpenJDK v. 1.80_151 with a 7.7 GiB heap space limit.

As an admissible order on clauses for our implementation of Algorithm 1, we chose the relation $\precsim$ that compares the number of different answer literals in clauses. When using this order, cardinality-minimal justifications are found first. To control resolution inferences, we used three different selection strategies (for Horn clauses) that we detail next. For a propositional atom p, let $\#(p)$ be the number of input clauses in which p appears as a (positive) literal. Given a clause c, the *BottomUp* strategy, selects a negative literal $\neg p$ of c whose value $\#(p)$ is minimal; if there are no negative literals, the (only) positive literal of c is selected. The *TopDown* strategy selects a positive literal, if there is one, and otherwise selects a negative literal like in BottomUp. Finally, the *Threshold* strategy selects a negative literal $\neg p$ with the minimal value $\#(p)$ if $\#(p)$ does not exceed a given threshold value or there is no positive literal in c; otherwise the positive literal is selected. In our experiments we used the threshold value of 2. In-

[10] `http://www.opengalen.org/sources/sources.html`

[11] `http://www.snomed.org/`

[12] `http://owlcs.github.io/owlapi/`

[13] The project for conducting the experiments can be found at `https://github.com/liveontologies/pinpointing-experiments`; a docker image is available at `https://github.com/liveontologies/docker-pinpointing-experiments`

Table 4. Number of queries attempted in 1h / number of 60s timeouts / % of attempted queries in the number of all queries / % of 60s timeouts in the number of queries attempted in 1h

		GO-PLUS	GALEN	SNOMED
sat	BottomUp	2967 / 47 / 3.3 / 1.58	133 / 58 / 0.1 / 43.6	5630 / 26 / 1.2 / 0.46
	TopDown	25025 / 46 / 27.2 / 0.18	5687 / 16 / 6.2 / 0.28	16541 / 28 / 3.5 / 0.17
	Threshold	36236 / 43 / 40.1 / 0.12	3356 / 3 / 3.7 / 0.09	48994 / 16 / 10.5 / 0.03
	EL2MUS	12760 / 48 / 14.1 / 0.38	5077 / 34 / 5.6 / 0.67	14076 / 39 / 3.0 / 0.28
	EL2MCS	6758 / 47 / 7.5 / 0.70	3194 / 27 / 3.5 / 0.85	6275 / 47 / 1.3 / 0.75
	SATPin	4390 / 53 / 4.9 / 1.21	1475 / 39 / 1.6 / 2.64	3490 / 46 / 0.7 / 1.32
elk	BottomUp	3694 / 47 / 4.1 / 1.27	10584 / 28 / 11.6 / 0.26	159820 / 22 / 34.1 / 0.01
	TopDown	20249 / 44 / 22.4 / 0.22	7016 / 35 / 7.7 / 0.50	158992 / 12 / 33.9 / 0.01
	Threshold	35622 / 52 / 39.4 / 0.15	35462 / 16 / 38.8 / 0.05	468478 / 0 / 100 / 0.00
	EL2MUS	13554 / 47 / 15.0 / 0.35	13024 / 38 / 14.3 / 0.29	15708 / 42 / 3.4 / 0.27
	EL2MCS	6758 / 47 / 7.5 / 0.70	8725 / 47 / 9.6 / 0.54	6466 / 48 / 1.4 / 0.74
	SATPin	4625 / 54 / 5.1 / 1.17	3037 / 49 / 3.3 / 1.61	4144 / 50 / 0.9 / 1.21

tuitively, the BottomUp strategy simulates the *Unit resolution*, the TopDown simulates the *SLD resolution*, and Threshold is some combination thereof.

Table 4 shows for how many queries all justifications were computed within the global and local timeouts.[14] The first six rows correspond to experiments on sat inferences and the other six rows to experiments on elk inferences. Note that, generally, the tools processed more queries and had fewer percentage of timeouts for elk inferences. Also, the Threshold strategy performed best in almost all cases. In particular, it could process all queries of SNOMED without timeouts. None of the SAT-based tools was able to find all justifications for all queries of SNOMED even after running for 24 hours. We have then further verified (on a slightly faster PC with more memory) that Threshold without timeouts could process all 5415670 (not necessarily direct) entailed subsumptions of SNOMED in about 21 hours using 10 GiB of java heap space. The hardest query took about 17 minutes and returned 658932 justifications. The largest number of justifications 942658 was returned by the third-hardest query in about 5 minutes.

To have an idea which strategy was best for which query, we have plotted in Figure 3 the distributions of the query times for all strategies. Each point $\langle x, y\rangle$ of a plot represents the proportion x of queries that were solved by the method in under the time y. For instance, TopDown solved about 90% of the queries of GO-PLUS for sat inferences in under 0.01 second. Each plot considers only queries attempted by all tools on that plot. Since each plot represents the distribution of times and not a direct comparison of times for each query, even if one line is completely below another one, this does not mean that the corresponding method is faster for *every* query. To get a more detailed comparison, we have also plotted the distribution of minimum query times with a thin black line. For each query, the *minimum time* is the time spent by the tool that was the fastest on that query (among the tools on the same plot). If a plot for some tool coincides with this black line at point (x, y), then all queries solved within time y by some tool were also solved within time y by this tool. In particular, this tool is the fastest for all queries with

[14] The raw experimental data is available at `https://osf.io/4q6a9/`

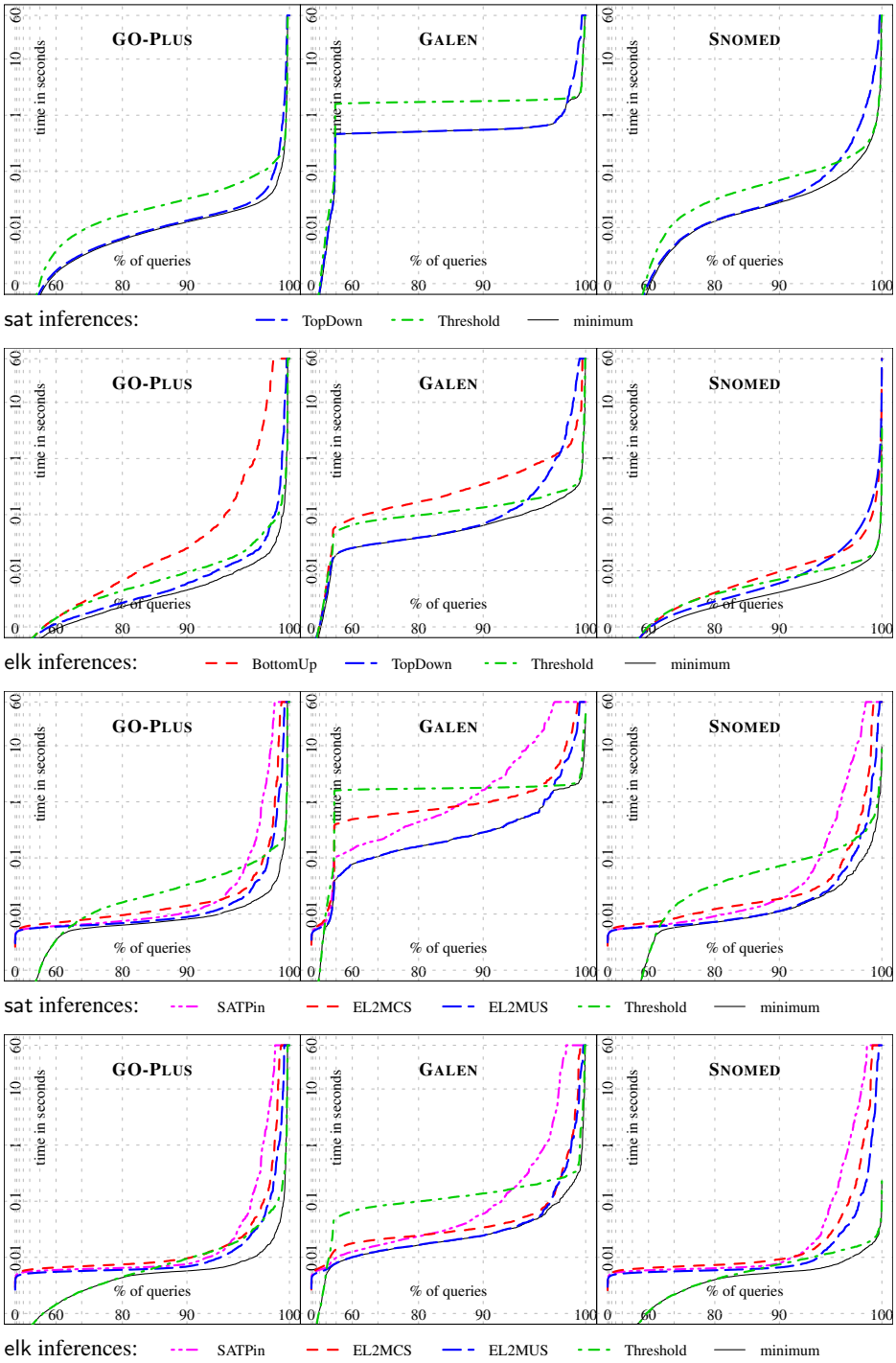

Fig. 3. Distribution of query times for SAT tools and resolution strategies, on sat and elk inference sets. The BottomUp strategy for sat is missing due too few processed queries.

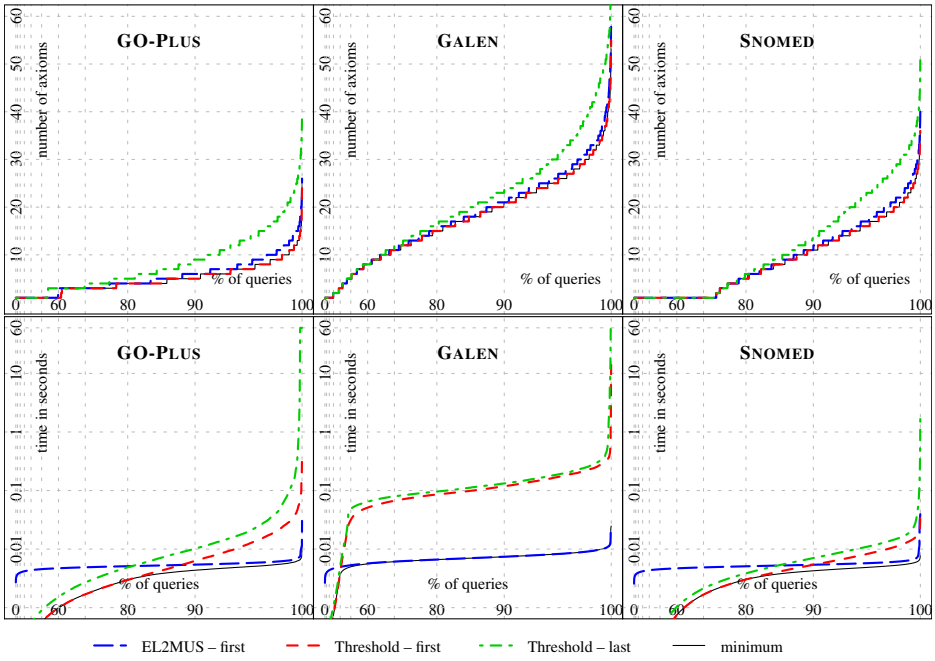

Fig. 4. Distribution of sizes (above) and computation times (below) for first and last justifications computed by Threshold and first justifications computed by EL2MUS on elk inferences

the minimal time y. This analysis shows, for example, that TopDown was often the best resolution tool for easy queries (solved under 0.1 seconds by some tool), and Threshold was the best tool for hard queries (solved over 1 second by all tools) on all ontologies. The sat-based tool EL2MUS was often the winner on medium-hard queries. Note that the time scale is logarithmic, so the times below 1 millisecond are not displayed.

As mentioned before, one important difference between SAT tools and resolution-based tools, is that the latter allow one to enumerate justifications in any admissible order. In particular, it is possible to find cardinality-minimal justifications without computing all justifications, which is useful, e.g., for ontology debugging. Since deciding whether there exists a justification within a given size bound is an NP-complete problem [4], finding a cardinality-minimal justification is not as easy as finding one (arbitrary) justification. To determine whether the difference is significant in practice, we compare computations of the first justification by EL2MUS, the first (cardinality-minimal) justification by Threshold, and the last (before timeout, cardinality-maximal) justification by Threshold. In Figure 4, we plot distributions of the sizes for these justifications and of times spent on computing them. Interestingly, the sizes of first justifications found by EL2MUS are very close to the minimal sizes. This is probably because small justifications are more likely to be obtained when minimizing sets of axioms for which the entailment holds. Unsurprisingly, EL2MUS can consistently compute the fist justifications in a few milliseconds. Although the times for computing a cardinality-minimal

justification can be significantly higher (especially for GALEN, which has large inference sets), they can still be a few orders of magnitude smaller than for computing all (= the last) justifications for the hard cases. In particular, Threshold was able to compute a cardinality-minimal justification for all queries of GO-PLUS, GALEN and SNOMED respectively in about 6 minutes, 1.5 hours, and 25 minutes.

6 Summary

We presented a new procedure that enumerates justifications using inferences that derive the goal consequence from an ontology. The inferences are encoded as Horn clauses and resolution with answer literals is applied. Our procedure can be parameterized by an ordering in which the justifications should be enumerated (as long as it extends the subset relation) and by a strategy that selects literals for resolution. The algorithm is relatively easy to implement and can be also easily used with non-Horn and non-propositional clauses. Our empirical evaluation shows that the procedure provides comparable, if not better performance than other tools that also use inferences as input. For example, for Snomed CT we were able to compute all justifications for all direct subsumptions in less than 1 hour, and for all (possibly indirect) subsumptions in less than 1 day. Currently, we cannot explain the difference in the performance of the evaluated selection strategies. We hope to explore this question in the future.

References

1. Arif, M.F., Mencía, C., Ignatiev, A., Manthey, N., Peñaloza, R., Marques-Silva, J.: BEACON: an efficient sat-based tool for debugging $\mathcal{EL}^+$ ontologies. In Creignou, N., Berre, D.L., eds.: Proc. 19th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'16). Volume 9710 of LNCS., Springer (2016) 521–530
2. Arif, M.F., Mencía, C., Marques-Silva, J.: Efficient axiom pinpointing with EL2MCS. In Hölldobler, S., Krötzsch, M., Peñaloza, R., Rudolph, S., eds.: Proc. 38th Annual German Conf. on Artificial Intelligence (KI'15). Volume 9324 of LNCS., Springer (2015) 225–233
3. Arif, M.F., Mencía, C., Marques-Silva, J.: Efficient MUS enumeration of horn formulae with applications to axiom pinpointing. CoRR **abs/1505.04365** (2015)
4. Baader, F., Peñaloza, R., Suntisrivaraporn, B.: Pinpointing in the description logic $\mathcal{EL}^+$. In Hertzberg, J., Beetz, M., Englert, R., eds.: Proc. 30th Annual GermanConf. on Artificial Intelligence (KI'07). Volume 4667 of LNCS., Springer (2007) 52–67
5. Horridge, M.: Justification based explanation in ontologies. PhD thesis, University of Manchester, UK (2011)
6. Junker, U.: QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems. In McGuinness, D.L., Ferguson, G., eds.: Proc. 19th AAAI Conf. on Artificial Intelligence (AAAI'04), AAAI Press / The MIT Press (2004) 167–172
7. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In Aberer, K., Choi, K.S., Noy, N., Allemang, D., Lee, K.I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P., eds.: Proc. 6th Int. Semantic Web Conf. (ISWC'07). Volume 4825 of LNCS., Springer (2007) 267–280
8. Manthey, N., Peñaloza, R., Rudolph, S.: Efficient axiom pinpointing in $\mathcal{EL}$ using SAT technology. In Lenzerini, M., Peñaloza, R., eds.: Proc. 29th Int. Workshop on Description Logics (DL'16). Volume 1577 of CEUR Workshop Proceedings., CEUR-WS.org (2016)

9. Parsia, B., Sirin, E., Kalyanpur, A.: Debugging OWL ontologies. In Ellis, A., Hagino, T., eds.: Proc. 14th Int. Conf. on World Wide Web (WWW'05), ACM (2005) 633–640
10. Kalyanpur, A.: Debugging and Repair of OWL Ontologies. PhD thesis, University of Maryland College Park, USA (2006)
11. Sebastiani, R., Vescovi, M.: Axiom pinpointing in lightweight description logics via Horn-SAT encoding and conflict analysis. In Schmidt, R.A., ed.: Proc. 22st Conf. on Automated Deduction (CADE'09). Volume 5663 of LNCS., Springer (2009) 84–99
12. Vescovi, M.: Exploiting SAT and SMT Techniques for Automated Reasoning and Ontology Manipulation in Description Logics. PhD thesis, University of Trento, Italy (2011)
13. Baader, F., Lutz, C., Suntisrivaraporn, B.: Efficient reasoning in $\mathcal{EL}^+$. In Parsia, B., Sattler, U., Toman, D., eds.: Proc. 19th Int. Workshop on Description Logics (DL'06). Volume 189 of CEUR Workshop Proceedings., CEUR-WS.org (2006)
14. Kazakov, Y., Krötzsch, M., Simančík, F.: The incredible ELK: From polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. J. of Automated Reasoning **53**(1) (2014) 1–61
15. Kazakov, Y.: Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In Boutilier, C., ed.: Proc. 21st Int. Joint Conf. on Artificial Intelligence (IJCAI'09), IJCAI (2009) 2040–2045
16. Simančík, F., Kazakov, Y., Horrocks, I.: Consequence-based reasoning beyond Horn ontologies. In Walsh, T., ed.: Proc. 22nd Int. Joint Conf. on Artificial Intelligence (IJCAI'11), AAAI Press/IJCAI (2011) 1093–1098
17. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In Kaelbling, L., Saffiotti, A., eds.: Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05), Professional Book Center (2005) 364–369
18. Greiner, R., Smith, B.A., Wilkerson, R.W.: A correction to the algorithm in reiter's theory of diagnosis. Artif. Intell. **41**(1) (1989) 79–88
19. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1) (1987) 57–95
20. Morgado, A., Liffiton, M.H., Marques-Silva, J.: Maxsat-based MCS enumeration. In Biere, A., Nahir, A., Vos, T.E.J., eds.: Proc. 8th Int. Haifa Verification Conf. (HVC'12). Volume 7857 of LNCS., Springer (2012) 86–101
21. Ansótegui, C., Bonet, M.L., Levy, J.: Sat-based maxsat algorithms. Artif. Intell. **196** (2013) 77–105
22. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In Robinson, J.A., Voronkov, A., eds.: Handbook of Automated Reasoning. Elsevier and MIT Press (2001) 19–99
23. Green, C.: Theorem proving by resolution as a basis for question-answering systems. Machine Intelligence (1969) 183–205
24. Schulz, S.: Simple and efficient clause subsumption with feature vector indexing. In Bonacina, M.P., Stickel, M.E., eds.: Automated Reasoning and Mathematics - Essays in Memory of William W. McCune. Volume 7788 of LNCS., Springer (2013) 45–67
25. Kazakov, Y., Klinov, P.: Goal-directed tracing of inferences in $\mathcal{EL}$ ontologies. In: The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II. (2014) 196–211