Enumerating Justifications using Resolution

Yevgeny Kazakov and Peter Skočovský

The University of Ulm, Germany
{yevgeny.kazakov, peter.skocovsky}@uni-ulm.de

Abstract. We propose a new procedure that can enumerate justifications of a logical entailment given a set of inferences using which this entailment can be derived from axioms in the ontology. The procedure is based on the extension of the resolution method with so-called answer literals. In comparison to other (SAT-based) methods for enumerating justifications, our procedure can enumerate justifications in any user-defined order that extends the subset relation. The procedure is easy to implement and can be parametrized with ordering and selection strategies used in resolution. We describe an implementation of the procedure provided in PULi—a new Java-based Proof Utility Library, and provide an empirical comparison of (several strategies of) our procedure and other SAT-based tools on popular $\mathcal{EL}$ ontologies. The experiments show that our procedure provides a comparable, if not better performance than those highly optimized tools. For example, using one of the strategies, we were able to compute all justifications for all direct subsumptions of Snomed CT in about 1.5 hour. No other tool used in our experiments was able to do it even within a much longer period.

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 [2–4, 10, 11, 17, 19, 23, 21]. 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 inferences 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 [5, 16, 14, 22]. 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 defines basic notions such as inferences 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 [19] using a 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 some additional information provided from a reasoner, such as inferences that the reasoner has used, and thus can only work with such reasoners.

In a nutshell, Black-Box procedures [6, 11–13] add or remove axioms from a set of axioms (a justification candidate) while maintaining the invariant that the set entails the given *goal* logical consequence. An external reasoner is used for checking this invariant. Finding *one* justification is relatively easy and can be done by trying to remove each axiom once and check if the result still entails the goal. If it does not, the removed axiom is added back. Provided the entailment check takes polynomial time in the number of all axioms, e.g., in the case of $\mathcal{EL}$, this procedure runs in polynomial time. However, finding all justifications is hard also in the case of $\mathcal{EL}$, because there may be exponentially many of them [6]. Should a procedure for finding one justification be extended to a procedure for justification enumeration, the main problem is how to prevent repetition of justifications. Most techniques rely on the hitting set duality that was introduced in the field of Model Based Diagnosis [9, 20] and later adapted for DLs [10, 13]. Having a collection of sets, its *hitting set* contains some element from each of them. A minimal hitting set of a set of all justifications of some goal is its *repair*—when removed from the ontology, the goal is no longer entailed. Having a number of justifications of some goal, we can find a new one by removing one of their hitting sets from the ontology and applying the procedure for finding one justification.

Many Glass-Box procedures focus on $\mathcal{EL}$, because $\mathcal{EL}$ reasoning is relatively simple and efficient. Some of them employ propositional satisfiability (SAT) solvers. Their input are inferences recorded by the reasoner during classification of an ontology. The inferences are used to reduce entailment to satisfiability of Horn clauses. This approach was first used by EL+SAT [21, 23], that includes an $\mathcal{EL}$ reasoner which records the inferences plus a few optimizations for reducing the set of inferences relevant to the goal. A common feature of the SAT-based procedures is enumeration of the candidate axiom sets exploiting the hitting set duality and further minimization of these candidates. EL+SAT and EL2MUS [4] use two instances of a SAT-solver—one for enumeration of candidates and the other one for checking whether a candidate entails the goal. SATPin [17] uses one SAT-solver for both of these tasks and encodes a candidate as the set of axiom atoms currently assigned true. Enumeration of the same candidate twice is avoided by adding a blocking clause to the enumerating solver. If a candidate contains a justification, its blocking clause consists of negated atoms that encode axioms from that justification. EL+SAT and SATPin block candidates that do not entail the goal in the same way as justifications. EL2MUS does it in a different way. When its specialized enumeration solver finds a non-entailing candidate, its complement is automatically a

repair. A repair is blocked by adding a clause consisting of positive atoms of its axioms. Further differences are that in EL2MUS the entailment checking solver is specialized for Horn clauses and that EL+SAT and SATPin extract justifications by a deletion-based procedure, while EL2MUS uses more efficient insertion-based procedure. EL2MCS [3] uses MaxSAT [1, 18] to enumerate all repairs and extracts justifications from them using the hitting set duality. BEACON [2] is a tool that integrates the justification procedure of EL2MUS.

Up to few optimizations, the mentioned SAT-based procedures use inferences only for the entailment check. Had they delegated the entailment check to a separate DL reasoner, they could be considered Black-Box. Our approach uses a similar encoding of derivability check, however, it does not rely on the hitting set duality and, instead, manipulates the inferences directly.

3 Preliminaries

3.1 Inferences, Supports, and Justifications

In this section, we introduce a general type of inferences that can manipulate any types of objects, that we call *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*. An *ontology* is a finite set of axioms.

Let I be an inference set. 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 $\mathcal{O}' \subseteq \mathcal{O}$ such that $\mathcal{O}' \vdash_{\mathsf{I}} \alpha$. A subset-minimal support for $\mathcal{O} \vdash_{\mathsf{I}} \alpha$ is called a *justification* for $\mathcal{O} \vdash_{\mathsf{I}} \alpha$.

Suppose that $\models$ is an entailment relation between ontologies and axioms. A *support* for $\mathcal{O} \models \alpha$ is a subset of $\mathcal{O}' \subseteq \mathcal{O}$ such that $\mathcal{O}' \models \alpha$. A subset-minimal support for $\mathcal{O} \models \alpha$ is called a *justification* for $\mathcal{O} \models \alpha$ (also called a *minimal axiom set* MinA [6]). 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 $\mathcal{O}' \subseteq \mathcal{O}$. Note that if I is a set of sound inferences that is complete for the entailment $\mathcal{O} \models \alpha$ then $\mathcal{O}' \models \alpha$ iff $\mathcal{O}' \vdash_{\mathsf{I}} \alpha$ for every $\mathcal{O}' \subseteq \mathcal{O}$. In particular, supports and justifications for $\mathcal{O} \vdash_{\mathsf{I}} \alpha$ coincide with supports and, respectively, justifications for $\mathcal{O} \models \alpha$.

Example 1. Consider the ontology $\mathcal{O}_e = \{A \sqsubseteq B \sqcap C; A \sqsubseteq B; A \sqsubseteq C\}$ over DL axioms and the axiom $\alpha_e = A \sqsubseteq C \sqcap B$. Assume that some consequence-based procedure performed inferences $\mathsf{I}_e = \{\langle A \sqsubseteq B \sqcap C \vdash A \sqsubseteq B \rangle, \langle A \sqsubseteq B \sqcap C \vdash A \sqsubseteq C \rangle, \langle A \sqsubseteq C, A \sqsubseteq B \vdash A \sqsubseteq C \sqcap B \rangle\}$ in order to derive α_e from $\mathcal{O}$. 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 standard 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 $\mathcal{O}_e \models \alpha_e$.

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

Fig. 1. Propositional resolution and factoring rules

3.2 Resolution with Answer Literals

In this section we introduce the *resolution calculus*, which is a popular method for automated theorem proving [7]. We will mainly use resolution for propositional Horn clauses. A (propositional) *atom* is a propositional variable a. 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 and all literals from c_2. A clause is *Horn* if it has at most one positive literal. The *empty clause* $\Box$ is the clause containing $n = 0$ literals. The inference rules for the propositional resolution calculus are given in Figure 1. We say that a set of clauses S is *closed* under the resolution rules, if every clause derived by the rules in Figure 1 from S already occurs in S. The resolution calculus is *refutationally complete*: a set of clauses S that is 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 of clauses 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 1 can be restricted using orderings and selection functions [7]. 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 [7]).[1] 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 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: $\Box$	$\langle 7, 9 \rangle$

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

[1] 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.

Although the resolution calculus is mainly used for checking satisfiability of a clause set, it can be also used for finding unsatisfiable subsets of clauses. To do this, it is sufficient to add to every input clause a fresh positive *answer literal* [8]. Resolution rules can then be applied to the extended clauses on the remaining (ordinary) literals using the usual orderings and selection functions. Thus, if a 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 inferences 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: $\boxed{\underline{a_1}}$	$\langle 7, 15\rangle$
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 in combination with the resolution procedure with answer literals.

Given the derivability problem $\mathcal{O} \vdash_{\mathsf{I}} \alpha_g$, we assign to each axiom α_i occurring in I a fresh 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 or not. For example, clauses 17-18 in Example 3 are subsumed by clause 19 that is derived later, and hence do not correspond to justifications. We address this problem by using *non-chronological* application of resolution inferences. Intuitively, instead of processing clauses in the order in which they are derived, we process clauses containing fewer answer literals first. Thus, in Example 3, we process 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. The 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 can take out only *minimal* elements w.r.t. Q.[2] Given such a queue Q, Algorithm 1 initializes it with the translation of the input problem $\mathcal{O} \vdash_{\mathsf{I}} \alpha$ (line 3)

[2] 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  S ← createEmptyList() ; // for processed clauses
2  Q ← createEmptyQueue(≾) ; // for unprocessed clauses
3  Q.addAll(encode(O ⊢I α)); // add the clause encoding of the problem
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');
```

and then repeatedly applies resolution inferences 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 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.

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 be 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 $m(s_1) < m(s_2)$), e.g., 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).[3] 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 element in a set sets one bit of the bit vector to 1 using its hash value. This idea is reminiscent of Bloom filters.[4] We store the sets in a trie[5] 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.

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 from inferences EL2MUS [4], EL2MCS [3] and SAT-Pin [17]. The inferences were extracted by EL+SAT [21, 23] (in the following called sat inferences) and ELK [15] (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 3.1). See Table 2 for statistics about the inferences obtained for the entailments.

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,[6] which imports a number of other ontologies. The provided distribution included subsumption axioms that were inferred (annotated with `is_inferred`), which we have removed. GALEN is the version 7 of OpenGALEN.[7] We did not use the more recent version 8,

[3] `https://github.com/liveontologies/puli`
[4] `https://en.wikipedia.org/wiki/Bloom_filter`
[5] `https://en.wikipedia.org/wiki/Trie`
[6] `http://geneontology.org/page/download-ontology`
[7] `http://www.opengalen.org/sources/sources.html`

Table 1. 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 2. 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

because the other tools were running out of memory. SNOMED is the version of Snomed CT[8] released on 2015-01-31. 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. With these ontologies, we have computed justifications for the entailed direct subsumptions between atomic concepts (in the following called *the queries*) using various tools. 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. To run the experiments we used a PC with Intel Core i5 2.5 GHz processor and 7.7 GiB RAM operated under 64-bit OS Ubuntu 14.04. Table 1 shows the numbers of axioms, atomic concepts, atomic roles, and queries of each input ontology.

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. Note that finding such justifications is NP-hard [6], however, practically, our algorithm with this order found first justifications of all the queries of GO-PLUS, GALEN and SNOMED respectively in about 13 minutes, 2 hours and 1.5 hours. 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. Intuitively, the BottomUp strategy simulates the *Unit resolution*, the TopDown simulates the *SLD resolution*, and the Threshold is a combination thereof.

Table 3 shows for how many queries all justifications were computed within the global and local timeouts.[9] The first six rows correspond to experiments on sat infer-

[8] http://www.snomed.org/

[9] All experimental data is available at https://osf.io/9sj8n/

Table 3. Number of queries processed in 1h / number of 60s timeouts / % of processed queries in the number of all queries / % of 60s timeouts in the number of queries attempted in 1h

		GO-PLUS	GALEN	SNOMED
sat	BottomUp	2004 / 43 / 2.2 / 2.15	123 / 56 / 0.1 / 45.5	3270 / 31 / 0.7 / 0.95
	TopDown	16662 / 48 / 18.4 / 0.29	5028 / 18 / 5.5 / 0.36	7388 / 30 / 1.6 / 0.41
	Threshold	23968 / 45 / 26.5 / 0.19	2873 / 3 / 3.1 / 0.10	18700 / 14 / 4.0 / 0.07
	EL2MUS	10357 / 42 / 11.5 / 0.41	4862 / 32 / 5.3 / 0.66	12773 / 37 / 2.7 / 0.29
	EL2MCS	6369 / 43 / 7.0 / 0.68	3194 / 27 / 3.5 / 0.85	6163 / 46 / 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	3113 / 41 / 3.4 / 1.32	8204 / 23 / 9.0 / 0.28	85977 / 20 / 18.4 / 0.02
	TopDown	18392 / 43 / 20.3 / 0.23	6757 / 33 / 7.4 / 0.49	73063 / 15 / 15.6 / 0.02
	Threshold	30579 / 47 / 33.8 / 0.15	25824 / 15 / 28.3 / 0.06	291151 / 1 / 62.1 / 0.00
	EL2MUS	12877 / 48 / 14.2 / 0.37	12349 / 40 / 13.5 / 0.32	15054 / 42 / 3.2 / 0.28
	EL2MCS	6694 / 49 / 7.4 / 0.73	8545 / 47 / 9.4 / 0.55	6466 / 48 / 1.4 / 0.74
	SATPin	4689 / 50 / 5.2 / 1.07	3050 / 48 / 3.3 / 1.57	4320 / 52 / 0.9 / 1.20

ences and the other six rows to experiments on elk inferences. Note that every tool processed more queries and had less relative timeouts on elk inferences. Notice that the resolution procedure with the Threshold selection strategy processed the most queries and had the least number of timeouts on all ontologies. In particular, for SNOMED it timed out only in one case. It turns out, that without any timeout Threshold was able to find all justifications for all queries of SNOMED within about 1.5 hours, with the longest query taking less than 4 minutes. None of the SAT-based tools was able to find all justifications for all queries of SNOMED even within 24 hours.

To determine whether the Threshold strategy was the best among the resolution strategies *on all queries*, we have plotted in Figure 2 the distributions of the query times for all resolution 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 GALEN in under 0.1 seconds. 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 a query, *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 the minimal time y. This analysis shows, for example, that TopDown was the best tool for easy queries (solved under 0.1 seconds by some tool) in GALEN, and Threshold was the best tool for hard queries (solved over 1 second by all tools) on all ontologies.

In Figure 3 we present a similar comparison of Threshold with the SAT-based tools. Threshold comes as the winner for hard queries (solved over 0.1 seconds) in

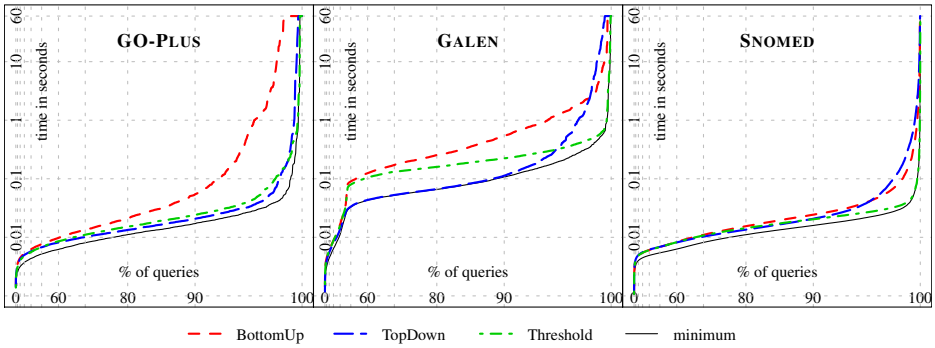

Fig. 2. Distribution of query times for the selection strategies on elk inferences

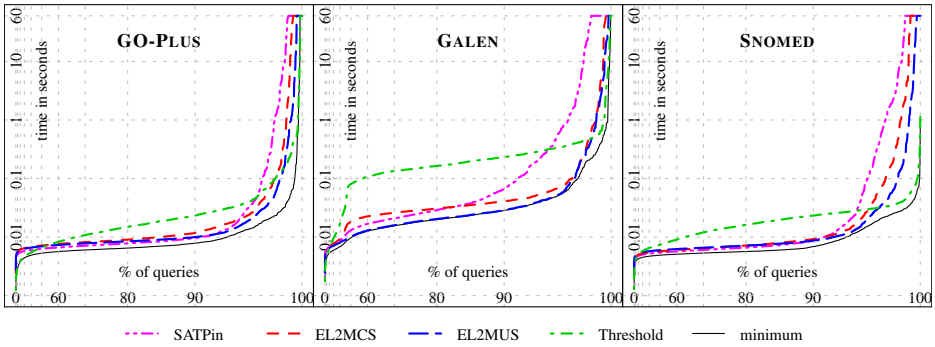

Fig. 3. Distribution of query times on elk inferences[10]

SNOMED.[11] EL2MUS wins on queries solved between about 10 and 50 milliseconds in GALEN. For all other query ranges, there appears to be no absolute winner.

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 it can be easily used also 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. We were able to compute all justifications for all direct subsumptions of Snomed CT in about 1.5 hours. Currently, we cannot explain the difference in the performance of the evaluated selection strategies. We hope to explore this question in the future.

[10] Plots on sat inferences have the same shape, except that all lines are shifted up.

[11] For other ontologies Threshold times out on some query for which some other tool does not.

References

1. Ansótegui, C., Bonet, M.L., Levy, J.: Sat-based maxsat algorithms. Artif. Intell. 196, 77–105 (2013), http://dx.doi.org/10.1016/j.artint.2013.01.002
2. 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). Lecture Notes in Computer Science, vol. 9710, pp. 521–530. Springer (2016), http://dx.doi.org/10.1007/978-3-319-40970-2_32
3. 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 GermanConf. on Artificial Intelligence (KI'15). Lecture Notes in Computer Science, vol. 9324, pp. 225–233. Springer (2015), http://dx.doi.org/10.1007/978-3-319-24489-1_17
4. Arif, M.F., Mencía, C., Marques-Silva, J.: Efficient MUS enumeration of horn formulae with applications to axiom pinpointing. CoRR abs/1505.04365 (2015), http://arxiv.org/abs/1505.04365
5. 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). CEUR Workshop Proceedings, vol. 189. CEUR-WS.org (2006)
6. Baader, F., Peñaloza, R., Suntisrivaraporn, B.: Pinpointing in the description logic EL^+. In: Hertzberg, J., Beetz, M., Englert, R. (eds.) Proc. 30th Annual GermanConf. on Artificial Intelligence (KI'07). Lecture Notes in Computer Science, vol. 4667, pp. 52–67. Springer (2007), https://doi.org/10.1007/978-3-540-74565-5_7
7. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 19–99. Elsevier and MIT Press (2001)
8. Green, C.: Theorem proving by resolution as a basis for question-answering systems. Machine Intelligence pp. 183–205 (1969)
9. Greiner, R., Smith, B.A., Wilkerson, R.W.: A correction to the algorithm in reiter's theory of diagnosis. Artif. Intell. 41(1), 79–88 (1989), http://dx.doi.org/10.1016/0004-3702(89)90079-9
10. Horridge, M.: Justification based explanation in ontologies. Ph.D. thesis, University of Manchester, UK (2011), http://www.manchester.ac.uk/escholar/uk-ac-man-scw:131699
11. 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). pp. 167–172. AAAI Press / The MIT Press (2004), http://www.aaai.org/Library/AAAI/2004/aaai04-027.php
12. Kalyanpur, A.: Debugging and Repair of OWL Ontologies. Ph.D. thesis, University of Maryland College Park, USA (2006)
13. 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). LNCS, vol. 4825, pp. 267–280. Springer (2007)
14. Kazakov, Y.: Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In: Boutilier, C. (ed.) Proc. 21st Int. Joint Conf. on Artificial Intelligence (IJCAI'09). pp. 2040–2045. IJCAI (2009)
15. 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. pp. 196–211 (2014)

16. 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), 1–61 (2014)
17. Manthey, N., Peñaloza, R., Rudolph, S.: Efficient axiom pinpointing in EL using SAT technology. In: Lenzerini, M., Peñaloza, R. (eds.) Proc. 29th Int. Workshop on Description Logics (DL'16). CEUR Workshop Proceedings, vol. 1577. CEUR-WS.org (2016), http://ceur-ws.org/Vol-1577/paper_33.pdf
18. 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). Lecture Notes in Computer Science, vol. 7857, pp. 86–101. Springer (2012), http://dx.doi.org/10.1007/978-3-642-39611-3_13
19. 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). pp. 633–640. ACM (2005), http://doi.acm.org/10.1145/1060745.1060837
20. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. 32(1), 57–95 (1987), http://dx.doi.org/10.1016/0004-3702(87)90062-2
21. 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). Lecture Notes in Computer Science, vol. 5663, pp. 84–99. Springer (2009), http://dx.doi.org/10.1007/978-3-642-02959-2_6
22. 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). pp. 1093–1098. AAAI Press/IJCAI (2011)
23. Vescovi, M.: Exploiting SAT and SMT Techniques for Automated Reasoning and Ontology Manipulation in Description Logics. Ph.D. thesis, University of Trento, Italy (2011), http://eprints-phd.biblio.unitn.it/477/