Efficient Axiom Pinpointing in $\mathcal{EL}$ using SAT Technology*

Norbert Manthey[1], Rafael Peñaloza[2], and Sebastian Rudolph[1]

[1] TU Dresden, Germany, {firstname.lastname}@tu-dresden.de
[2] Free University of Bozen-Bolzano, Italy, rafael.penaloza@unibz.it

Abstract. We propose a novel approach to axiom pinpointing based on a reduction to the SAT task of minimal unsatisfiable subformula enumeration, allowing highly optimized algorithms and data structures developed for SAT solving in the last two decades to be used in this problem. Exploiting the specific properties of axiom pinpointing, we apply further optimizations resulting in considerable runtime improvements of several orders of magnitude. Our ideas are implemented in SATPin, a system capable of performing axiom pinpointing in large biomedical ontologies faster than other existing tools. While our paper focuses on a slight extension of $\mathcal{EL}$, the presented approach directly generalizes to all ontology languages for which consequence-based reasoning methods are available.

1 Introduction

Axiom pinpointing is the task of identifying the axioms in an ontology that are responsible for a consequence to follow. This task has been successfully used to correct modeling errors and understand unexpected consequences from very large ontologies. For example, the 2007 version of the very large bio-medical ontology Snomed CT[3] incorrectly implied that every *amputation of finger* was also *amputation of arm*; i.e., whenever a patient had a finger amputated, the ontology would imply that they had also lost their arm. Using automated axiom pinpointing tools it was possible to identify the 6 axioms (from over 300,000) that caused this error [6]. Eventually, this led to a change in the modelling strategy followed by the developers of Snomed, to avoid the error that caused this fault [5]. Beyond understanding and correcting consequences, axiom pinpointing has applications in many different reasoning scenarios, like ontology revision [18], context-based reasoning [3], error-tolerant reasoning [15], and reasoning with probabilities [9, 20], provenance, and trust [21], to name a few. It is thus crucial to develop axiom pinpointing tools capable of handling huge ontologies.

For $\mathcal{EL}^+$ ontologies [2], it is possible to reduce axiom pinpointing to the enumeration of the minimal unsatisfiable subformulas (MUS) of a propositional formula [22]. Thus, one can take advantage of the numerous developments made by the SAT community over the last decades (e.g., clause learning, the two-watched-literal data structure) to build an efficient system for axiom pinpointing. We build on top of previous work and

* Partially supported by DFG throughd project HO 1294/11-1 and 'cfAED.'

[3] http://www.ihtsdo.org/snomed-ct

identify several optimizations for enumeration problem. Indeed, the propositional formula obtained by the reduction to SAT has a very specific shape, which can be exploited by specialized methods. We show that incremental SAT solving, partial restarts, and an improved search space pruning strategy can improve the efficiency of the axiom pinpointing.

We implemented these optimizations in the new SATPin system. We compared the efficiency of SATPin against other axiom-pinpointing tools over real-life bio-medical ontologies that have been used as benchmarks before. Our experiments show that SATPin is an efficient tool for axiom pinpointing that can be used in practice over large inputs. Interestingly, our approach depends only on the structure of the propositional formula, and can be directly generalized to any other ontology language allowing consequence-based reasoning beyond $\mathcal{EL}^+$.

2 Preliminaries

We assume that the reader is familiar with the DL $\mathcal{EL}^+$, and briefly describe the completion algorithm for deciding subsumption in this logic [2]. The algorithm has two phases: *normalization* and *completion*. Normalization transforms the ontology into an equivalent one (w.r.t. subsumption of concept names), where all GCIs are in *normal form* $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B$, or $\exists r.A \sqsubseteq B$, or $A \sqsubseteq \exists r.B$. In general, the normalization step maps every axiom α into a set of axioms $\mathsf{NF}(\alpha)$ obtained by the application of some simplification rules. For example, the GCI $A \sqsubseteq B_1 \sqcap B_2$ is not in normal form, but it can be simplified to obtain $\mathsf{NF}(A \sqsubseteq B_1 \sqcap B_2) = \{A \sqsubseteq B_1, A \sqsubseteq B_2\}$. For more details on the normalization procedure, we refer the reader to [2]. For the purpose of this paper, it suffices to recall that the function NF maps every axiom to a set of axioms in normal form, preserving the semantics of the original axiom w.r.t. subsumption of concept names. The normalization of the ontology $\mathcal{T}$ is $\mathsf{NF}(\mathcal{T}) := \bigcup_{\alpha \in \mathcal{T}} \mathsf{NF}(\alpha)$.

In the completion phase, the normalized ontology $\mathsf{NF}(\mathcal{T})$ is saturated through an exhaustive application of *completion rules* of the form $\mathcal{A} \Rightarrow \alpha$, where $\mathcal{A} \cup \{\alpha\}$ is a finite set of axioms in normal form. This rule is *applicable* to the ontology $\mathcal{T}$ if $\mathcal{A} \subseteq \mathcal{T}$; its application adds α to $\mathcal{T}$. Completion rules generate the logical closure of the axioms in normal form w.r.t. a limited signature. For example, the $\mathcal{EL}^+$ completion rule $\{X \sqsubseteq Y, Y \sqsubseteq Z\} \Rightarrow X \sqsubseteq Z$ expresses that if the two subsumption relations $X \sqsubseteq Y$, $Y \sqsubseteq Z$ hold, then X is also subsumed by Z. In general, the rule $\mathcal{A} \Rightarrow \alpha$ states that if all the axioms in $\mathcal{A}$ hold, α must hold too. To ensure termination, the rule $\mathcal{A} \Rightarrow \alpha$ is only applicable if $\alpha \notin \mathcal{T}'$. $\mathfrak{R}$ denotes the set of all completion rules. Let $\mathsf{c}(\mathcal{T})$ be the ontology obtained from $\mathcal{T}$ after normalization and completion. For every two concept names A, B appearing in $\mathcal{T}$, $\mathcal{T} \models A \sqsubseteq B$ iff $A \sqsubseteq B \in \mathsf{c}(\mathcal{T})$; i.e., the completion algorithm makes all the implicit subsumptions between concept names that can be derived from $\mathcal{T}$ explicit.

Example 1. Consider the ontology $\mathcal{T}_{\mathsf{exa}} := \{A \sqsubseteq \exists r.A \sqcap Y, Y \sqsubseteq B, \exists r.Y \sqsubseteq B\}$. The last two axioms are already in normal form, but the axiom $A \sqsubseteq \exists r.A \sqcap Y$ is not. It can be transformed to normal form by separating the conjunction on the right-hand side of the axiom, as described before. Hence,

$$\mathsf{NF}(\mathcal{T}_{\mathsf{exa}}) = \{A \sqsubseteq \exists r.A, A \sqsubseteq Y, Y \sqsubseteq B, \exists r.Y \sqsubseteq B\}.$$

Applying the completion rules to this normalized ontology yields

$$\mathsf{c}(\mathcal{T}_{\mathsf{exa}}) := \{A \sqsubseteq \exists r.A, A \sqsubseteq Y, Y \sqsubseteq B, \exists r.Y \sqsubseteq B\} \cup \{A \sqsubseteq \exists r.Y, A \sqsubseteq \exists r.B, A \sqsubseteq B\}.$$

We conclude $\mathcal{T}_{\mathsf{exa}} \models A \sqsubseteq B$, as this subsumption relation appears in $\mathsf{c}(\mathcal{T}_{\mathsf{exa}})$.

In axiom pinpointing we are interested in finding the axiomatic causes for a subsumption to hold, in order to understand this consequence. Thus, we want all the minimal sub-ontologies that entail this subsumption.

Definition 2 (MinA). *A* MinA *for $A \sqsubseteq B$ w.r.t. the ontology $\mathcal{T}$ is a subset $\mathcal{M} \subseteq \mathcal{T}$ s.t. (i) $\mathcal{M} \models A \sqsubseteq B$ and (ii) for all $\mathcal{S} \subset \mathcal{M}$, $\mathcal{S} \not\models A \sqsubseteq B$.* Axiom-pinpointing *is the task of finding all MinAs for a subsumption w.r.t. an ontology.*

For instance, there are two MinAs for $A \sqsubseteq B$ w.r.t. the ontology $\mathcal{T}_{\mathsf{exa}}$ of Example 1: $\mathcal{M}_1 := \{A \sqsubseteq \exists r.A \sqcap Y, Y \sqsubseteq B\}$, and $\mathcal{M}_2 := \{A \sqsubseteq \exists r.A \sqcap Y, \exists r.Y \sqsubseteq B\}$.

One approach for solving this problem is to create a Horn formula whose satisfying interpretations can be mapped to sub-ontologies entailing the subsumption relation. Before describing this reduction in detail, we recall some basic notions of SAT. Consider a fixed infinite set $\mathcal{V}$ of Boolean *variables*. A *literal* is a variable v (*positive literal*) or a negated variable $\overline{v}$ (*negative literal*). The variable of a literal x is denoted as $\mathsf{var}(x)$. The *complement* $\overline{x}$ of a positive (negative) literal x is the negative (resp., positive) literal with the same variable as x. The *complement* of a set of literals S is $\overline{S} := \{\overline{x} \mid x \in S\}$. A *clause* C is a finite set of literals, representing the disjunction of its elements. A clause containing single literal is called a *unit clause*. *Formulas* are finite multisets of clauses, which intuitively represent the conjunction of its elements; i.e., we consider only formulas in conjunctive normal form. For two clauses C_1, C_2 with $x \in C_1$, and $\overline{x} \in C_2$, the *resolvent of C_1 and C_2 upon x* is the clause $C_1 \otimes_x C_2 := (C_1 \setminus \{x\}) \cup (C_2 \setminus \{\overline{x}\})$.

A sequence of literals M is *consistent*, if $x \in M$ implies $\overline{x} \notin M$. For simplicity, we view consistent sequences M as sets throughout this paper. An *interpretation* is a consistent set of literals I. The *reduct* $F|_I$ of a formula F with respect to the interpretation I is the multiset $F|_I := \{C \setminus \overline{I} \mid C \in F, C \cap I = \emptyset\}$. I *satisfies* a formula F if $F|_I = \emptyset$. F is *satisfiable* if there is an interpretation that satisfies it. The standard reasoning problem in SAT is to decide satisfiability of a formula.

A major operation in modern SAT solvers is *unit propagation*, based on the fact that a unit clause $C = \{x\}$ is only satisfied by interpretations using the literal x. Given a formula F and a consistent sequence M of literals, unit propagation returns the set of all literals (including M) that must occur in an interpretation to satisfy $F|_M$. The interpretation J is initialized with M. If there are unit clauses in the current reduct, J is extended with the corresponding literal. Additionally, the clause C is stored as the reason for this extension. If no further unit clauses can be found, the algorithm returns the final interpretation J.

Most modern SAT solvers follow essentially the same approach, known as CDCL [16]. Unit propagation is applied as long as possible to obtain all the literals that must appear in every model of the formula. When no units can be propagated, it checks for a *conflict*: a clause falsified by the current interpretation. If there is no

conflict, the interpretation of a variable is guessed (*search decision*), and unit propagation is applied to deduce the consequences of this guess. If a conflict is found, some of the guesses made so far are wrong. *Conflict analysis* finds the choices that led to the conflict, and a new *learned clause* C is added to the formula to avoid repeating this choice. The clause C is used to undo parts of the current partial interpretation in a way that unit propagation can be applied again, and the process continues. If a conflict is found independent of any search decision, the formula is unsatisfiable. Otherwise, if all variables of the formula can be assigned a truth value without a conflict, the formula is satisfiable. Notice that the utility of SAT solving goes far beyond propositional logic: with specialized data structures, heuristics and simplification techniques, modern SAT solvers are the back-end for many industrial tasks [8].

One approach for axiom pinpointing originally proposed in [22] builds a propositional formula that simulates the process of the completion algorithm. The computation of the MinAs for an atomic subsumption is thus reduced to an enumeration problem over this formula. We briefly describe this translation next, but refer the reader to [22] for all the details. For every axiom $\alpha \in \mathcal{T} \cup \mathsf{c}(\mathcal{T})$ we introduce a unique Boolean variable x_α, which represents the axiom α throughout the completion process. Then, we build the formula $F_\mathcal{T} := F_\mathsf{n} \cup F_\mathsf{c}$, where

$$F_\mathsf{n} := \{\{\overline{x_\alpha}, x_\beta\} \mid \alpha \in \mathcal{T}, \beta \in \mathsf{NF}(\alpha)\},$$
$$F_\mathsf{c} := \{\{x_\alpha\} \cup \{\overline{x_\beta} \mid \beta \in \mathcal{A}\} \mid \mathcal{A} \Rightarrow \alpha \in \mathfrak{R}, \mathcal{A} \cup \{\alpha\} \subseteq \mathsf{c}(\mathcal{T})\}.$$

These formulas describe all the possible causes for an axiom to appear in $\mathsf{c}(\mathcal{S})$ for some subontology $\mathcal{S}$ of $\mathcal{T}$. The clauses in F_n (equivalent to implications $x_\alpha \rightarrow x_\beta$) state that, for every axiom α occurring in $\mathcal{S}$, all the axioms in normal form generated by α must be in $\mathsf{c}(\mathcal{S})$. The formula thus simulates the normalization step. Similarly, the completion rule $\mathcal{A} \Rightarrow \alpha$ expresses that if all axioms in $\mathcal{A}$ are contained in $\mathsf{c}(\mathcal{S})$, then so must be α; this logical dependency is expressed in the clauses of the formula F_c which can equivalently be written as $\bigwedge_{\beta \in \mathcal{A}} x_\beta \rightarrow x_\alpha$.

Example 3. Consider again the ontology $\mathcal{T}_\mathsf{exa}$ from Example 1. For brevity, we use the following shorthand for the axioms in $\mathcal{T}_\mathsf{exa} \cup \mathsf{c}(\mathcal{T}_\mathsf{exa})$:

$\alpha_1 : A \sqsubseteq \exists r.A \sqcap Y$, $\alpha_2 : Y \sqsubseteq B$, $\alpha_3 : \exists r.Y \sqsubseteq B$,

$\beta_1 : A \sqsubseteq \exists r.A$, $\beta_2 : A \sqsubseteq Y$, $\beta_3 : A \sqsubseteq \exists r.Y$, $\beta_4 : A \sqsubseteq \exists r.B$, $\beta_5 : A \sqsubseteq B$.

Then, $F_\mathsf{n} = \{\{\overline{x_{\alpha_1}}, x_{\beta_1}\}, \{\overline{x_{\alpha_1}}, x_{\beta_2}\}\}$ since the axioms β_1 and β_2 are generated by the normalization of the axiom α_1, and all other axioms in $\mathcal{T}_\mathsf{exa}$ are already in normal form. The formula F_c is composed by the clauses

$\{\overline{x_{\beta_1}}, \overline{x_{\beta_2}}, x_{\beta_3}\}, \{\overline{x_{\beta_3}}, \overline{x_{\alpha_2}}, x_{\beta_4}\}, \{\overline{x_{\beta_2}}, \overline{x_{\alpha_2}}, x_{\beta_5}\}, \{\overline{x_{\beta_3}}, \overline{x_{\alpha_3}}, x_{\beta_5}\}, \{\overline{x_{\beta_1}}, \overline{x_{\beta_5}}, x_{\beta_4}\}$.

There are two clauses in F_c that entail β_5. Although the completion algorithm uses only one of them, the translation to SAT must preserve both to know all the possible ways in which axioms can be derived from the knowledge in $\mathcal{T}_\mathsf{exa}$.

Given $\mathcal{S} \subseteq \mathcal{T}$, let $X_\mathcal{S} := \{\{x_\alpha\} \mid \alpha \in \mathcal{S}\}$; i.e., the conjunction of all the variables from the axioms in $\mathcal{S}$. A model I of $X_\mathcal{S} \wedge F_\mathcal{T}$ satisfies $\mathcal{S} \cup \mathsf{c}(\mathcal{S}) \subseteq \{\alpha | x_\alpha \in I\}$. On the other hand, the set $\{x_\alpha \mid \alpha \in \mathcal{S} \cup \mathsf{c}(\mathcal{S})\}$ is a model of $X_\mathcal{S} \wedge F_\mathcal{T}$. It follows that $\mathcal{S} \models A \sqsubseteq B$ iff $x_{A \sqsubseteq B} \in I$ for all interpretations I satisfying $X_\mathcal{S} \wedge F_\mathcal{T}$. This means that, in order to find all MinAs for $A \sqsubseteq B$ w.r.t. $\mathcal{T}$, it suffices to compute all

minimal subsets M of $X_\mathcal{T}$ such that $M \wedge F_\mathcal{T} \wedge \{\overline{x_{A \sqsubseteq B}}\}$ is unsatisfiable. Notice that $X_\mathcal{T} \wedge F_\mathcal{T}$ is always satisfiable. On the other hand, since $\mathcal{T} \models A \sqsubseteq B$, the formula $X_\mathcal{T} \wedge F_\mathcal{T} \wedge \{\overline{x_{A \sqsubseteq B}}\}$ is unsatisfiable. Hence, this problem is well-defined.

3 Enumerating MinAs

We have constructed, from an ontology $\mathcal{T}$, the formula $F_\mathcal{T}$ that encodes the derivation steps made by the completion algorithm, and the set of *choice variables* $X_\mathcal{T}$. By construction, $X_\mathcal{T} \wedge F_\mathcal{T}$ is satisfiable. Given a consequence α, we want to enumerate all the minimal subsets $M \subseteq X_\mathcal{T}$ such that $M \wedge F_\mathcal{T} \wedge \overline{x_\alpha}$ is unsatisfiable ($M \wedge F_\mathcal{T} \wedge \overline{x_\alpha} \equiv \bot$). Our approach does not depend on the precise shape of the formula $F_\mathcal{T}$, but rather on these properties. Hence, we consider an arbitrary satisfiable formula F, a set X of propositional variables such that $X \wedge F$ is satisfiable, and a propositional variable q such that $X \wedge F \wedge \overline{q} \equiv \bot$.

Enumerating all minimal subsets M of X such that a formula is unsatisfiable is closely related to finding all *minimal unsatisfiable subformulas* (MUS) of a propositional formula [8]. We consider the *group-MUS* problem, in which some clauses have to be handled together. In our case, only the clauses in X can be selected separately from the rest of the formula. A single group-MUS is a minimal subset $M \subseteq X$, s.t. $M \wedge F \wedge \overline{q} \equiv \bot$ still holds. We solve the *all-group-MUS* problem [13,19], and enumerate all such minimal sets M. However, we encounter only a special case of all-group-MUS: each group contains exactly one unit clause $\{x_\alpha\}$, corresponding to an axiom α from the ontology. Thus, instead of using a general-purpose all-group-MUS tool [13], we exploit the specific properties of axiom pinpointing and improve the performance of a solver.

In essence, we make several calls to a SAT solver to find all MinAs. For the first MinA, we try to prove satisfiability of $X \wedge F \wedge \overline{q}$ choosing, at search decisions, a variable from X to be satisfied. This process eventually leads to a conflict. Conflict analysis yields an $M \subseteq X$ that led to the conflict (a MinA). Learning the negation of this MinA guarantees that the conflict cannot be found again. The same approach leads to the next MinA, repeating this process until no more conflicts exist. To handle large ontologies, many optimizations are needed.

Incremental SAT Solving. To find one MinA, we use *incremental SAT* [8] solving. The execution of a SAT solver is initialized with a set of *assumption* literals, used as search decisions before any other variables. We use the activation variables X as assumption literals. Whenever a decision is needed during the satisfiability decision procedure, the algorithm first activates one of the variables in X as true (adding a new axiom to the current MinA candidate). Since $M \wedge F \wedge \overline{q} \equiv \bot$, an inconsistency is eventually found, which is characterized by the implication $(M \wedge F) \rightarrow q$, where $M \subseteq X$ is the set of assigned assumption literals. Hence, the solver is interrupted as soon as q is implied by the set M (see Algorithm 1).

Minimizing the Candidate Set. Algorithm 1 finds a (possibly not minimal) set M s.t. $(M \wedge F) \rightarrow q$. To reduce M to a MinA, we use conflict analysis [10]. Based on the reason clause C for q, produce a subset of literals of M by resolving all literals from C away with their reason clauses (Algorithm 2). Starting with the reason clause C for q

Algorithm 1 Return a set of literals $R \subseteq X$ that lead to $(F \wedge R) \rightarrow q$.

`implies` (formula F, literal q, literal sequence X, $reason$)

Output: $\bot$, or set of literals R with $(F \wedge R) \rightarrow q$

```
IMP1 M := ∅                                  // initialize as empty
IMP2 while X ≠ ∅                             // while literals left
IMP3   if q ∈ UP(F, M, reason)               // check value of q
IMP4     return analyze(q, reason)           // reduce candidate
IMP5   M := M ∪ {x} for some x ∈ X           // add x ∈ X to M
IMP6   X := X \ {x}                          // remove x from X
IMP7 return ⊥                                // return the result
```

we resolve on all literals of the intermediate resolvents, until no literal has a reason clause. C then contains only variables that have been assigned a truth value as search decision; i.e., an assumption. The set R obtained from this conflict analysis is still a MinA candidate that needs to be minimized, but typically (much) smaller than M. For each literal $r \in R$ we check the implication $(R \wedge F \setminus \{r\}) \rightarrow q$, and remove r from R, if the check succeeds. If no more removals are possible, the set R is returned.

Enumerating All MinAs. We iterate the MinA computation procedure to detect the remaining MinAs, guaranteeing that answers are not repeated, as shown in Algorithm 3. First we check whether there is at least one MinA (`ENU2`) and abort if this is not the case; i.e., the consequence does not follow from the ontology. Then, we create an object responsible for enumerating all candidate subsets M of literals. This is the major part of the algorithm. If R represents a potential MinA, it is minimized and added to the set of MinAs S (`ENU6`–`ENU8`). It is also added to the enumeration object, to avoid producing it again in future iterations (`ENU9`). If this addition makes no further candidate sets possible (`ENU9`), or if there are no other candidate sets (`ENU11`), the algorithm stops. Otherwise, the next candidate set $M \subseteq X$ is tested (`ENU14`). When the enumeration finishes, the resulting set S is returned (`ENU15`). Notice that the call in `ENU14` may not return any new potential MinA; e.g., if there is only one MinA. To ensure completeness this check has to be performed for all candidates M.

Example 4. We want to find all the minimal subsets M of $\{x_{\alpha_1}, x_{\alpha_2}, x_{\alpha_3}\}$ s.t. $M \wedge F_{\mathcal{T}_{\mathsf{exa}}} \wedge \overline{x_{\beta_5}} \equiv \bot$. We activate first the literal x_{α_1}, which unit propagates to

Algorithm 2 Return a set of literals R that lead to the implication $(F \wedge R) \rightarrow q$.

`analyze` (map $reason$, literal q)

Output: Set of literals R that imply q wrt to F, $(F \wedge R) \rightarrow q$

```
ANA1 C := reason(var(q))                     // find clause that implied q
ANA2 while C ≠ ∅                             // while literals left
ANA3   c ∈ C, C := C \ {c}                   // select a literal c ∈ C
ANA4   if reason(var(c)) ≠ ⊥                 // if there is a reason for c
ANA5     C := (C ∪ {c}) ⊗_c reason(var(c))   // resolve with this reason
ANA6   else R := R ∪ {c̄}                     // else add c̄ to the result
ANA7 return R                                // return the result
```

Algorithm 3 Return all minimal sets of literals $R \subseteq X$ that lead to $(F \wedge R) \rightarrow q$.

`enumerate` (formula F, set of literals X, literal q)

Output: Set S of set of literals R with $R \subseteq X$ and $(F \wedge R) \rightarrow q$

```
ENU1  S := ∅
ENU2  R := implies(F, q, X, reason)        // Is there a MinA?
ENU3  if R = ⊥ then return ∅               // there are no MinA
ENU4  setup enumerator(X)                  // setup enumeration
ENU5  while ⊤                              // check all candidates
ENU6    if R ≠ ⊥                           // if there was a MinA
ENU7      R := minimize(F, R, q)           // minimize candidate
ENU8      S := S ∪ {R}                     // add R to set of MinAs
ENU9      if enumerator.avoid(R) = ⊥       // disallow this MinA
ENU10       break                          // no more MinAs
ENU11   if enumerator.hasNext() = ⊥        // Do other MinAs exist?
ENU12     break                            // no more MinAs
ENU13   M := enumerator.next()             // next MinA candidate
ENU14   R := implies(F, q, M, reason)      // Is there a MinA?
ENU15 return S                             // return set of MinAs
```

x_{β_1}, x_{β_2}, and x_{β_3}. At this point, a new choice is needed. If we activate x_{α_2}, then unit propagation enforces x_{β_5} to be true, finding a contradiction; i.e., $\{\alpha_1, \alpha_2\}$ is a MinA. We learn the clause $\{\overline{x_{\alpha_1}}, \overline{x_{\alpha_2}}\}$ and restart the incremental SAT solver, finding the second MinA $\{\alpha_1, \alpha_3\}$. After disallowing this MinA, the only remaining candidate set is $\{x_{\alpha_2}, x_{\alpha_3}\}$, which does not entail the wanted implication.

Candidate Enumeration. Candidate enumeration is initialized with the set of literals X. While there are candidates left, a new $M \subseteq X$ is chosen. A naïve approach is to enumerate all subsets of X as candidates. This is unfeasible as it would need to verify $2^{|X|}$ candidates, with $|X|$ very large. We partition X into the set of *relevant* literals $V = \mathsf{lits}(S)$ and the remaining literals T. The relevant literals refer to axioms that belong to some MinA. At the beginning, we do not know which literals are relevant and which not. Hence, T is initialized to be X. Whenever a MinA R is found, V and T are updated accordingly: $V := V \cup R$, and $T := T \setminus R$; all the variables in this MinA are now known to be relevant. The new candidates are sets of the form $V' \cup T$ for $V' \subseteq V$. In this way, the number of candidates is bounded by $2^{|\mathsf{lits}(S)|}$. As $|\mathsf{lits}(S)|$ is typically much smaller than $|X|$ the search space is reduced considerably. Since the sets T and V change during the execution of the algorithm, it is important to verify that candidates are never tested twice. We also apply the Hitting Set Tree (HST) approach developed originally for axiom pinpointing in expressive DLs [11]. After one MinA M has been found, we try to find a new MinA over the set of candidate variables $X \setminus \{m\}$ for every $m \in M$. Iteratively repeating this approach yields a search tree, where each solution is different from all its predecessors in this tree. As in the relevant enumeration, the current set V' is extended with T to form a candidate. Finally, we exploit an idea developed for reducing the search space in group-MUS-enumeration [13]. Given a candidate $M \subseteq X$, if $(M \wedge F) \not\rightarrow q$, but $(X \wedge F) \rightarrow q$ holds, then we conclude that in any future set of literals M' at least one

literal $m' \in (X \setminus M)$ has to be present to result in $(M' \wedge F) \rightarrow q$: if $(M \wedge F) \not\rightarrow q$, then the same still holds for any subset of M. Hence, once we found a candidate M that failed the imply check, any future candidate has to pick one of these literals in $X \setminus M$. In combination with the relevant enumeration, this set is reduced to $V \setminus V'$.

Example 5. We give an example demonstrating the benefits of candidate enumeration. Suppose that we want to compute the MinAs for $A \sqsubseteq B$ w.r.t. the ontology $\{H\sqsubseteq B,\ A\sqsubseteq C,\ A\sqsubseteq D,\ C\sqsubseteq E,\ D\sqsubseteq E,\ E\sqsubseteq B,\ A\sqsubseteq F,\ F\sqsubseteq B,\ A\sqsubseteq G\}$. Initially, all axioms belong to the set T of (potentially) non-relevant axioms. The first MinA we find is $\{A \sqsubseteq C, C \sqsubseteq E, E \sqsubseteq B\}$. These three axioms are marked as relevant:

$$\underbrace{H\sqsubseteq B,\ A\sqsubseteq D,\ D\sqsubseteq E,\ A\sqsubseteq F,\ F\sqsubseteq B,\ A\sqsubseteq G}_{T},\ \underbrace{A\sqsubseteq C,\ C\sqsubseteq E,\ E\sqsubseteq B}_{V}$$

We start to look for MinAs in $T \cup V'$ for all proper subsets $V' \subset V$. Let $V' = \{A\sqsubseteq C, C\sqsubseteq E\}$. We find the MinA $\{A\sqsubseteq F,\ F\sqsubseteq B\}$, and add these axioms to V.

$$\underbrace{H\sqsubseteq B,\ A\sqsubseteq D,\ D\sqsubseteq E,\ A\sqsubseteq G}_{T},\ \underbrace{A\sqsubseteq C,\ C\sqsubseteq E,\ E\sqsubseteq B,\ A\sqsubseteq F,\ F\sqsubseteq B}_{V}$$

We continue searching for new MinAs in all sets $T \cup V'$ for all proper subsets V' of V which do not contain a known MinA. We can pick, e.g., $V' = \{A\sqsubseteq C, C\sqsubseteq E, A\sqsubseteq F\}$. Then $T \cup V'$ does not imply the subsumption $A\sqsubseteq B$. Any future MinA must contain at least one of the axioms from $V \setminus V' = \{E\sqsubseteq B,\ F\sqsubseteq B\}$. We continue by trying a new subset of V, e.g. $V' = \{A\sqsubseteq C,\ E\sqsubseteq B,\ A\sqsubseteq F\}$ and obtain $\{A\sqsubseteq D, D\sqsubseteq E, E\sqsubseteq B\}$ as MinA. This again requires to update T and V:

$$\underbrace{H\sqsubseteq B,\ A\sqsubseteq G}_{T},\ \underbrace{A\sqsubseteq C,\ C\sqsubseteq E,\ E\sqsubseteq B,\ A\sqsubseteq F,\ F\sqsubseteq B,\ A\sqsubseteq D,\ D\sqsubseteq E}_{V}$$

The algorithm will then tell us that no other MinAs exist. Recall that typically, the set T will be much bigger than V and using our strategy, we will never have to go through subsets of it.

We realized the enumerator object as a SAT solver. Model enumeration is based on CDCL: once a model I is found, $\overline{I}$ is added to the formula, and the next model is generated. We add only the decision literals of I: all other literals of I are implied. To avoid repeating MinAs, the literals of every MinA R are added to the formula as clause $\overline{R}$, too. For the enumeration, we incrementally add the variables of the last MinA R to the solver. Previous clauses remain valid: previous candidates are not enumerated twice and known MinAs cannot be repeated. For the HST enumeration, we modify the decision heuristic of the solver. A stack of found solutions is kept and the decision heuristic follows the hitting set scheme to enumerate candidates. Finally, inverse enumeration is realized adding the necessary clause $X \setminus M$ to include one of the missing literals. The algorithm spends most time on the imply check, as all literals $x \in X$ have to be applied. Depending on the ontology, X can be very large: the encoding of SNOMED contains 378579 literals. As discussed, the set of relevant literals $V = \mathsf{lits}(S)$ might be much smaller. In our experiments, the largest set V found for SNOMED contains 88 literals. For each imply check, 378491 ($378579 - 88$) literals could be kept in line `IMP1`. Instead of initializing $M = \emptyset$, we can initialize $M = T$ to the set of (currently) irrelevant literals. The initialization is sound, since $(F \wedge T) \not\rightarrow q$. In the implementation we do not undo and recreate the set M, but keep the last state and only perform the needed updates. Theoretically, this optimization improves the

Table 1: Structure of the translation of ontologies

	GO	NCI	FGalen	Snomed
Axioms	20466	46800	36544	378579
Variables	237389	338380	2729734	13419995
Clauses	294782	342825	3843812	38276251

algorithm by two orders of magnitude over Snomed: 99.98 % of the work is saved in the `implies` routine; i.e., maintaining relevant variables may be the difference between solvability and unfeasibility.

4 Experimental Evaluation

We implemented a new tool called SATPin, based on Minisat 2.2.[4] To test our ideas, we ran SATPin on four well-known $\mathcal{EL}^+$ biomedical ontologies, which have been widely used as benchmarks for DL reasoners, specially in the context of axiom-pinpointing: the Gene Ontology, NCI, the $\mathcal{EL}^+$ version of FullGalen, and the 2010 version of Snomed. All computations ran with a 5h timeout (18000s) on an Intel Xeon CPU at 2.6GHz and a memory limit of 6.5GB. We compare the performance of SATPin with the state-of-the-art MUS enumeration tool Marco [13], and the $\mathcal{EL}^+$ axiom pinpointing tool EL2MUS [1], which is also based on a translation to SAT.[5]

Each ontology was transformed into a propositional formula by *el2sat_all* [22]. Table 1 summarizes the properties of these ontologies and their translations; the number of axioms in the original ontology is also the number of selection variables used by SATPin. Snomed is an order of magnitude larger than the other three test ontologies; in fact, one of the main problems when dealing with Snomed is to handle the memory consumption issues. For each of the three smaller ontologies, we computed all the MinAs for 100 different consequences: 50 randomly chosen, and 50 selected as those where the variable $x_{A \sqsubseteq B}$ appears the most often in $F_{\mathcal{T}}$, indicating that they have the most MinAs, as originally designed in [23], and later used also in [1]. For Snomed, we selected 34 consequences that are known to be problematic for axiom pinpointing, due to the large number and size of their MinAs [23]. We ran SATPin, EL2MUS, and Marco on all 334 problems, where SATPin uses the combination of all enumeration mechanisms and the relevant variable selection optimization. All systems terminated successfully on the 300 instances corresponding to GO, NCI, and FullGalen, but Marco ran out of memory on all Snomed instances. Thus we consider only the first 300 tests for comparison. The results are summarized in Table 2.

SATPin clearly outperforms the other two tools in GO. In FullGalen, Marco behaves much worse than the other two tools. At first sight, it seems that SATPin has the worst behavior in NCI by far. However, EL2MUS was faster than SATPin in only 23 of the samples tested for this ontology, and was much slower (up to an order of magnitude) in all others. The median and 90th percentile for SATPin on these tests is lower than those for Marco and EL2MUS. The average performance of SATPin is

[4] `http://minisat.se/`

[5] All experimental data is available at: `http://goo.gl/kJ0sE4`.

Table 2: CPU time (s) required by SATPin, EL2MUS, and Marco

	Tool	Avg	StDev	Max	Median	p-90
GO	Marco	20.01	26.73	171.79	10.11	42.29
	EL2MUS	7.63	18.60	118.02	0.83	21.54
	SATPin	3.74	7.54	46.93	1.16	7.76
NCI	Marco	75.95	184.44	1071.42	13.81	151.91
	EL2MUS	50.03	137.61	744.84	2.53	122.73
	SATPin	190.15	990.93	8823.32	0.72	87.57
FullGalen	Marco	100.94	9.71	163.94	97.59	106.45
	EL2MUS	5.80	4.09	24.18	4.52	10.15
	SATPin	7.38	11.80	90.25	3.91	13.83
Accumulated	Marco	65.64	112.61	1071.42	21.04	105.80
	EL2MUS	21.15	82.52	744.84	3.40	23.84
	SATPin	67.09	576.88	8823.32	2.98	14.75

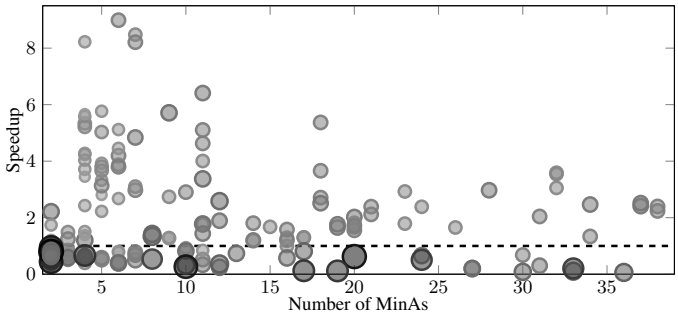

Fig. 1: Proportional speedup of SATPin w.r.t. EL2MUS against the number of MinAs (horizontal axis) and the average MinA size (node size and tone).

affected by *three* instances that took over 3000s, which explains the huge standard deviation. If only these three instances are removed, the average time taken by SATPin decreases drastically to only 40.55s. A similar, although less extreme situation was observed for FullGalen.

In theory, SATPin is affected by the branching factor produced by the HST method, and the number of relevant variables used. This is confirmed in Figure 1, which shows the proportional speedup of SATPin w.r.t. EL2MUS against the number of MinAs in each experiment. The improvement shown by SATPin grows inversely to the number of MinAs. The size and shade of each dot is proportional to the average MinA size in that instance. Clearly, the relative performance of SATPin decreases as this average increases: all instances containing large MinAs appear below the dashed line, which signals where both SATPin and EL2MUS perform equally. Notice that real-world ontologies are typically well-structured, and their consequences have only a few MinAs of small size [7, 26]. Our experiments also confirm that this is the case, as shown in Table 3.

We did another experiment to understand the influence of (i) the order of the selection variables, and (ii) the variable separation optimization. We ran the 20 instances of GO where SATPin behaved the worst again with (i) the order of the selection variables reversed, and (ii) the variable separation optimization deactivated. In the latter, the average CPU time increased from 13 to 279s; increasing

Table 3: Number and sizes of MinAs found

	#MinAs			Max	#Relevant Axioms		
	Avg	Max	Med	Size	Avg	Max	Med
GO	11.34	38	7	9	13.15	30	13
NCI	6.78	36	4	10	14.65	43	12
FGALEN	1.39	10	1	19	7.41	24	6
Accumul.	6.50	38	2	19	11.74	43	9

in the worst instance from 47 to 1239s; i.e., the optimization is really effective. The theoretical speedup is 681: at most 30 out of 20466 axioms appear in MinAs; however, this speedup is not reached in practice. When the order of the selection variables was reversed, the CPU time variated to up to 2x in both directions. The relative performance of SATPIN against EL2MUS and MARCO was not affected by the ordering.

We compared the performance of SATPIN and EL2MUS on the 34 very hard instances from SNOMED CT. These instances are so hard that their full set of MinAs was previously unknown. Only 9 of these instances were solved by both tools, and EL2MUS solved three additional ones. The solved cases had in average 16.4 MinAs (maximum 33) with an average size of 14 axioms each. In the other cases, before timing-out SATPIN found in average 32 MinAs containing 16 axioms each. In an extreme case, SATPIN found 96 MinAs, with up to 30 axioms. EL2MUS succeeded in this case, proving that there existed no more MinAs: SATPIN computed the full answer to this instance, but could not verify it within the time limit. Recall that these 34 test cases were specially selected for their hardness; most SNOMED CT consequences have less than ten MinAs [26].

Discussion. Other axiom-pinpointing systems for $\mathcal{EL}^+$ are CEL [25], Just [14], and $\mathcal{EL}^+$SAT [22]. Despite several efforts, we were unable to execute our tests on either of the two last systems. CEL limits its execution to the computation of 10 MinAs, and at most 1000s. Thus, we do not include them in our evaluation.

5 Conclusions

We exploited highly optimized tools and data structures from the SAT community to produce an efficient tool for axiom pinpointing in $\mathcal{EL}^+$. The core of our approach is based on the construction of a propositional formula encoding the derivation steps of the completion-based procedure for deciding atomic subsumption. While we focused on the standard completion algorithm for $\mathcal{EL}^+$ [2], the methods easily generalize to any ontology language with consequence-based reasoning methods (e.g. [4, 12, 17, 24]). We evaluated our approach over large bio-medical ontologies. The experiments show that SATPIN behaves better than the general group-MUS solver MARCO, and the pinpointing tool EL2MUS. However, its performance degrades as the number and size of the MinAs found increases. As future work, we will identify the causes and develop methods for avoiding this reduction in performance. We will extend the approach to provide a better support for supplemental reasoning tasks in DLs and other logics. Finally, we intend to perform a more thorough experimental evaluation including a larger class of ontologies and subsumption relations, as well as the missing reasoners.

References

1. 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
2. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. IJCAI-05. Morgan-Kaufmann (2005)
3. Baader, F., Knechtel, M., Peñaloza, R.: Context-dependent views to axioms and consequences of semantic web ontologies. J. of Web Semantics 12–13, 22–40 (2012)
4. Baader, F., Peñaloza, R.: Axiom pinpointing in general tableaux. Journal of Logic and Computation 20(1), 5–34 (2010)
5. Baader, F., Schulz, S., Spackmann, K., Suntisrivaraporn, B.: How should parthood relations be expressed in SNOMED CT? In: Proc. of OBML 2009 (2009)
6. Baader, F., Suntisrivaraporn, B.: Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In: Proc. of KR-MED'08. CEUR-WS, vol. 410 (2008)
7. Bail, S., Horridge, M., Parsia, B., Sattler, U.: The justificatory structure of the NCBO bioportal ontologies. In: Proc. of ISWC 2011, Part I. LNCS, vol. 7031, pp. 67–82. Springer (2011)
8. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
9. Ceylan, İ.İ., Peñaloza, R.: The Bayesian Description Logic $\mathcal{BEL}$. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Proceedings of the 7th International Joint Conference on Automated Reasoning (IJCAR'14). LNCS, vol. 8562, pp. 480–494. Springer International Publishing (2014)
10. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electronic Notes in Theoretical Computer Science 89(4), 543–560 (2003)
11. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: Proc. of 6th Int. Semantic Web Conf. LNCS, vol. 4825, pp. 267–280. Springer (2007)
12. Kazakov, Y., Krötzsch, M., Simancik, F.: The incredible ELK - from polynomial procedures to efficient reasoning with ontologies. J. Autom. Reas. 53(1), 1–61 (2014)
13. Liffiton, M.H., Malik, A.: Enumerating infeasibility: Finding multiple MUSes quickly. In: Gomes, C.P., Sellmann, M. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. LNCS, vol. 7874, pp. 160–175. Springer (2013)
14. Ludwig, M.: Just: a tool for computing justifications w.r.t. el ontologies. In: Proc. of ORE 2014. vol. 1207, pp. 1–7. CEUR Workshop Proceedings (2014)
15. Ludwig, M., Peñaloza, R.: Error-tolerant reasoning in the description logic el. In: Fermé, E., Leite, J. (eds.) Proc. of the 14th European Conf. on Logics in Artificial Intelligence (JELIA'14). LNAI, vol. 8761, pp. 107–121. Springer-Verlag (2014)
16. Marques-Silva, J.P., Sakallah, K.A.: GRASP – a new search algorithm for satisfiability. In: Proceedings of the 1996 IEEE/ACM international conference on computer-aided design. pp. 220–227. ICCAD '96, IEEE Computer Society (1996)
17. Ortiz, M., Rudolph, S., Simkus, M.: Worst-case optimal reasoning for the horn-dl fragments of OWL 1 and 2. In: Proc. of KR 2010 (2010)
18. Peñaloza, R., Thuluva, A.S.: Iterative ontology update using context labels. In: Proc. of OntoChange'15 (2015), to appear
19. Previti, A., Marques-Silva, J.: Partial MUS enumeration. In: des Jardins, M., Littman, M.L. (eds.) Proc. of the 27th AAAI Conference on Artificial Intelligence. AAAI Press (2013)

20. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Probabilistic description logics under the distribution semantics. Semantic Web - Interoperability, Usability, Applicability (To appear) (2015)
21. Schenk, S., Dividino, R., Staab, S.: Reasoning with provenance, trust and all that other meta knowlege in owl. In: SWPM. CEUR, vol. 526. CEUR-WS.org (2009)
22. Sebastiani, R., Vescovi, M.: Axiom pinpointing in lightweight description logics via horn-sat encoding and conflict analysis. In: Proc. of 22nd Int. Conf. on Automated Deduction. LNCS, vol. 5663, pp. 84–99. Springer (2009)
23. Sebastiani, R., Vescovi, M.: Axiom pinpointing in large $\mathcal{EL}+$ ontologies via sat and smt techniques. Tech. Rep. DISI-15-010, University of Trento, Italy (2015), `http://disi.unitn.it/%7Erseba/elsat/elsat_techrep.pdf`, under submission
24. Simancik, F., Kazakov, Y., Horrocks, I.: Consequence-based reasoning beyond horn ontologies. In: Proc. IJCAI-11. pp. 1093–1098. IJCAI/AAAI (2011)
25. Suntisrivaraporn, B.: Empirical evaluation of reasoning in lightweight DLs on life science ontologies. In: Proc. of MIWAI'08 (2008)
26. Suntisrivaraporn, B.: Polynomial time reasoning support for design and maintenance of large-scale biomedical ontologies. Ph.D. thesis, Dresden University of Technology (2009)