TECHNISCHE UNIVERSITÄT DRESDEN

Technische Universitat Dresden
Institute for Theoretical Computer Science
Chair for Automata Theory

LTCS–Report

Exploiting SAT Technology for Axiom Pinpointing

Norbert Manthey Rafael Peñaloza

LTCS-Report 15-05

Postal Address:
Lehrstuhl fur Automatentheorie
Institut fur Theoretische Informatik
TU Dresden
01062 Dresden

http://lat.inf.tu-dresden.de

Visiting Address:
Nøthnitzer Str. 46
Dresden

Exploiting SAT Technology for Axiom Pinpointing

Norbert Manthey
Knowledge Representation and Reasoning Group
Technische Universität Dresden, Germany
norbert.manthey@tu-dresden.de

Rafael Peñaloza
KRDB Research Centre
Free University of Bozen-Bolzano, Italy
rafael.penaloza@unibz.it

Abstract

Axiom pinpointing is the task of identifying the axioms that are responsible for a consequence. It is a fundamental step for tasks like ontology revision and context-based reasoning, among many others. One known approach is to reduce axiom pinpointing to an enumeration problem over a set of Horn clauses. We introduce the new SATPin system, which combines techniques from axiom pinpointing and minimal unsatisfiable subformula enumeration, and exploits the numerous optimizations developed for SAT solving in the last two decades. By adding a novel optimization method the runtime can improve by a factor up to 4300. Our experiments show that SATPin can find all the MinAs of large biomedical ontologies an order of magnitude faster than existing tools.

1 Introduction

Description logics (DLs) [2] are a family of knowledge representation formalisms that have been successfully employed for modeling large knowledge domains. They are also the logical bases for the standard web ontology language OWL 2 and its profiles.[1] As more and larger ontologies are being built, it becomes necessary to provide automated tools for explaining and correcting unexpected or unwanted consequences.

[1] http://www.w3.org/TR/owl2-overview/

Axiom pinpointing is the task of finding all the axioms that are responsible for a consequence c to follow from a given ontology. This is achieved by finding all the minimal sub-ontologies that still entail c. These sub-ontologies are called *MinAs* or *justifications* in the literature [19, 5, 10]. Knowing these MinAs provides a complete view of what the ontology states about the consequence. For that reason, axiom pinpointing is at the heart of many supplemental reasoning tasks like context-based [3], probabilistic [18], defeasible [7], and error-tolerant reasoning [14], to name just a few.

For the light-weight DL $\mathcal{EL}^+$ [1], one approach suggested in [20] is to translate the problem into an enumeration problem over a Horn formula. In a nutshell, Horn clauses represent the derivation steps of the reasoning algorithm, and the axioms are represented through distinguished variables significantly. One can then use search and unit propagation to find the axioms that, when made true, lead to a successful derivation of the consequence. Such a translation allows us to exploit the numerous optimizations and techniques that make modern SAT solvers so efficient. In particular, we show that axiom pinpointing can be reduced to the enumeration of minimal unsatisfiable subformulas (MUS) of the Horn translation of the ontology.

In this paper we introduce the new SATPin system for axiom pinpointing. SATPin takes advantage of the sophisticated data structures underlying SAT solvers; in particular the two-watched-literal structure [9, 16]. In order to enumerate all the MinAs, several very similar formulas need to be tested for satisfiability. To reduce the overhead of these repeated calls, SATPin uses incremental SAT solving [8]. Moreover, we developed a novel optimization method that reduces the number checks over the distinguished variables.

Our experiments show that SATPin can be effectively used for axiom pinpointing in very large bio-medical ontologies. For Snomed CT, which has almost 400,000 axioms, SATPin was able to solve hard instances that had never been solved before. Compared to the state-of-the-art MUS enumeration system Marco [12], SATPin showed a much better performance in execution time and in memory consumption.

2 Preliminaries

The logic $\mathcal{EL}^+$ is a light-weight DL that allows for polynomial-time reasoning. As all DLs, it is based on *concepts* (unary first-order predicates) and *roles* (binary predicates). Let $\mathsf{N_C}$ and $\mathsf{N_R}$ be two disjoint sets of *concept names* and *role names*, respectively. Complex concepts are built using the grammar rule $C ::= A \mid C \sqcap C \mid \exists r.C \mid \top$, where $A \in \mathsf{N_C}$ and $r \in \mathsf{N_R}$. An $\mathcal{EL}^+$-ontology is a finite set of *axioms* that can be *general concept inclusions* (GCIs) $C \sqsubseteq D$ with C, D two concepts, or *role inclusions* (RIs) $r_1 \circ \cdots \circ r_n \sqsubseteq r$, $n \geq 1$, with $r_i, r \in \mathsf{N_R}$.

Table 1: $\mathcal{EL}^+$ Completion Rules

Precondition: $\mathcal{S} \subseteq \mathcal{T}'$		**Add**: α
$A \sqsubseteq A_i$, $1 \leq i \leq n$	$A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B$	$A \sqsubseteq B$
$A \sqsubseteq \exists r.A_1$	$A_1 \sqcap \exists r.B$	$A \sqsubseteq \exists r.B$
$A \sqsubseteq \exists r.B, B \sqsubseteq B_1$	$\exists r.B_1 \sqsubseteq B_2$	$A \sqsubseteq B_2$
$A_{i-1} \sqsubseteq \exists r_i.A_i$, $1 \leq i \leq n$	$r_1 \sqcap \cdots \sqcap r_n \sqsubseteq r$	$A_0 \sqsubseteq \exists r.A_n$

The semantics of this logic is given by interpretations $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$, where $\Delta^\mathcal{I}$ is a non-empty *domain* and $\cdot^\mathcal{I}$ maps every $A \in \mathsf{N_C}$ to a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$, and every $r \in \mathsf{N_R}$ to a binary relation $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$. This function is extended to arbitrary concepts inductively by setting $\top^\mathcal{I} = \Delta^\mathcal{I}$, $(C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$, and $(\exists r.C)^\mathcal{I} = \{\delta \in \Delta^\mathcal{I} \mid \exists \gamma.(\delta, \gamma) \in r^\mathcal{I}, \gamma \in C^\mathcal{I}\}$. The interpretation $\mathcal{I}$ *satisfies* the GCI $C \sqsubseteq D$ iff $C^\mathcal{I} \subseteq D^\mathcal{I}$, and the RI $r_1 \circ \cdots \circ r_n \sqsubseteq r$ iff $r_1^\mathcal{I} \circ \cdots \circ r_n^\mathcal{I} \subseteq r^\mathcal{I}$. $\mathcal{I}$ a *model* of the ontology $\mathcal{T}$ if it satisfies all axioms in $\mathcal{T}$.

The main reasoning problem in $\mathcal{EL}^+$ is *atomic subsumption*: given two concept names $A, B \in \mathsf{N_C}$ and an ontology $\mathcal{T}$, decide whether $A^\mathcal{I} \subseteq B^\mathcal{I}$ holds for all models $\mathcal{I}$ of $\mathcal{T}$. In this case, we denote it as $\mathcal{T} \models A \sqsubseteq B$. Subsumption between two concept names can be decided in $\mathcal{EL}^+$ using a completion-based algorithm [1]. Completion algorithms, also called *ground tableaux* [4], work in two phases. The first phase, called *normalization*, transforms the ontology into an equivalent one, where all the axioms have a restricted simplified shape. In $\mathcal{EL}^+$ GCIs in normal form are $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B$, $\exists r.A \sqsubseteq B$, and $A \sqsubseteq \exists r.B$, where $n \geq 1$, $A_i, A, B \in \mathsf{N_C}$, and $r \in \mathsf{N_R}$; all RIs are in normal form. To achieve this, every axiom α is mapped to a set of axioms $\mathsf{NF}(\alpha)$ in normal form. The normalization of the ontology $\mathcal{T}$ is then $\mathsf{NF}(\mathcal{T}) := \bigcup_{\alpha \in \mathcal{T}} \mathsf{NF}(\alpha)$.

In the second phase, called *completion*, the normalized ontology $\mathcal{T}' := \mathsf{NF}(\mathcal{T})$ is saturated through an exhaustive application of *completion rules* of the form $(\mathcal{S}, \alpha)$, where $\mathcal{S} \cup \{\alpha\}$ is a finite set of axioms in normal form. This rule is *applicable* if $\mathcal{S} \subseteq \mathcal{T}'$, and its application adds α to $\mathcal{T}'$. To ensure termination, the rule is only applicable if $\alpha \notin \mathcal{T}'$. The $\mathcal{EL}^+$ completion rules are shown in Table 1. Each rule checks whether all the axioms appearing in the first two columns appear in $\mathcal{T}'$, and if so, adds the axiom from the third column. We denote as $\mathfrak{R}$ the set of all completion rules $(\mathcal{S}, \alpha)$. 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 $A \sqsubseteq B \in \mathsf{c}(\mathcal{T})$.

Rather than just deciding whether $\mathcal{T} \models A \sqsubseteq B$, we are interested in finding the axiomatic causes for this subsumption. Formally, a subontology $\mathcal{M} \subseteq \mathcal{T}$ is a *MinA* for $A \sqsubseteq B$ w.r.t. $\mathcal{T}$ if (i) $\mathcal{M} \models A \sqsubseteq B$ and (ii) for all $\mathcal{S} \subset \mathcal{M}$, $\mathcal{S} \not\models A \sqsubseteq B$. That is, a MinA is a minimal subontology that entails the subsumption. *Axiom-pinpointing* refers to the task of finding all MinAs for a subsumption w.r.t. an ontology.

One approach for solving this problem is to create a Horn formula whose satisfying interpretations can be mapped to subontologies entailing the subsumption relation. Before describing in detail how this propositional formula is constructed, we recall some basic notions of propositional logic.

We 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 *complement* $\overline{x}$ of a positive (resp., negative) literal x is the negative (resp., positive) literal with the same variable as x. The variable v of a literal x is denoted as $\mathsf{var}(x)$. The complement of a set of literals S, denoted by $\overline{S}$ is defined as $\overline{S} := \{\overline{x} \mid x \in S\}$. A clause C is a finite set of literals, and is understood as disjunction of literals. A clause that contains only a single literal is called *unit clause*. *Formulas* are finite multisets of clauses, and are understood as conjunction of clauses. A sequence of literals M is *consistent*, if whenever $x \in M$, then $\overline{x} \notin M$. Whenever convenient, we view consistent sequences M as sets throughout this paper.

An *interpretation* I is a set of literals, such that if $x \in I$, then $\overline{x} \notin I$. The *reduct* $F|_I$ of a formula F with respect to I is the multiset $F|_I := \{C \setminus \overline{I} \mid C \in F, C \cap I = \emptyset\}$. An interpretation I *satisfies* a formula F, if $F|_I = \emptyset$. F is *satisfiable* if there is an interpretation that satisfies it.

Let C_1 and C_2 be clauses and $x \in C_1$ and $\overline{x} \in C_2$. Then the clause $(C_1 \setminus \{x\}) \cup (C_2 \setminus \{\overline{x}\})$, denoted by $C_1 \otimes_x C_2$, is the *resolvent of the clauses* C_1 *and* C_2 *upon the literal* x.

We now show how to compute all MinAs for a consequence w.r.t. an $\mathcal{EL}^+$ TBox using satisfiability solving techniques. For every axiom $\alpha \in \mathcal{T} \cup \mathsf{c}(\mathcal{T})$ we introduce a unique Boolean variable x_α. Using these variables, we build the formula $F_\mathcal{T} := F_\mathsf{n} \cup F_\mathsf{c}$, where

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

Intuitively these formulas describe all the possible causes for an axiom α to appear in $\mathsf{c}(\mathcal{T})$. For a subontology $\mathcal{S} \subseteq \mathcal{T}$, let $X_\mathcal{S} := \{\{x_\alpha\} \mid \alpha \in \mathcal{S}\}$. If an interpretation I satisfies $X_\mathcal{S} \wedge F_\mathcal{T}$, then $\{\alpha \mid x_\alpha \in I\} = \mathcal{S} \cup \mathsf{c}(\mathcal{S})$. 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.

Before showing how to enumerate all the MinAs of a consequence using SAT technology, it is worth noticing that the translation above can be applied to any kind of completion-like algorithm with a bounded number of consequences. Thus, this approach can be used for other logics, or other $\mathcal{EL}^+$ decision methods [11].

UP (CNF formula F, sequence literals M, map $reason$)		
Output: Set of propagated literals, updated map		
UP1	$J := M$	// add all literals of M to J
UP2	**while** $C \in F$ and $C\|_J = \{x\}$	// if there is a unit
UP3	$J := J \cup \{x\}$	// add the literal to J
UP4	$reason(\mathsf{var}(x)) := C$	// update reason information
UP5	**return** J	// return set of literals

Figure 1: The UP procedure, which finds the set of literals that can be propagated w.r.t. F and the literals M.

3 Satisfiability Testing

A major operation in modern SAT solvers is *unit propagation*. A unit clause $C = \{x\}$ can only be satisfied if the literal x occurs in the interpretation. Given a formula F, and a consistent sequence M of literals that is used to initialize propagation, the algorithm in Figure 1 returns the set of all literals (including those in M) that must occur in an interpretation to satisfy $F|_M$. The interpretation J is initialized in line UP1. Next, if there are unit clauses in the current reduct (line UP2), the interpretation J is extended with the corresponding literal. Additionally, the clause C is stored as the reason for this extension (line UP3). If no further unit clauses can be found, the algorithm returns the final interpretation.

The SAT problem consists in deciding whether a formula is satisfiable. Briefly, modern SAT solvers use the following approach [15]. First, unit propagation is performed as long as possible. Afterwards, if there is no *conflict* (i.e., a clause that is falsified by the current interpretation), a *search decision* is applied, and unit propagation is executed again. If a conflict is found, then *conflict analysis* is performed, and a *learned clause* C is generated by resolution and added to the formula. This clause C is used to undo parts of the current partial interpretation in a way that unit propagation can be applied again. If a conflict is detected independently of search decisions, then the formula is found to be unsatisfiable. Otherwise, if all variables of the formula can be assigned a truth value without finding a conflict, then the formula is satisfiable.

With specialized data structures, heuristics and simplification techniques, modern SAT solvers are used as a back-end for many industrial tasks [6].

4 Enumerating Implicants

We now show how the MinAs for a consequence are enumerated using SAT technology. Recall that 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 (denoted as $X_{\mathcal{T}} \wedge F_{\mathcal{T}} \equiv \top$). Given a consequence α, we are interested in enumerating all the minimal subsets $M \subseteq X_{\mathcal{T}}$ such that $M \wedge F_{\mathcal{T}} \wedge \overline{x_\alpha}$ is unsatisfiable (denoted as $M \wedge F_{\mathcal{T}} \wedge \overline{x_\alpha} \equiv \bot$).

Our approach does not depend on the precise shape of the formula $F_{\mathcal{T}}$, but rather in the properties described above. Hence, for the rest of this paper, let F be an arbitrary satisfiable formula, and X a set of propositional variables such that $X \wedge F \equiv \top$. Moreover, let q be a propositional variable such that $X \wedge F \wedge \overline{q} \equiv \bot$, which is $(X \wedge F) \rightarrow q$.

The task of enumerating all minimal subsets M of X such that the given formula is unsatisfiable is strongly related to the task of finding all *minimal unsatisfiable subformulas* (MUS) of a given formula [6]. 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. For finding a single group-MUS, the set X is reduced to a minimal subset M, such that $M \wedge F \wedge \overline{q} \equiv \bot$ still holds. As we are interested in finding *all* MinAs, we solve the *all-group-MUS* problem [17, 12], and enumerate all such minimal sets M. However, we need to solve only a special case of all-group-MUS, since for axiom pinpointing we need each group to contain exactly one unit clause, corresponding to an axiom from the ontology. In its general form, all-group-MUS allows for arbitrary sets of clauses to be grouped together.

In order to enumerate all the MinAs, one could then use a general-purpose all-group-MUS extraction tool [12]. However, the specific properties of the enumeration problem required for axiom pinpointing can be further exploited to improve the performance.

We use *incremental SAT solving* [6] to find the next MinA M. Let F be the working formula. In incremental SAT solving, we can initialize the execution of a SAT solver with a set of *assumption* literals. These assumption literals are satisfied as search decisions before the usual search is performed. In our case, we use the set of activation variables X as assumption literals. When a decision has to be made, the algorithm will first activate one of the variables in X, meaning that a new axiom is added to the MinA. Furthermore, we modify the incremental SAT call such that it checks the implication $(M \wedge F) \rightarrow q$, where $M \subseteq X$ is the set of currently assigned assumption literals. This way, we can interrupt the SAT call as soon as q is implied by the current set M. The corresponding pseudo code is given in Figure 2.

`implies` (formula F, literal q, literal sequence X, map $reason$)		
Output: $\bot$, or set of literals R with $(F \wedge R) \rightarrow q$		
IMP1	$M := \emptyset$	// initialize as empty set
IMP2	**while** $X \neq \emptyset$	// while there are literals
IMP3	$q \in \mathtt{UP}(F, M, reason)$	// check value of q
IMP4	**return** $\mathtt{analyze}(q, reason)$	// reduce candidate
IMP5	$M := M \cup \{x\}$ for some $x \in X$	// add $x \in X$ to M
IMP6	$X := X \setminus \{x\}$	// and remove x from X
IMP7	**return** $\bot$	// return the result

Figure 2: The `implies` procedure, which returns a set of literals $R \subseteq X$ 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	$C := reason(\mathsf{var}(q))$	// find clause that implied q
ANA2	**while** $C \neq \emptyset$	// as long as there are literals left
ANA3	$c \in C, C := C \setminus \{c\}$	// select and erase a literal $c \in C$
ANA4	**if** $reason(\mathsf{var}(c)) \neq \bot$	// if there exists a reason for c
ANA5	$C := C \otimes_c D$	// resolve with this reason
ANA6	**else** $R := R \cup \{\overline{c}\}$	// otherwise add $\overline{c}$ to the result
ANA7	**return** R	// return the result

Figure 3: The `analyze` procedure, which returns a set of literals R that lead to the implication $(F \wedge R) \rightarrow q$.

This procedure finds a set M such that $(M \wedge F) \rightarrow q$. However, M might not be minimal. Such a minimal $M' \subseteq M$ can be obtained checking the implication $(F \wedge M \setminus \{m'\}) \rightarrow q$ for each $m' \in M'$. In SAT solving, M can also be reduced by performing conflict analysis once more [8]. Based on M and the reason clause for q, a subset of literals of M is selected based on resolving all literals from this reason clause away with their reason clauses. This procedure is illustrated in the algorithm in Figure 3, and could also be used to solve specialized group-MUS problems. Starting with the reason clause C for the literal q we perform resolution on all literals of the intermediate resolvents, until there is no more literal that has a reason clause. Hence, the clause contains only variables that have been assigned a truth value as search decision, actually as assumption.

The set R obtained from this conflict analysis is only a candidate for a MinA, which needs to be minimized. 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. The corresponding pseudo code is presented in Figure 4.

So far, we have described how to compute one MinA. We now show how to use this method as a sub-procedure for enumerating all MinAs. The algorithm we propose is presented in Figure 5. First we check whether there is a MinA (line ENU2) and abort if this is not the case. Otherwise, an enumeration object is created, which is responsible for enumerating all candidate subsets M of literals. Details on this major part of the algorithm are presented in Section 4.1.

If R represents a potential MinA, then it is minimized and added to the result set of MinAs S (ENU6–ENU8). This set is also added to the enumeration object, such that this solution is excluded from any future answer (ENU9). If this addition makes no further candidate sets possible (ENU9), or if there are no other candidate sets (ENU11), then the algorithm stops (ENU10 and ENU12). Otherwise, the next candidate set $M \subset X$ is tested (ENU14). Finally, after enumeration stopped, the result set S is returned (ENU15).

The check in ENU14 might not return a new potential MinA R, as there are problems that have only a single MinA. Still, to ensure completeness, for all $r \in R$ the imply check has to be performed for the candidate sets $M := X \setminus \{r\}$.

4.1 Enumerate Candidates

The candidate enumeration is initialized with the set of literals X. As long as there are candidates left, a new $M \subset X$ has to be returned in the above algorithm (ENU13). A naive approach would be to enumerate all subsets of X as candidates. Clearly, this approach is unfeasible as it would need to verify $2^{|X|}$ candidates.

A first improvement is to partition X into the set of relevant literals $V = \mathsf{lits}(S)$ and the remaining literals T. The relevant literals are those that represent axioms that belong to at least one MinA. When the enumeration starts, we do not know which literals are relevant and which not. Hence, T is initialized to contain all the variables in X. Whenever a MinA R is found, V and T are updated accordingly;

`minimize` (formula F, set of literals R, literal q	
Output: Set of literals R' such that $(F \wedge M) \rightarrow q$ and $R' \subseteq R$	
MIN1 **for** $r \in R$	// test all literals
MIN2 **if** $q \in \mathtt{UP}(F, R \setminus \{r\}, reason)$	// is r *necessary*?
MIN3 $R := R \setminus \{r\}$	// remove r
MIN4 **return** R	// return the result

Figure 4: The `minimize` procedure, which returns a minimal set of literals $R' \subseteq R$ 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	$S := \emptyset$	
ENU2	$R := \texttt{implies}(F, q, X, reason)$	// Is there a MinA?
ENU3	**if** $R = \bot$ **then return** $\emptyset$	// there are no MinA
ENU4	**setup** enumerator(X)	// setup enumeration
ENU5	**while** $\top$	// check all candidates
ENU6	**if** $R \neq \bot$	// if there was a MinA
ENU7	$R := \texttt{minimize}(F, R, q)$	// minimize candidate
ENU8	$S := S \cup R$	// add R to set of MinAs
ENU9	**if** enumerator.avoid(R) $= \bot$	// disallow this MinA
ENU10	**break**	// no more MinAs
ENU11	**if** enumerator.hasNext() $= \bot$	// Do other MinAs exist?
ENU12	**break**	// no more MinAs
ENU13	$M :=$ enumerator.next()	// next MinA candidate
ENU14	$R := \texttt{implies}(F, q, M, reason)$	// Is there a MinA?
ENU15	**return** S	// return set of MinAs

Figure 5: The `enumerate` procedure, which returns all minimal sets of literals $R \subseteq X$ that lead to $(F \wedge R) \rightarrow q$.

that is, $V := V \cup R$, and $T := T \setminus R$. Given these sets, the new candidates are the sets $V' \cup T$ for $V' \subseteq V$. Using this approach the number of candidates is bounded by $2^{|\mathsf{lits}(S)|}$. Since $|\mathsf{lits}(S)|$ is typically much smaller than $|X|$ this partition reduces the search space considerably. Based on CDCL SAT solvers, we furthermore ensure that candidates that have been tested already are never tested again. This check is obvious in the naive enumeration, but since the sets T and V change over time, this check is important.

We also apply the Hitting Set Tree (HST) enumeration approach developed for axiom pinpointing in DLs [10]. The idea is that, after one MinA M has been found, one can try to find a new MinA over the set of candidate variables $X \setminus \{m\}$ for every $m \in M$. This guarantees that any new solution found is different from M. By iteratively repeating this approach, one constructs a search tree, where each solution is different from all its predecessors. As in the relevant enumeration, the current set V' is extended with T to form a candidate.

Finally, we also use an idea developed for reducing the search space in the group-MUS-enumeration tool MARCO [12]. Given the candidate $M \subseteq X$, if the imply check of M fails, i.e. $(M \wedge F) \not\rightarrow q$, but $(X \wedge F) \rightarrow q$ holds, then we can 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 $(M' \wedge F) \rightarrow q$. Briefly, if $(M \wedge F) \not\rightarrow q$, then the same

statement holds for any subset of M. Hence, once we found a candidate M that failed the imply check, we store the set $X \setminus M$ and any future candidate has to pick one of these literals. In combination with the relevant enumeration, this set is equal to $V \setminus V'$.

4.2 Implementing The Candidate Enumeration

The enumerator object is realized as SAT solver. Model enumeration based on the CDCL algorithm is simple: once a model I is found, a clause $\overline{I}$ is added to the formula, and the next model is generated. Smaller clauses are possible by using only the decision literals of I, because the other literals of I are implied by these literals. To avoid finding the same MinA twice, the literals of a MinA R are added to the formula as clause $\overline{R}$ as well.

For the naive enumeration, we initialize the solver with the variables of X and enumerate all models, where we only consider satisfied literals X for the next candidate M. For the relevant enumeration, we incrementally add the variables of the last MinA R to the solver. Previously added clauses remain valid: previous candidates are not enumerated twice and previously found MinAs cannot be found again. For the HST enumeration the decision heuristic of the solver is modified. Instead of using the default heuristic, a stack of all found solutions is created and the decision heuristic follows the scheme described above to enumerate all candidates. Finally, the inverse enumeration can be realized by adding the necessary clause $X \setminus M$ to include one of the missing literals.

The algorithm spends most time in the imply check, as all literals $x \in X$ have to be applied to the formula. Depending on the used ontology, X can be very large; for instance, the encoding of SNOMED contains 378579 literals. In the way the algorithm was described, each call to the `implies` method assigns all literals, and afterwards undoes all of them once the current check is completed. As already discussed in the relevant enumeration, the relevant literals $V = \mathsf{lits}(S)$ of all MinAs might be a much smaller subset. Considering again SNOMED, the largest set V found by our experiments contained 88 literals. For each imply check, 378491 ($378579 - 88$) literals could be kept in line IMP1. Instead of initializing $M = \emptyset$, we could 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 actually do not undo and recreate the set M in the routine, but keep the last state and only perform the necessary updates.

From an algorithmic point of view, this implementation method improves the algorithm by two orders of magnitude over SNOMED: 99.98 % of the work are saved in the `implies` routine, and the run time of the tool (theoretically) improves by factor 4300. Hence, maintaining relevant variables makes the difference between being solvable and not being solvable.

Table 2: 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

The *cone-of-influence* reduction of the ontology, which is used in other tools [20] is also implemented on the CNF level in our tool, but deactivated as this optimization turns out to not improve the performance beyond the gain provided by relevant variables.

5 Experimental Evaluation

We implemented the previous algorithms in the tool SATPin, which is based on the SAT solver Minisat 2.2. To test our approach we ran SATPin over four well-known biomedical ontologies written in $\mathcal{EL}^+$, which have been widely used as benchmarks for standard DL reasoners. These are the Gene Ontology (GO), NCI, the $\mathcal{EL}^+$ version of FullGalen, and the 2010 version of Snomed. All computations have been performed with a five hour timeout (18000 seconds) on an Intel Xeon CPU at 2.6 GHz and a memory limit of 6.5 GB. As comparison to state-of-the-art MUS enumeration tools, we use Marco [12].[2]

The ontologies were transformed into a propositional formula as described in Section 2 using *el2sat_all* [20]. Table 2 summarizes the properties of these ontologies and their translation. The first row shows the number of axioms in the original ontology, which is also the number of selection variables used by SATPin. As it can be seen, Snomed is an order of magnitude larger than the other three test ontologies. In fact, one of the main problems when dealing with this ontology is to be able to handle the memory consumption issues effectively.

For each of the three smaller ontologies, we computed all the MinAs for 100 different consequences: 50 randomly chosen consequences, and 50 selected as those where the variable $x_{A \sqsubseteq B}$ appears the most often in $F_{\mathcal{T}}$, which is an indication for them to have the most MinAs. For Snomed, we selected 34 consequences that are known to be problematic for axiom pinpointing, due to the number and size of their MinAs.

We ran SATPin and the MUS enumeration tool Marco on all 334 problems, where SATPin uses the combination of all four enumeration mechanisms as well as the relevant variable selection from Section 4.2. Both systems terminated

[2]The used ontologies, created CNF formulas, used tools including command lines, and the logs of the executed runs are available at `http://goo.gl/CDkvEb`.

Table 3: CPU time (s) required by SATPin and Marco

	Tool	Avg	StDev	Max	Median	p-90
GO	Marco	20.01	26.73	171.79	10.11	42.29
	SATPin	3.74	7.54	46.93	1.16	7.76
NCI	Marco	75.95	184.44	1071.42	13.81	151.91
	SATPin	190.15	990.93	8823.32	0.72	87.57
Gln	Marco	100.94	9.71	163.94	97.59	106.45
	SATPin	7.38	11.80	90.25	3.91	13.83
All	Marco	65.64	112.61	1071.42	21.04	105.80
	SATPin	67.09	576.88	8823.32	2.98	14.75

successfully on the 300 instances corresponding to GO, NCI, and FullGalen. However, Marco ran out of memory on all Snomed instances. Thus we consider only the first 300 tests for our comparison. The results of the execution are summarized in Table 3. All the numbers correspond to CPU time in seconds. The results are separated by ontology, and accumulated at the bottom.

As it can be seen, SATPin clearly outperforms Marco in GO and FullGalen. At first sight, it might seem that Marco behaves much better in the NCI samples, taking in average less than half the time required by SATPin. However, Marco was faster only in 6 out of the 100 NCI instances, and was much slower in most of them. This can be verified by looking at the last two columns of Table 3: the median and 90th-percentile for SATPin over NCI are 0.72s and 87.57s, respectively, while for Marco these numbers grow to 13.81s and 151.91s, respectively. What affects SATPin's average performance are three instances that took over 3000s. Removing these three instances reduces the average time to 50.60s for Marco, and 40.55s. for SATPin. Notice moreover that in all instances Marco consumed at least 3 times as much memory as SATPin. This is perhaps the main reason why Marco could not handle any of the Snomed instances.

The efficiency of SATPin is affected by the branching factor produced by the HST algorithm, and the number of relevant variables used. The plot from Figure 6 confirms this observation. The plot shows the proportional speedup of SATPin w.r.t. Marco against the average MinA size in each experiment. As the average size of the MinAs increases, the improvement shown by SATPin decreases. The size of each dot is proportional to the number of MinAs found in that instance. Clearly, the relative performance of SATPin decreases as the number of MinAs increases.

Our experiments suggest that Marco tends to spend more time trying to decrease the search space for the successive solutions. This overhead is helpful for instances with many large solutions, but is too expensive for simple instances.

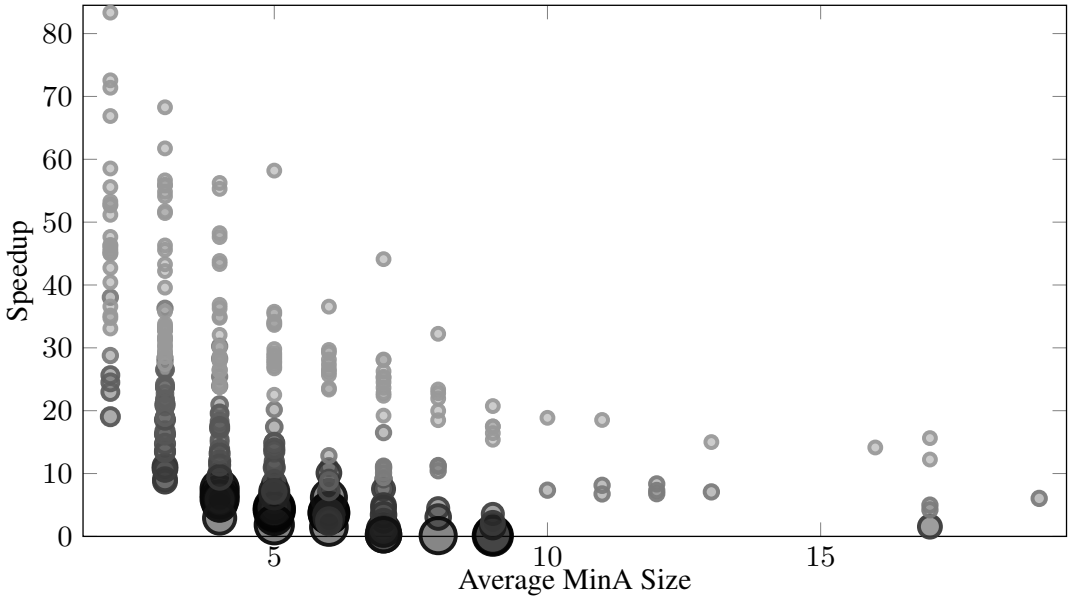

Figure 6: Proportional speedup of SATPin w.r.t. Marco against the average MinA size (horizontal axis) and number of MinAs (node size and tone).

It is worth noticing that real-world ontologies are typically well-structured, and hence their consequences usually have only a few MinAs of small size. This is certainly the case for our experiments, as shown in Table 4.

We did an additional experiment to understand the influence of (i) the order in which the selection variables are provided, and (ii) the variable separation optimization. For this, we took the 20 instances of GO in which SATPin behaved the worst, and ran them again on SATPin with (i) the order of the selection variables reversed, and (ii) with the variable separation optimization deactivated. In the second case, the average CPU time was increased 21 times, from 13 to 279 seconds; in the worst case, it increased from 47 to 1239s. Thus, the optimization is really effective. The theoretical speedup is 681, as at most 30 out of 20466 axioms are present in the MinAs, however, as the assumption literals have to be reordered, this speedup cannot be reached. When the order of the selection variables was reversed, the CPU time variated to up to 2x in both directions:

Table 4: Number and sizes of MinAs found

	#MinAs				#Relevant Axioms		
	Avg	Max	Mdn	MaxSize	Avg	Max	Mdn
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
All	6.50	38	2	19	11.74	43	9

some cases required half the time, while others required double. Regardless of the ordering used, SATPin was still much faster than Marco.

As a stress test, we ran SATPin on 34 consequences of Snomed known to have many large MinAs. In fact, the full set of MinAs for these consequences had never been computed before. As expected, these instances were hard for SATPin, which timed-out in 25 of them. In the nine cases where it succeeded, SATPin found in average 16.4 MinAs (maximum 33) with an average size of 14 axioms each. In the remaining cases, before timing-out it found in average 32 MinAs containing 16 axioms each. In one extreme case, SATPin found 96 MinAs, and the largest of them contained 30 axioms. Recall that these test cases were specially selected for their hardness. In fact, most Snomed consequences have less than 10 MinAs [22].

Discussion We know three other axiom-pinpointing systems for $\mathcal{EL}^+$: Cel [21], $\mathcal{EL}^+$SAT [20], and Just [13]. Despite several efforts, we were not able to execute our test ontologies on either of the two last systems. On the other hand, Cel limits its execution to the computation of 10 MinAs, and at most 1000 seconds. For those reasons, these systems are not included in our evaluation.

6 Conclusions

We have introduced SATPin, a new tool for axiom pinpointing based on the Horn encoding of a reasoning procedure. Since it relies on the formula only, SATPin can be used for axiom pinpointing in any logic that allows for a completion-like reasoning algorithm. It is not restricted to $\mathcal{EL}^+$ or the specific completion rules presented here, as long as a translation is available.

Our experiments show that SATPin can be effectively used for pinpointing in very large practical ontologies. Its median answer time over our experiments was under 3s, and it was capable of answering very hard instances of Snomed. Compared to the MUS enumeration tool Marco, SATPin was at least 10 times faster in more than 65% of all the experiments, and consistently required less than a third of the memory. We have also identified the parts of the system that need to be optimized to improve its performance. These optimizations and parallel enumeration will be in the focus of future work.

We also intend to study the impact of using different translations for the performance of SATPin. Finally, we will extend the approach to provide a better support for supplemental reasoning tasks in DLs and other logics.

References

[1] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proc. IJCAI-05*. Morgan-Kaufmann, 2005.

[2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2nd edition, 2007.

[3] Franz Baader, Martin Knechtel, and Rafael Peñaloza. Context-dependent views to axioms and consequences of semantic web ontologies. *J. of Web Semantics*, 12–13:22–40, 2012.

[4] Franz Baader and Rafael Peñaloza. Axiom pinpointing in general tableaux. *Journal of Logic and Computation*, 20(1):5–34, 2010.

[5] Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn. Pinpointing in the description logic $\mathcal{EL}+$. In *Proc. of the 30th German Conf. on Artif. Intel.*, volume 4667 of *Lecture Notes in Artificial Intelligence*, pages 52–67, Osnabrück, Germany, 2007. Springer-Verlag.

[6] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.

[7] Giovanni Casini, Thomas Meyer, Kodylan Moodley, and Riku Nortje. Relevant closure: A new form of defeasible reasoning for description logics. In *Proc. of 14th European Conf. Logics in Artif. Intel.*, volume 8761 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2014.

[8] Niklas En and Niklas Srensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003. BMC'2003, First International Workshop on Bounded Model Checking.

[9] Ian P. Gent. Optimal implementation of watched literals and more general techniques. *Journal of Artificial Intelligence Research*, 48:231–251, 2013.

[10] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In *Proc. of 6th Int. Semantic Web Conf.*, volume 4825 of *Lecture Notes in Computer Science*, pages 267–280. Springer, 2007.

[11] Yevgeny Kazakov, Markus Krötzsch, and Frantisek Simancik. The incredible ELK - from polynomial procedures to efficient reasoning with ontologies. *J. Autom. Reasoning*, 53(1):1–61, 2014.

[12] Mark H. Liffiton and Ammar Malik. Enumerating infeasibility: Finding multiple MUSes quickly. In Carla P. Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lecture Notes in Computer Science*, pages 160–175. Springer, 2013.

[13] Michel Ludwig. Just: a tool for computing justifications w.r.t. el ontologies. In *Proc. of the 3rd Int. Workshop on OWL Reasoner Evaluation (ORE 2014)*, volume 1207, pages 1–7. CEUR Workshop Proceedings, 2014.

[14] Michel Ludwig and Rafael Peñaloza. Error-tolerant reasoning in the description logic el. In Eduardo Fermé and João Leite, editors, *Proc. of the 14th European Conf. on Logics in Artificial Intelligence (JELIA'14)*, volume 8761 of *Lecture Notes in Artificial Intelligence*, pages 107–121. Springer-Verlag, 2014.

[15] Joo P. Marques-Silva and Karem A. Sakallah. GRASP – a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM international conference on computer-aided design*, ICCAD '96, pages 220–227. IEEE Computer Society, Washington, DC, USA, 1996.

[16] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 530–535. ACM, New York, NY, USA, 2001.

[17] Alessandro Previti and João Marques-Silva. Partial MUS enumeration. In Marie des Jardins and Michael L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*. AAAI Press, 2013.

[18] Fabrizio Riguzzi, Elena Bellodi, Evelina Lamma, and Riccardo Zese. BUNDLE: A reasoner for probabilistic ontologies. In *Proc. of 7th Int. Conf. Web Reasoning and Rule Systems*, volume 7994 of *Lecture Notes in Computer Science*, pages 183–197. Springer, 2013.

[19] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proc. of 18th Int. Joint Conf. on Artif. Intel.*, pages 355–362. Morgan Kaufmann, 2003.

[20] Roberto Sebastiani and Michele Vescovi. Axiom pinpointing in lightweight description logics via horn-sat encoding and conflict analysis. In *Proc. of 22nd Int. Conf. on Automated Deduction*, volume 5663 of *Lecture Notes in Computer Science*, pages 84–99. Springer, 2009.

[21] Boontawee Suntisrivaraporn. Empirical evaluation of reasoning in lightweight DLs on life science ontologies. In *Proceedings of the 2nd Mahasarakham International Workshop on AI (MIWAI'08)*, 2008.

[22] Boontawee Suntisrivaraporn. *Polynomial time reasoning support for design and maintenance of large-scale biomedical ontologies.* PhD thesis, Dresden University of Technology, 2009.