Axiom Pinpointing in Lightweight Description Logics via Horn-SAT Encoding and Conflict Analysis ⋆

Roberto Sebastiani and Michele Vescovi

DISI, Università di Trento, Italy.
{rseba,vescovi}@disi.unitn.it

Abstract. The recent quest for tractable logic-based languages arising from the field of bio-medical ontologies has raised a lot of attention on *lightweight* (i.e. less expressive but tractable) description logics, like $\mathcal{EL}$ and its family. To this extent, automated reasoning techniques in these logics have been developed for computing not only concept subsumptions, but also to pinpoint the set of axioms causing each subsumption. In this paper we build on previous work from the literature and we propose and investigate a simple and novel approach for axiom pinpointing for the logic $\mathcal{EL}^+$. The idea is to encode the classification of an ontology into a Horn propositional formula, and to exploit the power of Boolean Constraint Propagation and Conflict Analysis from modern SAT solvers to compute concept subsumptions and to perform axiom pinpointing. A preliminary empirical evaluation confirms the potential of the approach.

1 Motivations and goals

The recent quest for tractable logic-based languages arising from the field of bio-medical ontologies has attracted a lot of attention on *lightweight* (i.e. less expressive but tractable) description logics, like $\mathcal{EL}$ and its family [1,3,5,2]. In particular, the logic $\mathcal{EL}^+$ [3,5] extends $\mathcal{EL}$ and is of particular relevance due to its algorithmic properties and due to its capability of expressing several important and widely-used bio-medical ontologies, such as Snomed-CT [16], NCI [15], GeneOntology [7] and the majority of Galen [12]. In fact in $\mathcal{EL}^+$ not only standard logic problems such as *concept subsumption* (e.g., "is `Amputation-of-Finger` a subconcept of `Amputation-of-Arm` in the ontology Snomed-CT?" [6]), but also more sophisticated logic problems such as *axiom pinpointing* (e.g., "Find a minimal set of axioms in Snomed-CT which are responsible of the fact that `Amputation-of-Finger` is a subconcept of `Amputation-of-Arm`" [6]) are tractable. Importantly, the problem of axiom pinpointing in $\mathcal{EL}^+$ is of great interest for debugging complex bio-medical ontologies (see, e.g., [6]). To this extent, the problems of concept subsumption and axiom pinpointing in $\mathcal{EL}^+$ have been thoroughly investigated, and efficient algorithms

⋆ The first author is partly supported by SRC under GRC Custom Research Project 2009-TJ-1880 WOLFLING, and by MIUR under PRIN project 20079E5KM8_002.

for these two functionalities have been implemented and tested with success on large ontologies, including SNOMED-CT (see e.g. [3,5,6]).

In this paper we build on previous work from the literature of $\mathcal{EL}^+$ reasoning [3,5,6] and of SAT/SMT [11,17,8,9], and propose a simple and novel approach for (concept subsumption and) axiom pinpointing in $\mathcal{EL}^+$ (and hence in its sub-logics $\mathcal{EL}$ and $\mathcal{ELH}$). In a nutshell, the idea is to generate *polynomial-size* Horn propositional formulas representing part or all the deduction steps performed by the classification algorithms of [3,5], and to manipulate them by exploiting the functionalities of modern conflict-driven SAT/SMT solvers —like *Boolean Constraint Propagation (BCP)* [11], *conflict analysis under assumptions* [11,8], and *all-SMT* [9]. In particular, we show that from an ontology $\mathcal{T}$ it is possible to generate in polynomial time Horn propositional formulas $\phi_\mathcal{T}$, $\phi_\mathcal{T}^{one}$ and $\phi_{\mathcal{T}(po)}^{all}$ of increasing size s.t., for every pair of primitive concepts C_i, D_i:
(i) concept subsumption is performed by one run of BCP on $\phi_\mathcal{T}$ or $\phi_\mathcal{T}^{one}$;
(ii) *one* (non-minimal) set of axioms responsible for the derivation of $C_i \sqsubseteq_\mathcal{T} D_i$ (nMinA) is computed by one run of BCP and conflict analysis on $\phi_\mathcal{T}^{one}$ or $\phi_{\mathcal{T}(po)}^{all}$;
(iii) *one minimal* such set (MinA) is computed by iterating process (ii) on $\phi_{\mathcal{T}(po)}^{all}$ for an amount of time up-to-linear in the size of the first nMinA found;
(iv) the same task of (iii) can also be computed by iteratively applying process (ii) on an up-to-linear sequence of increasingly-smaller formulas $\phi_\mathcal{T}^{one}$,$\phi_{\mathcal{S}_1}^{one}$,...$\phi_{\mathcal{S}_k}^{one}$;
(v) *all* MinAs can be enumerated by means of all-SMT techniques on $\phi_{\mathcal{T}(po)}^{all}$, using step (iii) as a subroutine.
It is worth noticing that (i) and (ii) are instantaneous even with huge $\phi_\mathcal{T}$, $\phi_\mathcal{T}^{one}$ and $\phi_{\mathcal{T}(po)}^{all}$, and that (v) requires building a *polynomial-size* formula $\phi_{\mathcal{T}(po)}^{all}$, in contrast to the exponential-size formula required by the *all-MinAs* process of [5].

We have implemented a prototype tool and performed a preliminary empirical evaluation on the available ontologies, whose results confirm the potential of our novel approach. For lack of space we omit many details and optimizations of our procedures, which can be found in an extended version of this paper [14].

2 Background

2.1 Classification, Subsumption and Axiom Pinpointing in $\mathcal{EL}^+$

The Logic $\mathcal{EL}^+$. The description logic $\mathcal{EL}^+$ belongs to the $\mathcal{EL}$ family, a group of lightweight description logics which allow for conjunctions, existential restrictions and support TBox of GCIs (general concept inclusions) [3]; $\mathcal{EL}^+$ extends $\mathcal{EL}$ adding *complex role inclusion axioms*. In more details, the *concept descriptions* in $\mathcal{EL}^+$ are inductively defined through the constructors listed in the upper half of Table 1, starting from a set of *primitive concepts* and a set of *primitive roles*. (We use the uppercase letters X, X_i, Y, Y_i, to denote generic concepts, the uppercase letters C, C_i, D, D_i, E, E_i to denote concept names and the lowercase letters r, r_i, s to denote role names.) An $\mathcal{EL}^+$ *TBox* (or *ontology*) is a finite set of general concept inclusion (GCI) and role inclusion (RI) axioms as defined in the lower half of Table 1. Given a TBox $\mathcal{T}$, we denote with $\mathsf{PC}_\mathcal{T}$ the

	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
conjunction	$X \sqcap Y$	$X^{\mathcal{I}} \cap Y^{\mathcal{I}}$
existential restriction	$\exists r.X$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \wedge y \in X^{\mathcal{I}}\}$
general concept inclusion	$X \sqsubseteq Y$	$X^{\mathcal{I}} \subseteq Y^{\mathcal{I}}$
role inclusion	$r_1 \circ \cdots \circ r_n \sqsubseteq s$	$r_1^{\mathcal{I}} \circ \cdots \circ r_k^{\mathcal{I}} \subseteq s^{\mathcal{I}}$

Table 1. Syntax and semantics of $\mathcal{EL}^+$.

set of the *primitive concepts* for $\mathcal{T}$, i.e. the smallest set of concepts containing: (i) the top concept $\top$; (ii) all concept names used in $\mathcal{T}$. We denote with $\mathsf{PR}_{\mathcal{T}}$ the set of the *primitive roles* for $\mathcal{T}$, i.e. the set of all the role names used in $\mathcal{T}$. We use the expression $X \equiv Y$ as an abbreviation of the two GCIs $X \sqsubseteq Y$ and $Y \sqsubseteq X$.

The semantics of $\mathcal{EL}^+$ is defined in terms of *interpretations*. An interpretation $\mathcal{I}$ is a couple $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the domain, i.e. a non-empty set of individuals, and $\cdot^{\mathcal{I}}$ is the interpretation function which maps each concept name C to a set $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and maps each role name r to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. In the right-most column of Table 1 the inductive extensions of $\cdot^{\mathcal{I}}$ to arbitrary concept descriptions are defined. An interpretation $\mathcal{I}$ is a *model* of a given TBox $\mathcal{T}$ if and only if the conditions in the Semantics column of Table 1 are respected for every GCI and RI axiom in $\mathcal{T}$. A TBox $\mathcal{T}'$ is a *conservative extension* of the TBox $\mathcal{T}$ if every model of $\mathcal{T}'$ is also a model of $\mathcal{T}$, and every model of $\mathcal{T}$ can be extended to a model of $\mathcal{T}'$ by appropriately defining the interpretations of the additional concept and role names.

A concept Y *subsumes* a concept X w.r.t. the TBox $\mathcal{T}$, written $X \sqsubseteq_{\mathcal{T}} Y$, iff $X^{\mathcal{I}} \subseteq Y^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$. The computation of all subsumption relations between concept names occurring in $\mathcal{T}$ is called *classification* of $\mathcal{T}$. Subsumption and classification in $\mathcal{EL}^+$ can be performed in polynomial time [1,5].

Normalization. In $\mathcal{EL}^+$ it is convenient to establish and work with a *normal form* of the input problem, which helps to make explanations, proofs, reasoning rules and algorithms simpler and more general. Usually the following normal form for the $\mathcal{EL}^+$ TBoxes is considered [1,3,4,5]:

$$(C_1 \sqcap ... \sqcap C_k) \sqsubseteq D, \quad k \geq 1 \qquad C \sqsubseteq \exists r.D \qquad \exists r.C \sqsubseteq D \tag{1}$$

$$r_1 \circ \cdots \circ r_n \sqsubseteq s, \quad n \geq 1 \tag{2}$$

s.t. $C_1, ..., C_k, D \in \mathsf{PC}_{\mathcal{T}}$ and $r_1, ..., r_n, s \in \mathsf{PR}_{\mathcal{T}}$. A TBox $\mathcal{T}$ can be turned into a normalized TBox $\mathcal{T}'$ that is a conservative extension of $\mathcal{T}$ [1], by introducing new concept names. In a nutshell, normalization consists in substituting all instances of complex concepts of the forms $\exists r.C$ and $C_1 \sqcap ... \sqcap C_k$ with fresh concept names (namely, C' and C''), and adding the axioms $C' \sqsubseteq \exists r.C$ [resp. $\exists r.C \sqsubseteq C'$] and $C'' \sqsubseteq C_1, ..., C'' \sqsubseteq C_k$ [resp. $(C_1 \sqcap ... \sqcap C_k) \sqsubseteq C''$] for every substitution in the right [resp. left] part of an axiom. This transformation can be done in linear time and the size of $\mathcal{T}'$ is linear w.r.t. that of $\mathcal{T}$ [1]. We call *normal concept* of

Subsumption assertions ($\ldots \in \mathcal{A}$)		**TBox's axioms** ($\ldots \in \mathcal{T}$)	... added to $\mathcal{A}$
$X \sqsubseteq C_1$, $X \sqsubseteq C_2$, ... $X \sqsubseteq C_k$	$k \geq 1$	$C_1 \sqcap \cdots \sqcap C_k \sqsubseteq D$	$X \sqsubseteq D$
$X \sqsubseteq C$		$C \sqsubseteq \exists r.D$	$X \sqsubseteq \exists r.D$
$X \sqsubseteq \exists r.E$, $E \sqsubseteq C$		$\exists r.C \sqsubseteq D$	$X \sqsubseteq D$
$X \sqsubseteq \exists r.D$		$r \sqsubseteq s$	$X \sqsubseteq \exists s.D$
$X \sqsubseteq \exists r_1.E_1$, ..., $E_{n-1} \sqsubseteq \exists r_n.D$	$n \geq 1$	$r_1 \circ \cdots \circ r_n \sqsubseteq s$	$X \sqsubseteq \exists s.D$

Table 2. Completion rules of the concept subsumption algorithm for $\mathcal{EL}^+$. A rule reads as follows: if the assertions/axioms in the left column belong to $\mathcal{A}$, the GCI/RI of the central column belongs to $\mathcal{T}$, and the assertion of the right column is not already in $\mathcal{A}$, then the assertion of the right column is added to $\mathcal{A}$.

a normal TBox $\mathcal{T}'$ every non-conjunctive concept description occurring in the concept inclusions of $\mathcal{T}'$; we call $\mathsf{NC}_{\mathcal{T}'}$ the set of all the normal concepts of $\mathcal{T}'$.

Concept subsumption in $\mathcal{EL}^+$. Given a normalized TBox $\mathcal{T}$ over the set of primitive concepts $\mathsf{PC}_{\mathcal{T}}$ and the set of primitive roles $\mathsf{PR}_{\mathcal{T}}$, the subsumption algorithm for $\mathcal{EL}^+$ [5] generates and extends a set $\mathcal{A}$ of *assertions* through the completion rules defined in Table 2. (By "*assertion*" we mean every known or deduced subsumption relation between normal concepts of the TBox $\mathcal{T}$.) The algorithm starts with the initial set $\mathcal{A} = \{a_i \in \mathcal{T} | a_i \text{ is a GCI}\} \cup \{C \sqsubseteq C | C \in \mathsf{PC}_{\mathcal{T}}\} \cup \{C \sqsubseteq \top | C \in \mathsf{PC}_{\mathcal{T}}\}$ and extends $\mathcal{A}$ using the rules of Table 2 until no more assertions can be added. (Notice that a rule is applied only if it extends $\mathcal{A}$.)

In [1] the soundness and completeness of the algorithm are proved, together with the fact that the algorithm terminates after polynomially-many rule applications, each of which can be performed in polynomial time.

Once a complete classification of the normalized TBox is computed and stored in some ad-hoc data structure, if $C, D \in \mathsf{PC}_{\mathcal{T}}$, then $C \sqsubseteq_{\mathcal{T}} D$ iff the pair C, D can be retrieved from the latter structure. The problem of computing $X \sqsubseteq_{\mathcal{T}} Y$ s.t. $X, Y \notin \mathsf{PC}_{\mathcal{T}}$ can be reduced to that of computing $C \sqsubseteq_{\mathcal{T} \cup \{C \sqsubseteq X, Y \sqsubseteq D\}} D$, s.t. C and D are two new concept names.

Axiom Pinpointing in $\mathcal{EL}^+$. We consider $C_i, D_i \in \mathsf{PC}_{\mathcal{T}}$ s.t. $C_i \sqsubseteq_{\mathcal{T}} D_i$. We call $\mathcal{S}$ s.t. $\mathcal{S} \subseteq \mathcal{T}$ a (possibly non-minimal) *axiom set for $\mathcal{T}$ wrt. $C_i \sqsubseteq D_i$*, written *nMinA*, if $C_i \sqsubseteq_{\mathcal{S}} D_i$; we call an nMinA $\mathcal{S}$ a *minimal axiom set for $C_i \sqsubseteq D_i$*, written *MinA*, if $C_i \not\sqsubseteq_{\mathcal{S}'} D_i$ for every $\mathcal{S}'$ s.t. $\mathcal{S}' \subset \mathcal{S}$.

Baader et al. [5] proposed a technique for computing all MinAs for $\mathcal{T}$ wrt. $C_i \sqsubseteq_{\mathcal{T}} D_i$, which is based on building from a classification of $\mathcal{T}$ a *pinpointing formula* (namely $\Phi^{C_i \sqsubseteq_{\mathcal{T}} D_i}$), which is a monotone propositional formula on the set of propositional variables $\mathcal{P}_{\mathcal{T}} \stackrel{\text{def}}{=} \{s_{[ax_j]} | ax_j \in \mathcal{T}\}$ s.t., for every $\mathcal{O} \subseteq \mathcal{T}$, $\mathcal{O}$ is a MinA wrt. $C_i \sqsubseteq_{\mathcal{T}} D_i$ iff $\{s_{[ax_k]} | ax_k \in \mathcal{O}\}$ is a minimal valuation of $\Phi^{C_i \sqsubseteq_{\mathcal{T}} D_i}$. Thus, the all-MinAs algorithm in [5] consists of (i) building $\Phi^{C_i \sqsubseteq_{\mathcal{T}} D_i}$ and (ii) computing all minimal valuations of $\Phi^{C_i \sqsubseteq_{\mathcal{T}} D_i}$. According to [5], however, this algorithm has serious limitations in terms of complexity: first, the algorithm for generating

```
1.   SatValue DPLL (formula φ, assignment μ)
2.       while (1)
3.           while (1)
4.               status = bcp(φ, μ);
5.               if (status == sat)
6.                   return sat;
7.               else if (status == conflict)
8.                   blevel = analyze_conflict(φ, μ);
9.                   if (blevel == 0) return unsat;
10.                  else backtrack(blevel,φ, μ);
11.              else break;
12.          decide_next_branch(φ, μ);
```

Fig. 1. Schema of a conflict-driven DPLL SAT solver.

$\Phi^{C_i \sqsubseteq_{\mathcal{T}} D_i}$ requires intermediate logical checks, each of them involving the solution of an NP-complete problem; second, the size of $\Phi^{C_i \sqsubseteq_{\mathcal{T}} D_i}$ can be exponential wrt. that of $\mathcal{T}$. More generally, [5] proved also that there is no output-polynomial algorithm for computing all MinAs (unless P=NP). (To the best of our knowledge, there is no publicly-available implementation of the all-MinAs algorithm above.) Consequently, [5] concentrated the effort on finding polynomial algorithms for finding *one* MinA at a time, proposing a linear-search minimization algorithm which allowed for finding MinAs for FULL-GALEN efficiently. This technique was further improved in [6] by means of a binary-search minimization algorithm, and by a novel algorithm exploiting the notion of *reachability-modules*, which allowed to find efficiently MinAs for the much bigger SNOMED-CT ontology. We refer the readers to [5,6] for a detailed description.

2.2 Basics on Conflict-Driven SAT Solving

Notation. We assume the standard syntactic and semantic notions of propositional logic (including the standard notions of formula, atom, literal, CNF formula, Horn formula, truth assignment, clause, unit clause). We represent a truth assignment μ as a conjunction of literals $\bigwedge_i l_i$ (or analogously as a set of literals $\{l_i\}_i$) with the intended meaning that a positive [resp. negative] literal p_i means that p_i is assigned to true [resp. false]. Notationally, we often write clauses as implications: "$(\bigwedge_i l_i) \rightarrow (\bigvee_j l_j)$" for "$\bigvee_i \neg l_i \vee \bigvee_j l_j$"; also, if η is a conjunction of literals $\bigwedge_i l_i$, we write $\neg\eta$ for the clause $\bigvee_i \neg l_i$, and vice versa.

Conflict-driven SAT solving. The schema of a modern conflict-driven DPLL SAT solver is shown in Figure 1 [11,17]. The propositional formula φ is in CNF; the assignment μ is initially empty, and it is updated in a stack-based manner.

In the main loop, decide_next_branch(φ, μ) (line 12.) chooses an unassigned literal l from φ according to some heuristic criterion, and adds it to μ. (This operation is called *decision*, l is called *decision literal* and the number of decision literals in μ after this operation is called the *decision level* of l.) In the inner loop, $\texttt{bcp}(\varphi, \mu)$ iteratively deduces literals l from the current assignment and updates φ and μ accordingly; this step is repeated until either μ satisfies φ, or μ falsifies φ, or no more literals can be deduced, returning sat, conflict and unknown respectively. In the first case, DPLL returns sat. In the second case, $\texttt{analyze\_conflict}(\varphi, \mu)$ detects the subset η of μ which caused the conflict (*conflict set*) and the decision level blevel to backtrack. (This process is called *conflict analysis*, and is described in more details below.) If blevel is 0, then a conflict exists even without branching, so that DPLL returns unsat. Otherwise, $\texttt{backtrack}(\texttt{blevel}, \varphi, \mu)$ adds the *blocking clause* $\neg\eta$ to φ (*learning*) and backtracks up to blevel (*backjumping*), popping out of μ all literals whose decision level is greater than blevel, and updating φ accordingly. In the third case, DPLL exits the inner loop, looking for the next decision.

bcp is based on *Boolean Constraint Propagation (BCP)*, that is, the iterative application of *unit propagation*: if a unit clause l occurs in φ, then l is added to μ, all negative occurrences of l are declared false and all clauses with positive occurrences of l are declared satisfied. Current SAT solvers include extremely fast implementations of bcp based on the *two-watched-literal scheme* [11]. Notice that a complete run of bcp requires an amount of steps which is at most linear in the number of clauses containing the negation of some of the propagated literals.

analyze_conflict works as follows (see, e.g., [11,17]). Each literal is tagged with its decision level, that is, the literal corresponding to the nth decision and the literals derived by unit-propagation after that decision are labeled with n; each non-decision literal l in μ is also tagged by a link to the clause ψ_l causing its unit-propagation (called the *antecedent clause* of l). When a clause ψ is falsified by the current assignment —in which case we say that a *conflict* occurs and ψ is the *conflicting clause*— a *conflict clause* ψ' is computed from ψ s.t. ψ' contains only one literal l_u which has been assigned at the last decision level. ψ' is computed starting from $\psi' = \psi$ by iteratively resolving ψ' with the antecedent clause ψ_l of some literal l in ψ' (typically the last-assigned literal in ψ', see [17]), until some stop criterion is met. E.g., with the *Decision Scheme*, ψ' must contain only decision literals, including the last-assigned one.

If φ is a Horn formula, then one single run of bcp is sufficient to decide the satisfiability of φ. In fact, if $\texttt{bcp}(\varphi, \{\})$ returns conflict, then φ is unsatisfiable; otherwise φ is satisfiable because, since all unit clauses have been removed from φ, all remaining clauses contain at least one negative literal, so that assigning all unassigned literals to false satisfies φ.

Conflict-driven SAT solving under assumptions. The schema in Figure 1 can be adapted to check also the satisfiability of a CNF propositional formula φ *under a set of assumptions* $\mathcal{L} \stackrel{def}{=} \{l_1, ..., l_k\}$. (From a purely-logical viewpoint, this corresponds to check the satisfiability of $\bigwedge_{l_i \in \mathcal{L}} l_i \wedge \varphi$.) This works as follows: $l_1, ..., l_k$ are initially assigned to true, they are tagged as decision literals and added to μ, then the decision level is reset to 0 and DPLL enters the external loop. If $\bigwedge_{l_i \in \mathcal{L}} l_i \wedge \varphi$ is consistent, then DPLL returns sat; otherwise, DPLL eventually backtracks up to level 0 and then stops, returning conflict. Importantly, if `analyze_conflict` uses the Decision Scheme mentioned above, then the final conflict clause will be in the form $\bigvee_{l_j \in \mathcal{L}'} \neg l_j$ s.t. $\mathcal{L}'$ is the (possibly much smaller) subset of $\mathcal{L}$ which actually caused the inconsistency revealed by the SAT solver (i.e., s.t. $\bigwedge_{l_j \in \mathcal{L}'} l_j \wedge \varphi$ is inconsistent). In fact, at the very last branch, `analyze_conflict` will iteratively resolve the conflicting clause with the antecedent clauses of the unit-propagated literals until only decision literals are left: since this conflict has caused a backtrack up to level 0, these literals are necessarily all part of $\mathcal{L}$.

This technique is very useful in some situations. First, sometimes one needs checking the satisfiability of a (possibly very big) formula φ under many different sets of assumptions $\mathcal{L}_1, ..., \mathcal{L}_N$. If this is the case, instead of running `DPLL` on $\bigwedge_{l_i \in \mathcal{L}_j} l_i \wedge \varphi$ for every $\mathcal{L}_j$ —which means parsing the formulas and initializing `DPLL` from scratch each time— it is sufficient to parse φ and initialize `DPLL` only once, and run the search under the different sets of assumptions $\mathcal{L}_1, ..., \mathcal{L}_N$. This is particularly important when parsing and initialization times are relevant wrt. solving times. In particular, if φ is a Horn formula, solving φ under assumptions requires only one run of `bcp`, whose computational cost depends linearly only on the clauses where the unit-propagated literals occur.

Second, this technique can be used in association with the use of *selector variables*: all the clauses ψ_i of φ can be substituted by the corresponding clauses $s_i \rightarrow \psi_i$, all s_is being fresh variables, which are initially assumed to be true (i.e., $\mathcal{L} = \{s_i \mid \psi_i \in \varphi\}$). If φ is unsatisfiable, then the final conflict clause will be of the form $\bigvee_{s_j \in \mathcal{L}'} \neg s_j$, s.t. $\{\psi_j \mid s_j \in \mathcal{L}'\}$ is the actual subset of clauses which caused the inconsistency of φ. This technique is used to compute unsatisfiable cores of CNF propositional formulas [10].

3 Axiom Pinpointing via Horn SAT and Conflict Analysis

We assume that $\mathcal{T}$ is the result of a normalization process, as described in §2.1. (We will consider the issue of normalization at the end of §3.2.)

3.1 Classification and Concept Subsumption via Horn SAT solving

We consider the problem of concept subsumption. We build a Horn propositional formula $\phi_{\mathcal{T}}$ representing the classification of the input ontology $\mathcal{T}$. A basic encoding works as follows. For every normalized concept X in $\mathsf{NC}_{\mathcal{T}}$ we introduce exactly one uniquely-associated fresh Boolean variable $p_{[X]}$. We initially set $\phi_{\mathcal{T}}$ to the empty set of clauses. We run the classification algorithm of §2.1: for every non-trivial [1] axiom or assertion a_i of the form (1) which is added to $\mathcal{A}$, we add to $\phi_{\mathcal{T}}$ one clause $\mathcal{EL}^{+}2sat(a_i)$ of the form

$$(p_{[C_1]} \wedge ... \wedge p_{[C_k]}) \rightarrow p_{[D]}, \quad k \geq 1 \qquad p_{[C]} \rightarrow p_{[\exists r.D]} \qquad p_{[\exists r.C]} \rightarrow p_{[D]} \tag{3}$$

[1] We do not encode axioms of the form $C \sqsubseteq C$ and $C \sqsubseteq \top$ because they generate valid clauses $p_{[C]} \rightarrow p_{[C]}$ and $p_{[C]} \rightarrow \top$.

respectively. Notice that (3) are non-unit Horn clauses with one positive literal (hereafter *definite* Horn clauses). It follows straightforwardly that $C \sqsubseteq_{\mathcal{T}} D$ if and only if the Horn formula $\phi_{\mathcal{T}} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable, for every pair of concepts C, D in $\mathsf{PC}_{\mathcal{T}}$. In fact, by construction, the clause $p_{[C]} \rightarrow p_{[D]}$ is in $\phi_{\mathcal{T}}$ if and only if $C \sqsubseteq_{\mathcal{T}} D$. Notice that $\phi_{\mathcal{T}}$ is polynomial wrt. the size of $\mathcal{T}$, since the algorithm of §2.1 terminates after a polynomial number of rule applications. A more compact encoding is described in [14].

Once $\phi_{\mathcal{T}}$ has been generated, in order to perform concept subsumption we exploit the techniques of conflict-driven SAT solving under assumptions described in §2.2: once $\phi_{\mathcal{T}}$ is parsed and `DPLL` is initialized, each subsumption query $C_i \sqsubseteq_{\mathcal{T}} D_i$ corresponds to solving $\phi_{\mathcal{T}}$ under the assumption list $\mathcal{L}_i \stackrel{\text{def}}{=} \{\neg p_{[D_i]}, p_{[C_i]}\}$. This corresponds to one run of `bcp`: since $\phi_{\mathcal{T}}$ contains the clause $p_{[C_i]} \rightarrow p_{[D_i]}$, `bcp` stops as soon as $\neg p_{[D_i]}$ and $p_{[C_i]}$ are unit-propagated.

3.2 Computing single and all MinAs via Conflict Analysis

We consider the general problem of generating MinAs. We build another Horn propositional formula $\phi_{\mathcal{T}}^{all}$ representing the complete classification DAG of the input normalized ontology $\mathcal{T}$. [2] The size of $\phi_{\mathcal{T}}^{all}$ is polynomial wrt. that of $\mathcal{T}$.

Building the formula $\phi_{\mathcal{T}}^{all}$. For every normalized concept X in $\mathsf{NC}_{\mathcal{T}}$ we introduce exactly one uniquely-associated fresh Boolean variable $p_{[X]}$; further (*selector*) Boolean variables will be introduced, through the steps of the algorithm, to uniquely represent axioms and assertions. We initially set $\phi_{\mathcal{T}}^{all}$ to the empty set of clauses. Then we run an extended version of the classification algorithm of §2.1:

1. for every RI axiom a_i we introduce the axiom selector variable $s_{[a_i]}$; for every GCI axiom a_i of the form $C \sqsubseteq C$ or $C \sqsubseteq \top$, $s_{[a_i]}$ is the "true" constant $\top$;
2. for every non-trivial GCI axiom a_i we add to $\phi_{\mathcal{T}}^{all}$ a clause of the form
$$s_{[a_i]} \rightarrow \mathcal{EL}^{+}2sat(a_i) \tag{4}$$
s.t. $s_{[a_i]}$ is the axiom selector variable for a_i and $\mathcal{EL}^{+}2sat(a_i)$ is the encoding in (3);
3. for every application of a rule (namely r) generating some assertion $gen(r)$ (namely a_i) which was not yet present in $\mathcal{A}$ (and thus adding a_i to $\mathcal{A}$), we add to $\phi_{\mathcal{T}}^{all}$ a clause (4) and a clause of the form
$$(\bigwedge_{a_j \in \ ant(r)} s_{[a_j]}) \rightarrow s_{[a_i]} \tag{5}$$
s.t. $s_{[a_i]}$ (that is $s_{[gen(r)]}$) is the selector variable for a_i and $ant(r)$ are the *antecedents* of a_i wrt. rule r (that is, the assertions and the RI or GCI axiom in the left and central columns of Table 2 for rule r respectively);

[2] Here "complete" means "including also the rule applications generating already-generated assertions".

AxiomSet lin-extract-MinA$_{DPLL}$(Concept C_i, D_i, AxiomSet $\mathcal{T}^*$, formula $\phi_{\mathcal{T}}^{all}$)

    $\mathcal{S}$ = $\mathcal{T}^*$;
    for each axiom a_j in $\mathcal{T}^*$
        $\mathcal{L}$ = $\{\neg p_{[D_i]}, p_{[C_i]}\} \cup \{s_{[a_i]} \mid a_i \in \mathcal{S} \setminus \{a_j\}\}$;
        if (DPLLUnderAssumptions($\phi_{\mathcal{T}}^{all}$,$\mathcal{L}$) == unsat)
            $\mathcal{S}$ = $\mathcal{S} \setminus \{a_j\}$;
    return $\mathcal{S}$;

Fig. 2. SAT-based variant of the linear MinA-extracting algorithm in [5].

4. for every application of a rule (namely r) generating some assertion $gen(r)$ (namely a_i) which was already present in $\mathcal{A}$ (and thus not adding a_i to $\mathcal{A}$), we add to $\phi_{\mathcal{T}}^{all}$ only a clause of the form (5).

In order to ensure termination, we perform step 3. and 4. in a queue-based manner, which assures that every possible distinct (i.e. with different antecedents) rule application is applied only once. Following this idea, in [14] we show that the extended algorithm requires a polynomial amount of steps wrt. the size of $\mathcal{T}$ and that $\phi_{\mathcal{T}}^{all}$ is polynomial in the size of $\mathcal{T}$.

Notice that (4) and (5) are definite Horn clauses since all (3) are definite Horn clauses. (We call (4) and (5) *assertion clauses* and *rule clauses* respectively.) Notice also that step 4. is novel wrt. the classification algorithm of §2.1.

It follows straighforwardly that, for every $\mathcal{S} \subseteq \mathcal{T}$ and for every pair of concepts C, D in $\mathsf{PC}_{\mathcal{T}}$, $C \sqsubseteq_{\mathcal{S}} D$ if and only if $\phi_{\mathcal{T}}^{all} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable. In fact, $C \sqsubseteq_{\mathcal{S}} D$ if and only if there exists a sequence or rule applications $r_1, ..., r_k$ generating $C \sqsubseteq D$ from $\mathcal{S}$. If (and only if) this is the case, by construction $\phi_{\mathcal{T}}^{all}$ contains the clause $s_{[C \sqsubseteq D]} \rightarrow (p_{[C]} \rightarrow p_{[D]})$ (4) and all the clauses (5) corresponding to all rule applications $r \in \{r_1, ..., r_k\}$. This means that $\bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge p_{[C]} \wedge \neg p_{[D]}$ forces the unit-propagation of $s_{[C \sqsubseteq D]}$, $p_{[C]}$ and $\neg p_{[D]}$, which falsify the clause (4) above. (See [14] for details.)

Computing one MinA. Once $\phi_{\mathcal{T}}^{all}$ is generated, in order to compute one MinA, we can exploit the techniques of conflict-driven SAT solving under assumptions described in §2.2. After $\phi_{\mathcal{T}}^{all}$ is parsed and DPLL is initialized, each query $C_i \sqsubseteq_{\mathcal{T}} D_i$ corresponds to solving $\phi_{\mathcal{T}}^{all}$ under the assumption list $\mathcal{L}_i \stackrel{\text{def}}{=} \{\neg p_{[D_i]}, p_{[C_i]}\} \cup \{s_{[a_i]} \mid a_i \in \mathcal{T}\}$. This corresponds to a single run of bcp and one run of analyze_conflict, whose cost depends linearly only on the clauses where the unit-propagated literals occur. (Actually, if bcp does not return conflict, then sat is returned without even performing conflict analysis.) If bcp returns conflict, as explained in §2.2, then analyze_conflict produces a conflict clause $\psi_{\mathcal{T}^*}^{C_i, D_i} \stackrel{\text{def}}{=} p_{[D_i]} \vee \neg p_{[C_i]} \vee \bigvee_{a_i \in \mathcal{T}^*} \neg s_{[a_i]}$ s.t. $\mathcal{T}^*$ is an nMinA wrt. $C_i \sqsubseteq_{\mathcal{T}} D_i$. In fact, the presence of both $\neg p_{[D_i]}$ and $p_{[C_i]}$ in $\mathcal{L}_i$ is necessary for causing the conflict, so that, due to the Decision Scheme, the conflict set necessarily contains both of them.

Notice that $\mathcal{T}^*$ may not be minimal. In order to minimize it, we can apply the SAT-based variant of the linear minimization algorithm of [5] in Figure 2. (We assume that $\phi^{all}_{\mathcal{T}}$ has been parsed and DPLL has been initialized, and that $\phi^{all}_{\mathcal{T}}$ has been solved under the assumption list $\mathcal{L}_i$ above, producing the conflict clause $\psi^{C_i,D_i}_{\mathcal{T}^*}$ and hence the nMinA $\mathcal{T}^*$; then `lin-extract-MinA`$_{DPLL}$`(`C_i, D_i`,` $\mathcal{T}^*$`,` $\phi^{all}_{\mathcal{T}}$`)` is invoked.) In a nutshell, the algorithm tries to remove one-by-one the axioms a_js in $\mathcal{T}^*$, each time checking whether the reduced axiom set $\mathcal{S} \setminus \{a_j\}$ is still such that $C_i \sqsubseteq_{S\setminus\{a_j\}} D_i$. As before, each call to DPLLUnderAssumptions requires only one run of bcp. This schema can be improved as follows: if DPLLUnderAssumptions performs also conflict analysis and returns (the conflict clause corresponding to) an nMinA $\mathcal{S}'$ s.t. $\mathcal{S}' \subset \mathcal{S} \setminus \{a_i\}$, then $\mathcal{S}$ is assigned to $\mathcal{S}'$ and all axioms in $(\mathcal{S} \setminus \{a_j\}) \setminus \mathcal{S}'$ will not be selected in next loops. As an alternative choice, one can implement instead (a SAT-based version of) the binary-search variant of the minimization algorithm (see e.g. [6]).

It is important to notice that the formula $\phi^{all}_{\mathcal{T}}$ is never updated: in order to check $C_i \sqsubseteq_{S\setminus\{a_j\}} D_i$, it suffices to drop $s_{[a_j]}$ from the assumption list. The latter fact makes (the encoding of) the axiom a_j useless for bcp to falsify the clause encoding $C_i \sqsubseteq_{\mathcal{T}} D_i$, so that DPLLUnderAssumptions returns unsat if and only if a different falsifying chain of unit-propagations can be found, corresponding to a different sequence of rule applications generating $C_i \sqsubseteq_{\mathcal{T}} D_i$. Notice that this fact is made possible by step 4. of the encoding, which allows for encoding all alternative sequences of rule applications generating the same assertions.

We also notice that one straightforward variant to this technique, which is feasible since typically $|\mathcal{T}^*| \ll |\mathcal{T}|$, is to compute another formula $\phi^{all}_{\mathcal{T}^*}$ from scratch and to feed it to the algorithm of Figure 2 instead of $\phi^{all}_{\mathcal{T}}$.

One very important remark is in order. During pinpointing the only clause of type (4) in $\phi^{all}_{\mathcal{T}}$ which is involved in the conflict analysis process is $s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]} \rightarrow (p_{[C_i]} \rightarrow p_{[D_i]})$, which reduces to the unit clause $\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]}$ after the unit-propagation of the assumption literals $\neg p_{[D_i]}, p_{[C_i]}$. Thus, one may want to decouple pinpointing from classification/subsumption, and produce a reduced "pinpointing-only" version of $\phi^{all}_{\mathcal{T}}$, namely $\phi^{all}_{\mathcal{T}(po)}$. The encoding of $\phi^{all}_{\mathcal{T}(po)}$ works like that of $\phi^{all}_{\mathcal{T}}$, except that no clause (4) is added to $\phi^{all}_{\mathcal{T}(po)}$. Thus each query $C_i \sqsubseteq_{\mathcal{T}} D_i$ corresponds to solving $\phi^{all}_{\mathcal{T}(po)}$ under the assumption list $\mathcal{L}_i \stackrel{\text{def}}{=} \{\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]}\} \cup \{s_{[a_i]} \,|a_i \in \mathcal{T}\}$, so that the algorithm for pinpointing is changed only in the fact that $\phi^{all}_{\mathcal{T}(po)}$ and $\{\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]}\}$ are used instead of $\phi^{all}_{\mathcal{T}}$ and $\{\neg p_{[D_i]}, p_{[C_i]}\}$ respectively. Thus, w.l.o.g. in the remaining part of this section we will reason using $\phi^{all}_{\mathcal{T}(po)}$ and $\{\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]}\}$. (The same results, however, can be obtained using $\phi^{all}_{\mathcal{T}}$ and $\{\neg p_{[D_i]}, p_{[C_i]}\}$ instead.)

Computing all MinAs. We describe a way of generating *all* MinAs wrt. $C_i \sqsubseteq_{\mathcal{T}} D_i$ from $\phi^{all}_{\mathcal{T}(po)}$ and $\{\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]}\}$. In a nutshell, the idea is to assume $\{\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]}\}$ and to enumerate all possible minimal truth assignments on the axiom selector variables in $\mathcal{P}_{\mathcal{T}} \stackrel{\text{def}}{=} \{s_{[ax_j]} \,|ax_j \in \mathcal{T}\}$ which cause the inconsistency of the formula $\phi^{all}_{\mathcal{T}(po)}$. This is implemented by means of a variant of the all-SMT technique in [9]. A naive version of this technique is described as follows.

We consider a propositional CNF formula φ on the variables in $\{s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]}\} \cup \mathcal{P}_{\mathcal{T}}$. φ is initially set to $\top$. One top-level instance of DPLL (namely DPLL1) is used to enumerate a complete set of truth assignments $\{\mu_k\}_k$ on the axiom selector variables in $\mathcal{P}_{\mathcal{T}}$ which satisfy φ under the assumption of $\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]}$. Every time that a novel assignment μ_k is generated, $\{\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]}\} \cup \mu_k$ is passed to an ad-hoc "$\mathcal{T}$-solver" checking whether it causes the inconsistency of the formula $\phi^{all}_{\mathcal{T}(po)}$. If this is the case, then the $\mathcal{T}$-solver returns conflict and a minimal subset $\{\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]}\} \cup \{s_{[ax_j]} \mid ax_j \in \mathcal{T}^*_k\}$, s.t. $\mathcal{T}^*_k$ is a MinA, which caused such inconsistency. $\psi^*_k \stackrel{\text{def}}{=} s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]} \vee \bigvee_{ax_j \in \mathcal{T}^*_k} \neg s_{[ax_j]}$ is then added to φ as a blocking clause and it is used as a conflict clause for driving next backjumping step. Otherwise, $\mathcal{T}$-solver returns sat, and DPLL1 uses $s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]} \vee \neg\mu_k$ as a "fake" conflict clause, which is added to φ as a blocking clause and is used as a conflict clause for driving next backjumping step. The process terminates when `backtrack` back-jumps to `blevel` zero. The set of all MinAs $\mathcal{T}^*_k$ are returned as output.

The $\mathcal{T}$-solver is the procedure described in the previous paragraph "Compute one MinA" (with $\phi^{all}_{\mathcal{T}(po)}$, $\{\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]}\}$ instead of $\phi^{all}_{\mathcal{T}}$, $\{\neg p_{[D_i]}, p_{[C_i]}\}$), using a second instance of DPLL, namely DPLL2. As before, we assume $\phi^{all}_{\mathcal{T}(po)}$ is parsed and DPLL2 is initialized only once, before the whole process starts.

In [14] we explain this naive procedure with more details and show that it returns all MinAs wrt. $C_i \sqsubseteq_{\mathcal{T}} D_i$.

One important improvement to the naive procedure above is that of exploiting *early pruning* and *theory propagation*, two well-known techniques from SMT (see, e.g., [13]). The $\mathcal{T}$-solver is invoked also on partial assignments μ_k on $\mathcal{P}_{\mathcal{T}}$: if this causes the unit-propagation of one (or more) $\neg s_{[ax_j]}$ s.t. $s_{[ax_j]} \in \mathcal{P}_{\mathcal{T}}$ and $s_{[ax_j]}$ is unassigned, then the antecedent clause of $\neg s_{[ax_j]}$ can be fed to `analyze_conflict` in DPLL2, which returns the clause $s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]} \vee \neg\mu'_k$ s.t. $\mu'_k \subseteq \mu_k$ and $\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]} \wedge \mu'_k$ causes the propagation of $\neg s_{[ax_j]}$. (As before, we assume that `analyze_conflict` uses the Decision Scheme.) Intuitively, this is equivalent to say that, if $\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]} \wedge \mu_k \wedge s_{[ax_j]}$ is passed to the $\mathcal{T}$-solver, then it would return conflict and the $\mathcal{T}$-conflict clause $\psi^*_k \stackrel{\text{def}}{=} s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]} \vee \neg\mu'_k \vee \neg s_{[ax_j]}$. Thus $\mu'_k \wedge s_{[ax_i]}$ represents a non-minimal set of axioms causing the inconsistency of $\phi^{all}_{\mathcal{T}(po)}$, which can be further minimized by the algorithm of Figure 2, as described above.

One problem of the naive procedure above, regardless of early pruning and theory propagation, is that adding to φ a "fake" blocking clause (namely $\neg\eta_k$) each time a new satisfying truth assignment η_k is found may cause an exponential blowup of φ. As shown in [9], this problem can be overcome by exploiting conflict analysis techniques. Each time a model η_k is found, it is possible to consider $\neg\eta_k$ as a conflicting clause to feed to `analyze_conflict` and to perform conflict-driven backjumping as if the blocking clause $\neg\eta_k$ belonged to the clause set; importantly, *it is not necessary to add permanently the conflicting clause $\neg\eta_k$ to*

φ *as a blocking clause*, and it is sufficient to keep the conflict clause resulting from conflict analysis only as long as it is active. [3]

In [9] it is proved that this technique terminates and allows for enumerating all models. (Notice that the generation of blocking clauses ψ_k^* representing MinAs is not affected, since in this case we add ψ_k^* to φ as blocking clause.) The only potential drawback of this technique is that some models may be found more than once. However, according to the empirical evaluation in [9], this events appears to be rare and it has very low impact on performances, which are much better than those of the naive version.

We refer the reader to [9] and [14] for more detailed explanations of all-SMT and of our procedure respectively.

Computing one MinA using a much smaller formula. Although polynomial, $\phi_{\mathcal{T}}^{all}/\phi_{\mathcal{T}(po)}^{all}$ may be huge for very-big ontologies $\mathcal{T}$ like SNOMED-CT. For these situations, we propose here a variant of the one-MinA procedure using the much smaller formula $\phi_{\mathcal{T}}^{one}$(which is an improved SAT-based version of the simplified one-MinA algorithm of [5]). [4] $\phi_{\mathcal{T}}^{one}$ is computed like $\phi_{\mathcal{T}}^{all}$, except that step 4. is never performed, so that only one deduction of each assertion is computed. This is sufficient, however, to compute *one* non-minimal axiom set $\mathcal{T}^*$ by one run of `bcp` and `analyze_conflict`, as seen before. Since $\phi_{\mathcal{T}}^{one}$ does not represent all deductions of $C_i \sqsubseteq_{\mathcal{T}} D_i$, we cannot use the algorithm in Figure 2 to minimize it. However, since typically $\mathcal{T}^* \ll \mathcal{T}$, one can cheaply compute $\phi_{\mathcal{T}^*}^{one}$ and run a variant of the algorithm in Figure 2 in which at each loop a novel formula $\phi_{\mathcal{S}\setminus\{a_i\}}^{one}$ is computed and fed to `DPLLUnderAssumptions` together with the updated $\mathcal{L}$. One further variant is to compute instead $\phi_{\mathcal{T}^*(po)}^{all}$ and feed it to the algorithm in Figure 2.

Handling normalization. The normalized TBox $\mathcal{T} \stackrel{\text{def}}{=} \{ax_1, ..., ax_N\}$ can result from normalizing the non-normal one $\hat{\mathcal{T}} \stackrel{\text{def}}{=} \{\hat{ax}_1, ..., \hat{ax}_{\hat{N}}\}$ by means of the process hinted in §2.1. $|\mathcal{T}|$ is $O(|\hat{\mathcal{T}}|)$. Each original axiom $\hat{ax}_i$ is converted into a set of normalized axioms $\{ax_{i1}, ..., ax_{ik_i}\}$, and each axiom ax_{ik_i} can be reused in the conversion of several original axioms $\hat{ax}_{j1}, ..., \hat{ax}_{jk_j}$. In order to handle non-normal TBoxes $\hat{\mathcal{T}}$, we adopt one variant of the technique in [5]: for every $\hat{ax}_i$, we add to $\phi_{\mathcal{T}(po)}^{all}$ [resp. $\phi_{\mathcal{T}}^{all}$] the set of clauses $\{s_{[\hat{ax}_i]} \rightarrow s_{[ax_{i1}]}, ..., s_{[\hat{ax}_i]} \rightarrow s_{[ax_{ik_i}]}\}$, and then we use $\mathcal{P}_{\hat{\mathcal{T}}} \stackrel{\text{def}}{=} \{s_{[\hat{ax}_1]}, ..., s_{[\hat{ax}_{\hat{N}}]}\}$ as the novel set of axiom selector variables for the one-MinA and all-MinAs algorithms described above. Thus `analyze_conflict` finds conflict clauses in terms of variables in $\mathcal{P}_{\hat{\mathcal{T}}}$ rather than in $\mathcal{P}_{\mathcal{T}}$. Since $\mathcal{P}_{\hat{\mathcal{T}}}$ is typically smaller than $\mathcal{P}_{\mathcal{T}}$, this may cause a significant reduction in search for `DPLL1` in the all-MinAs procedure. (Hereafter we will call $\mathcal{T}$ the input TBox, no matter whether normal or not.)

[3] We say that a clause is currently *active* if it occurs in the implication graph, that is, if it is the antecendent clause of some literal in the current assignment. (See [17].)

[4] We prefer considering $\phi_{\mathcal{T}}^{one}$ rather the corresponding formula $\phi_{\mathcal{T}(po)}^{one}$ since it fits better with an optimization described in [14].

4 Discussion

By comparing the pinpointing formula $\Phi^{C_i \sqsubseteq_{\mathcal{T}} D_i}$ of [5] (see also §2.1) with $\phi^{all}_{\mathcal{T}(po)}$, and by analyzing the way they are built and used, we highlight the following differences: (i) $\Phi^{C_i \sqsubseteq_{\mathcal{T}} D_i}$ is built only on axiom selector variables in $\mathcal{P}_{\mathcal{T}} \stackrel{\text{def}}{=} \{s_{[ax_j]} \ |ax_j \in \mathcal{T}\}$, whilst $\phi^{all}_{\mathcal{T}(po)}$ is build on *all* selector variables in $\mathcal{P}_{\mathcal{A}} \stackrel{\text{def}}{=} \{s_{[a_j]} \ |a_j \in \mathcal{A}\}$ (i.e., of both axioms and inferred assertions); (ii) the size of $\Phi^{C_i \sqsubseteq_{\mathcal{T}} D_i}$ and the time to compute it are worst-case *exponential* in $|\mathcal{T}|$ [5], whilst the size of $\phi^{all}_{\mathcal{T}(po)}$ and the time to compute it are worst-case *polynomial* in $|\mathcal{T}|$; (iii) the algorithm for generating $\Phi^{C_i \sqsubseteq_{\mathcal{T}} D_i}$ in [5] requires intermediate logical checks, whilst the algorithm for building $\phi^{all}_{\mathcal{T}(po)}$ does not; (iv) each MinA is a *model* of $\Phi^{C_i \sqsubseteq_{\mathcal{T}} D_i}$, whilst it is (the projection to $\mathcal{P}_{\mathcal{T}}$ of) a *counter-model* of $\phi^{all}_{\mathcal{T}(po)}$. Moreover, our process can reason directly in terms of (the selector variables of) the input axioms, no matter whether normal or not.

In accordance with Theorem 5 in [5], also our approach is not output-polynomial, because in our proposed all-MinAs procedure even the enumeration of a polynomial amount of MinAs may require exploring an exponential amount of possible truth assignments. In our proposed approach, however, the potential exponentiality is completely relegated to the final step of our approach, i.e. to our variant of the all-SMT search, since the construction of the SAT formula is polynomial. Thus we can build $\phi^{all}_{\mathcal{T}(po)}$ once and then, for each $C_i \sqsubseteq_{\mathcal{T}} D_i$ of interest, run the all-SMT procedure until either it terminates or a given timeout is reached: in the latter case, we can collect the MinAs generated so far. (Notice that the fact that `DPLL1` selects *positive* axiom selector variables first tends to anticipate the enumeration of over-constrained assignments wrt. to that of under-constrained ones, so that it is more likely that counter-models, and thus MinAs, are enumerated during the first part of the search.) With the all-MinAs algorithm of [5], it may take an exponential amount of time to build the pinpointing formula $\Phi^{C_i \sqsubseteq_{\mathcal{T}} D_i}$ before starting the enumeration of the MinAs.

As far as the generation of each single MinA of §3.2 is concerned, another interesting feature of our approach relates to the minimization algorithm of Figure 2: we notice that, once $\phi^{all}_{\mathcal{T}(po)}$ is generated, in order to evaluate different subsets $\mathcal{S} \setminus \{a_j\}$ of the axiom sets, it suffices to assume different selector variables, without modifying the formula, and perform one run of `bcp`. Similarly, if we want to compute one or all MinAs for different deduced assertion, e.g. $C_1 \sqsubseteq_{\mathcal{T}} D_1, \ldots, C_j \sqsubseteq_{\mathcal{T}} D_j, \ldots$, we do not need recomputing $\phi^{all}_{\mathcal{T}(po)}$ each time, we just need assuming (i.e. querying) each time a different axiom selector variable, e.g. respectively: $\neg s_{[C_1 \sqsubseteq_{\mathcal{T}} D_1]}, \ldots, \neg s_{[C_j \sqsubseteq_{\mathcal{T}} D_j]}, \ldots$.

5 Empirical Evaluation

In order to test the feasibility of our approach, we have implemented an early-prototype version of the procedures of §3 (hereafter referred as $\mathcal{EL}^+$SAT) which

Ontology	notGalen	GeneOnt.	NCI	fullGalen	Snomed09
# of prim. concepts	2748	20465	27652	23135	310075
# of orig. axioms	4379	20466	46800	36544	310025
# of norm. axioms	8740	29897	46800	81340	857459
# of role names	413	1	50	949	62
# of role axioms	442	1	0	1014	12
Size (var#|clause#)					
$\phi_{\mathcal{T}}$	5.4e3|1.8e4	2.2e4|4.2e4	3.2e4|4.7e4	4.8e4|7.3e5	5.3e5|8.4e6
$\phi_{\mathcal{T}}^{one}$	2.3e4|2.7e4	5.5e4|5.4e4	7.8e4|4.7e4	7.3e5|1.4e6	8.4e6|1.6e7
$\phi_{\mathcal{T}(po)}^{all}$	1.7e5|2.2e5	2.1e5|2.6e5	2.9e5|3.0e5	5.3e6|1.2e7	2.6e7|8.4e7
Encode time					
$\phi_{\mathcal{T}}$	0.65	2.37	2.98	35.28	3753.04
$\phi_{\mathcal{T}}^{one}$	2.06	4.15	6.19	68.94	4069.84
$\phi_{\mathcal{T}(po)}^{all}$	1.17	1.56	2.37	178.41	198476.59
Load time					
$\phi_{\mathcal{T}}$	0.11	0.37	1.01	1.93	21.16
$\phi_{\mathcal{T}}^{one}$	0.18	0.55	1.17	5.95	59.88
Subsumption (on 10^5*)*					
$\phi_{\mathcal{T}}$	0.00002	0.00002	0.00003	0.00003	0.00004
$\phi_{\mathcal{T}}^{one}$	0.00003	0.00002	0.00003	0.00004	0.00008
nMinA $\phi_{\mathcal{T}}^{one}$ *(on 5000)*	0.00012	0.00027	0.00042	0.00369	0.05938
MinA $\phi_{\mathcal{T}}^{one}$ *(on 100)*					
− Load time	0.175	0.387	0.694	6.443	63.324
− Extract time	0.066	0.082	0.214	0.303	3.280
− DPLL Search time	0.004	0.004	0.002	0.010	0.093
MinA $\phi_{\mathcal{T}(po)}^{all}$ *(on 100)*					
− Load time	1.061	1.385	1.370	39.551	150.697
− DPLL Search time	0.023	0.027	0.036	0.331	0.351
allMinA $\phi_{\mathcal{T}(po)}^{all}$*(on 30)*					
− 50% *#MinA/time*	1/1.50	1/1.76	4/1.79	3/53.40	15/274.70
− 90% *#MinA/time*	2/1.59	4/2.11	6/1.86	9/63.61	32/493.61
− 100% *#MinA/time*	2/1.64	8/2.79	9/2.89	15/150.95	40/588.33

Table 3. "XeN" is "$X \cdot 10^N$". CPU times are in seconds.

does not yet include all optimizations described here and in [14], and we performed a preliminary empirical evaluation of $\mathcal{EL}^+$SAT on the ontologies of §1. [5] We have implemented $\mathcal{EL}^+$SAT in C++, modifying the code of the SAT solver MiniSat2.0 070721 [8]. All tests have been run on a biprocessor dual-core machine `Intel Xeon 3.00GHz` with 4GB RAM on `Linux RedHat 2.6.9-11`, except for $\phi_{\mathcal{T}(po)}^{one}$ of Snomed-CT'09 which is processed on a `Intel Xeon 2.66 GHz` machine with 16 GB RAM on `Debian Linux 2.6.18-6-amd64`. [6]

[5] The first four ontologies are available at `http://lat.inf.tu-dresden.de/~meng/toyont.html`; Snomed-CT'09 is courtesy of IHTSDO `http://www.ihtsdo.org/`.

[6] $\mathcal{EL}^+$SAT is available from `http://disi.unitn.it/~rseba/elsat/`.

The results of the evaluation are presented in Table 3. The first block reports the data of each ontology. The second and third blocks report respectively the size of the encoded formula, in terms of variable and clause number, and the CPU time taken to compute them. [7] The fourth block reports the time taken to load the formulas and to initialize `DPLL`. The fifth block reports the average time (on 100000 sample queries) required by computing subsumptions. [8] (Notice that $\phi_{\mathcal{T}}$ and $\phi_{\mathcal{T}}^{one}$ must be loaded and `DPLL` must be initialized only once for all queries.) The sixth block reports the same data for the computation of one nMinA, on 5000 sample queries. [9] (Loading times are the same as above.) The seventh block reports the average times on 100 samples required to compute one MinA with $\phi_{\mathcal{T}}^{one}$. [10] The eighth block reports the average times on 100 samples required to compute one MinA with $\phi_{\mathcal{T}(po)}^{all}$. The ninth block reports the results (50th, 90th and 100th percentiles) of running the all-MinAs procedure on 30 samples, each with a timeout of 1000s (loading included), and counting the number of MinAs generated and the time taken until the last MinA is generated. [11]

Although still very preliminary, these empirical results allow us to notice a few facts: (i) once the formulas are loaded, concept subsumption and computation of nMinAs are instantaneous, even with very-big formulas $\phi_{\mathcal{T}}$ and $\phi_{\mathcal{T}}^{one}$; (ii) in the computation of single MinAs, with both $\phi_{\mathcal{T}}^{one}$ and $\phi_{\mathcal{T}(po)}^{all}$, DPLL search times are very low or even negligible: most time is taken by loading the main formula (which can be performed only once for all) and by extracting the information from intermediate results. Notice that $\mathcal{EL}^+$SAT succeeded in computing some MinAs even with the huge ontology SNOMED-CT'09; (iii) although no sample concluded the full enumeration within the timeout of 1000s, the all-MinAs procedure allowed for enumerating a set of MinAs. Remarkably, all MinAs are all found in the very first part of the search, as expected.

6 Ongoing and Future Work.

The current implementation of $\mathcal{EL}^+$SAT is still very naive to many extents. We plan to implement an optimized version of $\mathcal{EL}^+$SAT, including all techniques and

[7] The classification alone (excluding the time taken in encoding the problem and in computing the additional *rule clauses* for pinpointing) required respectively: 0.60, 2.24, 2.84, 34.06 and 3738.82 seconds for $\phi_{\mathcal{T}}$, 0.99, 2.63, 4.13, 41.19 and 3893.20 seconds for $\phi_{\mathcal{T}}^{one}$. In the case of $\phi_{\mathcal{T}(po)}^{all}$ the times are not distinguishable.

[8] The queries have been generated randomly, extracting about 2000 primitive concept names from each ontology and then randomly selecting 100000 queries from all the possible combinations of these concept names.

[9] We chose the first 5000 "unsatisfiable" queries we encounter when analyzing all the possible pairwise combinations of primitive concept names of each ontology.

[10] The queries are selected randomly from the 5000 samples introduced above.

[11] First, we sort the assertions computed for each ontology wrt. the number of occurrences as implicate in *rule clauses* then, following this order, we pick with a probability of 0.25 (to avoid queries which are too similar) the 30 sample assertions to be queried.

optimizations presented here and in [14]. (We plan to investigate and implement also a SAT-based versions of the techniques based on reachability modules of [6].) Then we plan to perform a very-extensive empirical analysis of the optimized tools; we also plan to implement a user-friendly GUI for $\mathcal{EL}^+$SAT so that to make it usable by domain experts. Research-wise, we plan to investigate alternative sets of completion rules, which may be more suitable for producing smaller $\phi^{all}_{\mathcal{T}(po)}$ formulas, and to extend our techniques to richer logics.

References

1. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ Envelope. In *Proc. of IJCAI'05*, pages 364–369. Morgan-Kaufmann Publishers, 2005.
2. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ Envelope Further. In *Proc. of OWLED'08, DC Workshop*, 2008.
3. F. Baader, C. Lutz, and B. Suntisrivaraporn. Efficient Reasoning in $\mathcal{EL}^+$. In *Proc. of DL'06*, vol. 189 of *CEUR-WS*, 2006.
4. F. Baader and R. Penaloza. Axiom pinpointing in general tableaux. In *Proc. of TABLEAUX'07*, LNAI, pages 11–27. Springer, 2007.
5. F. Baader, R. Peñaloza, and B. Suntisrivaraporn. Pinpointing in the Description Logic $\mathcal{EL}^+$. In *Proc. of KI'07*, vol. 4667 of *LNCS*, pages 52–67. Springer, 2007.
6. F. Baader and B. Suntisrivaraporn. Debugging SNOMED CT Using Axiom Pinpointing in the Description Logic $\mathcal{EL}^+$. In *Proc. of KR-MED'08: Representing and Sharing Knowledge Using SNOMED*, vol. 410 of *CEUR-WS*, 2008.
7. T. G. O. Consortium. Gene ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.
8. N. Eén and N. Sörensson. An Extensible SAT-solver. In *Proc. of SAT'03*, vol. 2919 of *LNCS*, pages 502–518. Springer, 2004.
9. S. K. Lahiri, R. Nieuwenhuis, and A. Oliveras. SMT techniques for fast predicate abstraction. In *Proc. of CAV'06*, vol. 4144 of *LNCS*, pages 424–437. Springer, 2006.
10. I. Lynce and J. P. M. Silva. On Computing Minimum Unsatisfiable Cores. In *Proc. of SAT'04*, 2004. `http://www.satisfiability.org/SAT04/programme/110.pdf`.
11. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proc. of DAC'01*, pages 530–535. ACM, 2001.
12. A. Rector and I. Horrocks. Experience Building a Large, Re-usable Medical Ontology using a Description Logic with Transitivity and Concept Inclusions. In *Proc. of the Workshop on Ontological Engineering, AAAI'97*. AAAI Press, 1997.
13. R. Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation, JSAT*, 3:141–224, 2007.
14. R. Sebastiani and M. Vescovi. Axiom Pinpointing in Lightweight Description Logics via Horn-SAT Encoding and Conflict Analysis. Technical Report DISI-09-014, University of Trento, 2009. `http://disi.unitn.it/~rseba/elsat/`.
15. N. Sioutos, S. de Coronado, M. W. Haber, F. W. Hartel, W. Shaiu, and L. W. Wright. NCI Thesaurus: A semantic model integrating cancer-related clinical and molecular information. *Journal of Biomedical Informatics*, 40(1):30–43, 2007.
16. K. A. Spackman. Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED RT. *Journal of American Medical Informatics Association (Fall Symposium Special Issue)*, 2000.
17. L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient Conflict Driven Learning in Boolean Satisfiability Solver. In *Proc. of ICCAD*, pages 279–285, 2001.