UNIVERSITY
OF TRENTO

DIPARTIMENTO DI INGEGNERIA E SCIENZA DELL'INFORMAZIONE

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.disi.unitn.it

AXIOM PINPOINTING IN LIGHTWEIGHT DESCRIPTION LOGICS VIA HORN-SAT ENCODING AND CONFLICT ANALYSIS

Roberto Sebastiani and Michele Vescovi

First version: March 2009
Last revision: July 2009

Technical Report # DISI-09-014

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

Roberto Sebastiani and Michele Vescovi

DISI, Università di Trento, Via Sommarive 14, I-38123, Povo, 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 comfirms the potential of the approach.

1 Motivations and goals

In contrast to the trend of the last two decades [3], in which the research in description logic has focused on investigating increasingly expressive logics, 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 [1,4,6,12,16,2]. In particular, the logic $\mathcal{EL}^+$ [4,6,7] 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 [24,23,26], NCI [22], GENEONTOLOGY [8] and the majority of GALEN [17]. 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?" [7]), but also more sophisticated logic problems such as *axiom pinpointing* are tractable. (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`?" [7]) Importantly, the problem of axiom pinpointing in $\mathcal{EL}^+$ is of great interest for debugging complex bio-medical ontologies (see, e.g., [7]). To this extent, the problems of concept subsumption and axiom pinpointing in $\mathcal{EL}^+$ have been thoroughly investigated, and efficient algorithms for these two

functionalities have been implemented and tested with success on large ontologies, including SNOMED-CT (see e.g. [4,6,7]).

The description logic community has spent a considerable effort in the attempt of extending $\mathcal{EL}$ as much as possible, defining a maximal subset of logical constructors expressive enough to cover the needs of the practical applications above mentioned, but whose inference problems remain tractable. Beside the logic $\mathcal{EL}^+$ [4], on which we focus in this work, many other extension of $\mathcal{EL}$ have been studied [1,2].

In this paper we build on previous work from the literature of $\mathcal{EL}^+$ reasoning [4,6,7] and of SAT and SMT [15,27,11,13,18], 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 [4,6], and to manipulate them by exploiting the functionalities of modern conflict-driven SAT/SMT solvers — like *Boolean Contraint Propagation (BCP)* [15], *conflict analysis under assumptions* [15,11], and *all-SMT* [13]. In particular, we show that from an ontology $\mathcal{T}$ it is possible to generate in polynomial time Horn propositional formulas $\phi_{\mathcal{T}}$, $\phi^{one}_{\mathcal{T}}$ and $\phi^{all}_{\mathcal{T}(po)}$ 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 $\phi^{one}_{\mathcal{T}}$;
(ii) *one* non-minimal set of axioms (nMinA) responsible for the derivation of $C_i \sqsubseteq_{\mathcal{T}} D_i$ is computed by one run of BCP and conflict analysis on $\phi^{one}_{\mathcal{T}}$ or $\phi^{all}_{\mathcal{T}(po)}$;
(iii) *one minimal* such set (MinA) is computed by iterating process (ii) on $\phi^{all}_{\mathcal{T}(po)}$ for an amount of times 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 $\phi^{one}_{\mathcal{T}}$,$\phi^{one}_{\mathcal{S}_1}$,...,$\phi^{one}_{\mathcal{S}_k}$;
(v) *all* MinAs can be enumerated by means of all-SMT techniques on $\phi^{all}_{\mathcal{T}(po)}$, using step (iii) as a subroutine.

It is worth noticing that (i) and (ii) are instantaneous even with huge $\phi_{\mathcal{T}}$, $\phi^{one}_{\mathcal{T}}$ and $\phi^{all}_{\mathcal{T}(po)}$, and that (v) requires building a *polynomial-size* formula $\phi^{all}_{\mathcal{T}(po)}$, in contrast to the exponential-size formula required by the all-MinAs process of [6].

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.

Content. In §2 we provide the necessary background on $\mathcal{EL}^+$ reasoning and on conflict-driven SAT solving; in §3 we present our SAT-based procedures for concept subsumption, one-MinA extraction and all-MinAs enumeration; in §4 we discuss our techniques and compare them with those in [6]; in §5 we present our preliminary empirical evaluation, in §6 we draw some conclusions and outline directions for future research.

	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 $\mathcal{EL}^+$.

2 Background

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

We overview the main notions concerning concept subsumption, classification, 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) [4]; $\mathcal{EL}^+$ 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 part 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 part of Table 1. Given a TBox $\mathcal{T}$, we denote with $\mathsf{PC}_{\mathcal{T}}$ the 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.

Given the concepts X and Y, Y *subsumes* X w.r.t. the TBox $\mathcal{T}$, written $X \sqsubseteq_{\mathcal{T}} Y$ (or simply $X \sqsubseteq Y$ when it is clear to which TBox we refer to),

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

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 $\mathcal{A}$.

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 $\mathcal{T}$. Concept subsumption and classification in $\mathcal{EL}^+$ can be performed in polynomial time [1,6]. In particular, in [1,6], the problem of classifying an $\mathcal{EL}^+$ TBox is solved as a subcase of the polynomial-time algorithm for concept subsumption in which all the possible concept subsumptions in the TBox are deduced.

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 [1,4,5,6]:

$$(C_1 \sqcap ... \sqcap C_k) \sqsubseteq D \qquad k \geq 1 \tag{1}$$

$$C \sqsubseteq \exists r.D \tag{2}$$

$$\exists r.C \sqsubseteq D \tag{3}$$

$$r_1 \circ \cdots \circ r_n \sqsubseteq s \qquad n \geq 1 \tag{4}$$

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 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 $\mathcal{T}'$. (I.e., the set $\mathsf{NC}'_{\mathcal{T}}$ consists in all the concepts of the form C or $\exists r.C$, with $C \in \mathsf{PC}_{\mathcal{T}'}$ and $r \in \mathsf{PR}_{\mathcal{T}'}$.)

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 $\mathcal{EL}^+$ [6] 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 $\mathcal{A} = \{a_i \in \mathcal{T} \mid a_i \text{ is a GCI}\} \cup \{C \sqsubseteq C \mid C \in \mathsf{PC}_\mathcal{T}\} \cup \{C \sqsubseteq \top \mid C \in \mathsf{PC}_\mathcal{T}\}$ 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. Intuitively, since the number of concept and role names is linear in the size of the input TBox, the algorithm cannot add to $\mathcal{A}$ more than the cardinality of $\mathsf{PC}_\mathcal{T} \times \mathsf{PC}_\mathcal{T} \times \mathsf{PR}_\mathcal{T}$ assertions. Thus, since no rule removes assertions from $\mathcal{A}$, the algorithm stops after at most a polynomial number of rule applications. Moreover, it is easy to device that every rule application 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* $C_i \sqsubseteq D_i$ wrt. $\mathcal{T}$, 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. [6] 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 $\mathcal{P}_\mathcal{T} \stackrel{\text{def}}{=} \{s_{[ax_j]} \mid ax_j \in \mathcal{T}\}$ s.t., for every $\mathcal{O} \subseteq \mathcal{T}$, $\mathcal{O}$ is a MinA wrt. $C_i \sqsubseteq_\mathcal{T} D_i$ iff $\{s_{[ax_i]} \mid ax_i \in \mathcal{O}\}$ is a minimal valuation of $\Phi^{C_i \sqsubseteq_\mathcal{T} D_i}$. [1] Thus, the all-MinAs algorithm in [6] consists in (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 [6], however, this algorithm has serious limitations in terms of complexity: first, the algorithm for generating $\Phi^{C_i \sqsubseteq_\mathcal{T} D_i}$ 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, [6] 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, [6] 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 [7] by means of a binary-search minimization algorithm, and by a novel algorithm exploiting the

[1] A monotone propositional formula is a formula whose only connectives are $\wedge$ and $\vee$.

notion of *reachability-modules*, which allowed to find efficiently MinAs for the much bigger SNOMED-CT ontology. We refer the readers to [6,7] for a detailed description.

Further, in a very-recent work [25] the all-MinAs problem is solved with a different approach based on the techniques of the Hitting Set Tree (HST), where the universal set is the whole ontology and the set of the all MinAs is collection of the minimal subsets to be found. In particular the hitting set tree is expanded along the algorithm computing, at the end, all the MinAs for the given subsumption. In this approach the optimized algorithm and the linear minimization algorithm above exposed are used as subroutines respectively to initialize the algorithm and to minimize the sets found. However, also this techniques has the major drawback of performance in large-scale ontologies, thus it has been implemented and succeeded in finding all-MinAs in combination with the reachability-modules extraction technique wich drastically reduces the search space of the HST algorithm. We refer the readers to [25] for a richer explanation and detailed explanation of the approach.

2.2 Basics on Conflict-Driven SAT Solving

For the best comprehension of the content of §3, we recall some notions on SAT and on conflict-driven SAT solving. For a much deeper description, we refer the reader to the literature (e.g., [28,11,14]).

Basics on SAT and notation. We assume the standard syntactic and semantic notions of propositional logic. Given a non-empty set of primitive propositions $\mathcal{P} = \{p_1, p_2, \ldots\}$, the language of propositional logic is the least set of formulas containing $\mathcal{P}$ and the primitive constants $\top$ and $\bot$ ("true" and "false") and closed under the set of standard propositional connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$. We call a *propositional atom* every primitive proposition in $\mathcal{P}$, and a *propositional literal* every propositional atom (*positive literal*) or its negation (*negative literal*). We implicitly remove double negations: e.g., if l is the negative literal $\neg p_i$, by $\neg l$ we mean p_i rather than $\neg\neg p_i$. 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].

A propositional formula is in *conjunctive normal form, CNF,* if it is written as a conjunction of disjunctions of literals: $\bigwedge_i \bigvee_j l_{ij}$. Each disjunction of literals $\bigvee_j l_{ij}$ is called a *clause*. 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.

A *unit clause* is a clause with only one literal. A *Horn clause* is a clause containing at most one positive literal, and a *Horn formula* is a conjunction of Horn clauses. Notice that Horn clauses are either unary positive clauses, or they contain at least one negative literal. A *definite Horn clause* is a non-unary Horn clause containing exactly one positive literal (and hence at least one negative

```
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.

one), and a a *definite Horn formula* is a conjunction of definite Horn clauses. (Intuitively, definite Horn formulas represents sets of implications between Boolean variables $(\bigwedge_{i=1}^{n} p_i) \rightarrow p_j$ s.t. $n > 0$.) Notice that a definite Horn formula ϕ is always satisfiable, since it is satisfied by both the assignments $\mu_\top$ and $\mu_\bot$ which assign all variables to true and false respectively. Notice also that, for every subset $\{p_i\}_i$ of Boolean variables in ϕ, $\phi \wedge \bigwedge_i p_i$ and $\phi \wedge \bigwedge_i \neg p_i$ are satisfied by $\mu_\top$ and $\mu_\bot$ respectively. Thus, in order to falsify a definite Horn formula ϕ, it is necessary to add to it at least one positive and one negative literal.

The problem of detecting the satisfiability of a propositional CNF formula, also referred as the *SAT problem*, is NP-complete. A *SAT solver* is a tool able to solve the SAT problem. The problem of detecting the satisfiability of a propositional Horn formula, also referred as the *Horn-SAT problem*, is polynomial.

Conflict-driven SAT solving. Most state-of-the-art SAT procedures are evolutions of the Davis-Putnam-Longeman-Loveland (DPLL) procedure [10,9] and they are based on the conflict-driven paradigm [21,27]. A high-level schema of a modern conflict-driven DPLL engine, adapted from the one presented in [28], is shown in Figure 1. 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, `bcp`(φ, μ) 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, `analyze_conflict`(φ, μ) 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, backtrack(blevel, φ, μ) 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 [15]. 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 [21,15,27]. 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 [28]), until some stop criterion is met. E.g., with the *1st-UIP Scheme* the last-assigned literal in ψ' is the one always picked, and the process stops as soon as ψ' contains only one literal l_u assigned at the last decision level; 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 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 $\bigvee_{s_k \in \mathcal{L}'} \neg s_k$, s.t. $\{\psi_k \mid s_k \in \mathcal{L}'\}$ is the actual subset of clauses which caused the inconsistency of φ. This technique is used to compute unsatisfiable cores of CNF propositional formulas [14].

3 Axiom Pinpointing via Horn SAT and Conflict Analysis

In this section we present our novel contributions. Since our work is inspired by that in [6], we follow the same flow of that paper. 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 first 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 one fresh Boolean variable $p_{[X]}$ which is uniquely-associated to X. We initially set $\phi_{\mathcal{T}}$ to be the empty set of clauses. We run the classification algorithm of §2.1: for every non-trivial [2] axiom or assertion a_i of the form (1)-(3) which is added to $\mathcal{A}$, we add to $\phi_{\mathcal{T}}$ one clause $\mathcal{EL}^+2sat(a_i)$ of the form

$$p_{[C_1]} \wedge ... \wedge p_{[C_k]} \rightarrow p_{[D]} \qquad k \geq 1 \tag{5}$$

$$p_{[C]} \rightarrow p_{[\exists r.D]} \tag{6}$$

$$p_{[\exists r.C]} \rightarrow p_{[D]} \tag{7}$$

[2] 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 (5)-(7) are 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 $\mathsf{PC}_{\mathcal{T}}$. (See Theorem 1 in Appendix A for more details and a formal proof.) 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, namely $\phi^{\star}_{\mathcal{T}}$, is possible since we notice that the first two completion rules in Table 2 (which we call *propositional completion rules* hereafter) correspond to purely-propositional inference steps. Thus, we can omit adding to $\phi^{\star}_{\mathcal{T}}$ the clauses encoding assertions deriving from propositional completion rules, because these clauses are entailed by the clauses encoding the promises of the rules. Therefore, every clause ψ in $\phi_{\mathcal{T}} \setminus \phi^{\star}_{\mathcal{T}}$ is s.t. $\phi^{\star}_{\mathcal{T}} \models \psi$, so that $\phi^{\star}_{\mathcal{T}}$ is equivalent to $\phi_{\mathcal{T}}$. Thus, as before, $C \sqsubseteq_{\mathcal{T}} D$ if and only if $\phi^{\star}_{\mathcal{T}} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable. In order to improve the reduction in size, in the classification algorithm of §2.1 one can adopt a heuristic strategy of applying propositional completion rules first: if one assertion can be derived both from a propositional and a non-propositional rule, the first is applied, and no clause is added to $\phi_{\mathcal{T}}$.

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 single run of `bcp`, whose cost depends linearly only on the clauses where the unit-propagated literals occur. In practice, if the basic encoding was used, or if $C_i \sqsubseteq_{\mathcal{T}} D_i$ has been inferred by means of a non-propositional rule, then $\phi_{\mathcal{T}}$ contains the clause $p_{[C_i]} \rightarrow p_{[D_i]}$, so that `bcp` stops as soon as $\neg p_{[D_i]}$ and $p_{[C_i]}$ are unit-propagated; if instead the more compact encoding was used and $C_i \sqsubseteq_{\mathcal{T}} D_i$ has been inferred by means of a (chain of) propositional rule(s), then `bcp` stops as soon as the literals involved in this chain have been unit-propagated. In both cases, as discussed in §2.2, each query is instantaneous even for a huge $\phi_{\mathcal{T}}$.

3.2 Computing single and all MinAs via Conflict Analysis

We consider the general problem of generating MinAs. We build another Horn propositional formula $\phi^{all}_{\mathcal{T}}$ representing the complete classification DAG of the input normalized ontology $\mathcal{T}$. [3] The size of $\phi^{all}_{\mathcal{T}}$ is polynomial wrt. that of $\mathcal{T}$.

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

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

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

$$s_{[a_i]} \rightarrow \mathcal{EL}^+2sat(a_i) \tag{8}$$

   s.t. $s_{[a_i]}$ is the axiom selector variable for a_i and $\mathcal{EL}^+2sat(a_i)$ is the clause encoding a_i, as in (5)-(7);
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 (8) and a clause of the form

$$(\bigwedge_{a_j \in\ ant(r)} s_{[a_j]}) \rightarrow s_{[a_i]} \tag{9}$$

   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);
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 (9).

Notice that (8) and (9) are definite Horn clauses since all (5)-(7) are definite Horn clauses. (We call (8) and (9) *assertion clauses* and *rule clauses* respectively.) Notice also that step 4. is novel wrt. the classification algorithm of §2.1. Notice also that, if the rule clauses (9) were not added to $\phi_{\mathcal{T}}^{all}$, then $\phi_{\mathcal{T}}^{all} \wedge \bigwedge_{a_i \in \mathcal{T}} s_{[a_i]}$ would be simply a conservative extension of $\phi_{\mathcal{T}}$.

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. This can be achieved, e.g., with the following strategy: initially all GCI axioms are added to a queue Q and all axioms are added to $\mathcal{A}$; at each step an assertion a_k is dequeued, and steps 3. or 4. are applied to *all and only the rules applications whose antecedents are* a_k *and one or two of the previously-dequeued axioms/assertions* $a_1, ..., a_{k-1}$; the novel assertions a_{k+j} deduced by the rule application in step 3. are added to the queue Q and to $\mathcal{A}$. This process ends when the queue is empty. A pseudo-code representation of this algorithm is exposed in Figure 2.

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 $\mathcal{T}$. In order to make the explanation simpler, we assume wlog. that in all axioms in $\mathcal{T}$ all $\sqcap$'s and $\circ$'s are binary, i.e., that $1 \leq k \leq 2$ in (2) and $1 \leq n \leq 2$ in (4). [4] Thus,

[4] This is not restrictive, since, e.g., each GCI axiom of the form $C_1 \sqcap ... \sqcap C_k \sqsubseteq D$ in $\mathcal{T}$ can be rewritten into the set $\{C_1 \sqcap C_2 \sqsubseteq C_{1:2}, C_{1:2} \sqcap C_3 \sqsubseteq C_{1:3}, ..., C_{1:k-1} \sqcap C_k \sqsubseteq D\}$, and each RI axiom of the form $r_1 \circ \cdots \circ r_n \sqsubseteq s$ can be rewritten into the set $\{r_1 \circ r_2 \sqsubseteq r_{1:2}, r_{1:2} \circ r_3 \sqsubseteq r_{1:3}, ..., r_{1:n-1} \circ r_n \sqsubseteq s\}$, each $C_{1:i}$ and $r_{1:j}$ being a fresh concept name and a fresh role name respectively.

```
ClauseSet build-φ_T^all (NormalizedOntology T)
// Initialization of A and Q
 1.    Q = {}; A = {};
 2.    for each primitive concept C in T
 3.       add C ⊑ C and C ⊑ ⊤ to A; introduce s_[C⊑C] = s_[C⊑⊤] = ⊤;
 4.       enqueue {C ⊑ C, C ⊑ ⊤} into Q;
 5.    for each GCI or RI axiom ax in T
 6.       add ax to A; introduce s_[ax];
 7.       if ax is a non-trivial GCI axiom then
 8.          add the clause (s_[ax] → EL+2sat(ax)) to φ_T^all;
 9.          enqueue ax into Q;
// Updating A, B and Q (B is the set of already-handled assertions)
10.    B = ∅;
11.    while Q is not empty
12.       dequeue a from Q;
13.       for each rule instance r such that ant(r) \ B = {a}
14.          if gen(r) ∉ A then
15.             add gen(r) to A; introduce s_[gen(r)];
16.             add the clause (s_[gen(r)] → EL+2sat(gen(r))) to φ_T^all;
17.             enqueue gen(r) into Q;
18.          add the clause ((⋀_{a_j ∈ ant(r)} s_[a_j]) → s_[gen(r)]) to φ_T^all;
19.       B = B ∪ {a}
20.    return φ_T^all;
```

Fig. 2. Polynomial-time algorithm building the formula $\phi_{\mathcal{T}}^{all}$. Q is a queue of assertions, $\mathcal{A}$ and $\mathcal{B}$ are sets of assertions.

every rule application has at most three antecedents: one axiom and one or two assertions. Let $\mathcal{A}^*$ be the final set of assertions. Then the number of different rule applications on axioms and assertions in $\mathcal{A}^*$ is upper-bounded by $|\mathcal{A}^*|^2 \cdot |\mathcal{T}|$. Thus, it suffices to avoid repeating the same rule application more than once (as described above and shown in Figure 2) to keep both the algorithm and the final size of $\phi_{\mathcal{T}}^{all}$ polynomial.

It follows 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. (See Theorems 2 and 3 in Appendix A for details and formal proofs.)

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

```
AxiomSet lin-extract-MinA_DPLL(Concept C_i, D_i, AxiomSet T*, formula φ^all_T)
1.      S = T*;
2.      for each axiom a_j in T*
3.          L = {¬p_[D_i], p_[C_i]} ∪ {s_[a_i] | a_i ∈ S \ {a_j}};
4.          if (DPLLUnderAssumptions(φ^all_T, L) == unsat)
5.              S = S \ {a_j};
6.      return S;
```

Fig. 3. SAT-based variant of the linear MinA-extracting algorithm in [6].

$\psi^{C_i,D_i}_{\mathcal{T}^*} \stackrel{\text{def}}{=} p_{[D_i]} \vee \neg p_{[C_i]} \vee \bigvee_{a_i \in \mathcal{T}^*} \neg s_{[a_i]}$ 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. (Intuitively, `analyze_conflict` implicitly spans upward the classification sub-DAG rooted in $C_i \sqsubseteq_{\mathcal{T}} D_i$ and having $\mathcal{T}^*$ as leaf nodes, which contains all and only the nodes of the assertions which have been used to generate $C_i \sqsubseteq_{\mathcal{T}} D_i$.)

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 [6] in Figure 3.(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 $C_i \sqsubseteq_{S \setminus \{a_j\}} D_i$. (The correctness of this algorithm is a straightforward consequence of Theorem 3 in Appendix A.) 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. [7]).

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 3 instead of $\phi^{all}_{\mathcal{T}}$.

One very important remark is in order. During pinpointing the only clause of type (8) 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 (8) 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 $\mathcal{L}_i \stackrel{\text{def}}{=} \{\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]}\} \cup \{s_{[a_i]} \mid a_i \in \mathcal{T}\}$, 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 (see Theorem 4 in Appendix A). Thus, wlog. 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 of $C_i \sqsubseteq_{\mathcal{T}} D_i$ from $\phi^{all}_{\mathcal{T}(po)}$ and $\{\neg s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]}\}$. (As before, the same results can be obtained if $\phi^{all}_{\mathcal{T}}$ and $\{\neg p_{[D_i]}, p_{[C_i]}\}$ are used instead.) 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 $\mathcal{P}_{\mathcal{T}} \stackrel{\text{def}}{=} \{s_{[ax_j]} \mid ax_j \in \mathcal{T}\}$ which cause the inconsistency of the formula $\phi^{all}_{\mathcal{T}(po)}$. This can be implemented by means of a variant of the all-SMT technique in [13]. A naive version of this technique is described as follows.

We consider a propositional CNF formula φ on the set of axiom selector 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` can use $s_{[C_i \sqsubseteq_{\mathcal{T}} D_i]} \vee \neg \mu_k$ 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 whole 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.

Here we show that this naive procedure returns all MinAs of $C_i \sqsubseteq_{\mathcal{T}} D_i$. The procedure enumerates truth assignments on the variables in $\mathcal{P}_{\mathcal{T}}$ and checks whether they cause the inconsistency of the formula $\phi^{all}_{\mathcal{T}(po)}$ by `bcp` only. The

search ends when all possible such assignments violate some conflict clause added to φ from the $\mathcal{T}$-solver (either an actual conflict clause or a "fake" one), that is, when we have $\neg s_{[C_i \sqsubseteq_\mathcal{T} D_i]} \wedge \varphi \wedge \bigwedge_h (s_{[C_i \sqsubseteq_\mathcal{T} D_i]} \vee \bigvee_{ax_j \in \mathcal{T}_h^*} \neg s_{[ax_j]}) \wedge \bigwedge_k (s_{[C_i \sqsubseteq_\mathcal{T} D_i]} \vee \neg \mu_k) \models \bot$, [5] that is, $\bigwedge_h (\bigvee_{ax_j \in \mathcal{T}_h^*} \neg s_{[ax_j]}) \wedge \bigwedge_k \neg \mu_k \models \bot$ since φ is set to $\top$. This means that every total assignment η on the variables in $\mathcal{P}_\mathcal{T}$ violates some clause in the latter formula, in particular: if η is s.t. the formula $\neg s_{[C_i \sqsubseteq D_i]} \wedge \eta \wedge \phi^{all}_{\mathcal{T}(po)}$ is satisfiable, then η violates one of the clauses of the form $\neg \mu_k$, otherwise η violates one of the clauses of the form $\bigvee_{ax_j \in \mathcal{T}_h^*} \neg s_{[ax_j]}$. Let $\mathcal{S}$ be a set of axioms, and let $\eta_\mathcal{S} \stackrel{\text{def}}{=} \{s_{[ax_i]} \mid ax_i \in \mathcal{S}\} \cup \{\neg s_{[ax_i]} \mid ax_i \in \mathcal{T} \setminus \mathcal{S}\}$. If $C_i \sqsubseteq_\mathcal{S} D_i$, then $\neg s_{[C_i \sqsubseteq_\mathcal{T} D_i]} \wedge \eta_\mathcal{S} \wedge \phi^{all}_{\mathcal{T}(po)}$ is unsatisfiable. Thus, $\eta_\mathcal{S}$ violates some clause $\bigvee_{ax_j \in \mathcal{T}_h^*} \neg s_{[ax_j]}$, as stated above, that is, $\mathcal{S} \supseteq T_h^*$ for some MinA T_h^*. Thus, this procedure returns all MinAs of $C_i \sqsubseteq_\mathcal{T} D_i$.

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., [18]). The $\mathcal{T}$-solver can be 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 3, 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 [13], 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. [6]

In [13] 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

[5] In general, an SMT solver which is run on a $\mathcal{T}$-unsatisfiable formula φ stops when $\varphi \wedge \bigwedge_k \neg \eta_k \models \bot$, s.t. the η_ks are the $\mathcal{T}$-conflict sets returned by the $\mathcal{T}$-solver and "$\models$" is purely-propositional entailment.

[6] 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 [27].)

than once. However, according to the the empirical evaluation in [13], 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 [13] for a more detailed explanation of all-SMT.

One remark is in order. The reason why we use two different instances of `DPLL` is that we must distinguish unit-propagations of negated axiom selector variables $\neg s_{[ax_i]}$ on learned clauses from those performed on the clauses in $\phi^{all}_{\mathcal{T}(po)}$: on the one hand, we want to allow the former ones because they prevent exploring the same assignments more than once; on the other hand, we want to avoid the latter ones (or to perform them in a controlled way, as explained in the theory propagation variant) because they may prevent generating some counter-model of interest.

Computing one MinA using a much smaller formula. Although polynomial, $\phi^{all}_{\mathcal{T}}/\phi^{all}_{\mathcal{T}(po)}$ 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 $\phi^{one}_{\mathcal{T}}$ [resp. $\phi^{one}_{\mathcal{T}(po)}$] (which is an improved SAT-based version of the simplified one-MinA algorithm of [6]). [7] $\phi^{one}_{\mathcal{T}}$ [resp. $\phi^{one}_{\mathcal{T}(po)}$] is computed like $\phi^{all}_{\mathcal{T}}$ [resp. $\phi^{all}_{\mathcal{T}(po)}$], 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 $\phi^{one}_{\mathcal{T}}$ does not represent all deductions of $C_i \sqsubseteq_{\mathcal{T}} D_i$, we cannot use the algorithm in Figure 3 to minimize it. However, since typically $\mathcal{T}^* \ll \mathcal{T}$, one can cheaply compute $\phi^{one}_{\mathcal{T}^*}$ and run a variant of the algorithm in Figure 3 in which at each loop a novel formula $\phi^{one}_{\mathcal{S}\setminus\{a_i\}}$ is computed and fed to `DPLLUnderAssumptions` together with the updated $\mathcal{L}$. One further variant is to compute instead $\phi^{all}_{\mathcal{T}^*(po)}$ and feed it to the algorithm in Figure 3.

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 [6]: for every $\hat{ax}_i$, we add to $\phi^{all}_{\mathcal{T}(po)}$ [resp. $\phi^{all}_{\mathcal{T}}$] 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 $\mathcal{P}_{\mathcal{T}}$. (In practice, we treat normalization as the application of a novel kind of completion rules.) Since $\mathcal{P}_{\hat{\mathcal{T}}}$ is typically smaller than $\mathcal{P}_{\mathcal{T}}$, this may cause

[7] We prefer considering $\phi^{one}_{\mathcal{T}}$ rather than the corresponding formula $\phi^{one}_{\mathcal{T}(po)}$ since it fits better with the removal of transitive clauses described in §3.1.

a significant reduction in search for DPLL1 in the all-MinAs procedure. (Notice that when one ax_j is shared by $a\hat{x}_{j1}, ..., a\hat{x}_{jk_j}$, the clause set $\{s_{[a\hat{x}_{j1}]} \rightarrow s_{[a_j]}, ..., s_{[a\hat{x}_{jk_j}]} \rightarrow s_{[a_j]}\}$ is equivalent to $(s_{[a\hat{x}_{j1}]} \vee ... \vee s_{[a\hat{x}_{jk_j}]}) \rightarrow s_{[a_j]}$.) (Hereafter we will call $\mathcal{T}$ the input TBox, no matter whether normal or not.)

4 Discussion

We first compare our all-MinAs technique for $\mathcal{EL}^+$ of §3.2 with that presented in [6]. By comparing the pinpointing formula $\Phi^{C_i \sqsubseteq_\mathcal{T} D_i}$ of [6] (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 $\mathcal{P}_\mathcal{T} \stackrel{\text{def}}{=} \{s_{[ax_j]} \mid ax_j \in \mathcal{T}\}$, whilst $\phi^{all}_{\mathcal{T}(po)}$ is build on *all* selector variables in $\mathcal{P}_\mathcal{A} \stackrel{\text{def}}{=} \{s_{[a_j]} \mid a_j \in \mathcal{A}\}$ (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 $|\mathcal{T}|$ [6], 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 [6] 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 [6], 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 models. 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 [6], instead, 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 3: we notice that, once $\phi^{all}_{\mathcal{T}(po)}$ is generated, in order to evaluate different subsets $\mathcal{S} \backslash \{a_j\}$ 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: $\neg s_{[C_1 \sqsubseteq_\mathcal{T} D_1]}, \ldots, \neg s_{[C_j \sqsubseteq_\mathcal{T} D_j]}, \ldots$.

Ontology	notGalen	GeneOnt.	NCI	fullGalen	Snomed'09
# 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.

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 does not yet include all optimizations described here, and we performed a preliminary empirical evaluation of $\mathcal{EL}^+$SAT on the ontologies of §1. [8] We have implemented $\mathcal{EL}^+$SAT in C++, including and modifying the code of the SAT solver

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

MiniSat2.0 070721 [11]. 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`. [9]

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. [10] 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. [11] (Notice that $\phi_{\mathcal{T}}$ and $\phi^{one}_{\mathcal{T}}$ 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. [12] (Loading times are the same as above.) The seventh block reports the average times on 100 samples required to compute one MinA with $\phi^{one}_{\mathcal{T}}$, which computes the sequence of formulas $\phi^{one}_{\mathcal{T}}, \ldots, \phi^{one}_{\mathcal{S}\setminus a_i}, \ldots$. [13] (In order not to distinguish the loading time of the first formula with that of all the others, we report the sum the loading times; the process of loading of the first $\phi^{one}_{\mathcal{T}}$ can be shared by different samples.) The eighth block reports the average times on 100 samples required to compute one MinA with $\phi^{all}_{\mathcal{T}(po)}$. 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.[14]

Notice that, although huge, a Horn formula of up to 10^8 clauses is at the reach of a SAT solver like `MiniSAT` (e.g., in [19,20] we handled *non-Horn* formulas of $3.5 \cdot 10^7$ clauses).

Although still very preliminary, there 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 $\phi^{one}_{\mathcal{T}}$; (ii) in the computation of single MinAs, with both $\phi^{one}_{\mathcal{T}}$ and $\phi^{all}_{\mathcal{T}(po)}$, 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

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

[10] 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 $\phi^{one}_{\mathcal{T}}$. In the case of $\phi^{all}_{\mathcal{T}(po)}$ the times are not distinguishable.

[11] 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.

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

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

[14] 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.

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 Conclusions 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 optimizations presented here. (We plan to investigate and implement also a SAT-based versions of the techniques based on reachability modules of [7].) 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 and other reasoning services.

References

1. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ Envelope. In *Proc. IJCAI-05*, pages 364–369. Morgan-Kaufmann Publishers, 2005.
2. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ Envelope Further. In *In Proc. of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*, 2008.
3. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
4. F. Baader, C. Lutz, and B. Suntisrivaraporn. Efficient Reasoning in $\mathcal{EL}^+$. In *Proc. DL2006*, volume 189 of *CEUR-WS*, 2006.
5. F. Baader and R. Penaloza. Axiom pinpointing in general tableaux. In *Proceedings of TABLEAUX 2007*, LNAI, pages 11–27. Springer, 2007.
6. F. Baader, R. Peñaloza, and B. Suntisrivaraporn. Pinpointing in the Description Logic $\mathcal{EL}^+$. In *Proc. KI2007*, volume 4667 of *LNCS*, pages 52–67. Springer, 2007.
7. F. Baader and B. Suntisrivaraporn. Debugging SNOMED CT Using Axiom Pinpointing in the Description Logic $\mathcal{EL}^+$. In *Proc. KR-MED'08: Representing and Sharing Knowledge Using SNOMED*, volume 410 of *CEUR-WS*, 2008.
8. T. G. O. Consortium. Gene ontology: Tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.
9. M. Davis, G. Longemann, and D. Loveland. A machine program for theorem-proving. *Journal of the ACM*, 5(7):394–397, 1962.
10. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.
11. N. Eén and N. Sörensson. An Extensible SAT-solver. In *Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *LNCS*, pages 502–518. Springer, 2004.
12. B. Konev, D. Walther, and F. Wolter. The logical difference problem for description logic terminologies. In *Proceedings of IJCAR 2008, Sydney, Australia*, volume 5195 of *Lecture Notes in Computer Science*, pages 259–274. Springer, August 2008.

13. S. K. Lahiri, R. Nieuwenhuis, and A. Oliveras. SMT techniques for fast predicate abstraction. In *Proc. CAV*, volume 4144 of *LNCS*, pages 424–437. Springer, 2006.
14. I. Lynce and J. P. M. Silva. On Computing Minimum Unsatisfiable Cores. In *Proc. SAT*, 2004. `http://www.satisfiability.org/SAT04/programme/110.pdf`.
15. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *DAC*, pages 530–535. ACM, 2001.
16. B. Motik and I. Horrocks. Individual reuse in description logic reasoning. In *Proceedings of IJCAR 2008, Sydney, Australia*, volume 5195 of *Lecture Notes in Computer Science*, pages 242–258. Springer, August 2008.
17. 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.
18. R. Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation, JSAT*, 3:141–224, 2007.
19. R. Sebastiani and M. Vescovi. Encoding the Satisfiability of Modal and Description Logics into SAT: The Case Study of K(m)/ALC. In *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, 2006, Proceedings*, volume 4121 of *LNCS*, pages 130–135. Springer, 2006.
20. R. Sebastiani and M. Vescovi. Automated Reasoning in Modal and Description Logics via SAT Encoding: the Case Study of K(m)/ALC-Satisfiability. *Journal of Artificial Intelligence Research, JAIR.*, 35, June 2009.
21. J. P. M. Silva and K. A. Sakallah. GRASP - A new Search Algorithm for Satisfiability. In *Proc. ICCAD'96*, 1996.
22. 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.
23. K. A. Spackman. Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED RT. *J. of American Medical Informatics Association (Fall Symposium Special Issue)*, 2000.
24. K. A. Spackman, K. Campbell, and R. Cote. SNOMED RT: A reference terminology for healt care. *J. of American Medical Informatics Association (Fall Symposium Supplement)*, pages 640–644, 1997.
25. B. Suntisrivaraporn. *Polynomial-Time Reasoning Support for Design and Maintenance of Large-Scale Biomedical Ontologies*. PhD thesis, University of Dresden, 2009.
26. B. Suntisrivaraporn, F. Baader, S. Schulz, and K. Spackman. Replacing sep-triplets in snomed ct using tractable description logic operators. In *Proceedings of the 11th Conference on Artificial Intelligence in Medicine (AIME'07)*, LNCS. Springer-Verlag, 2007.
27. L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik. Efficient Conflict Driven Learning in Boolean Satisfiability Solver. In *ICCAD*, pages 279–285, 2001.
28. L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In *Proc. CAV'02*, number 2404 in LNCS, pages 17–36. Springer, 2002.

A Appendix: Proofs

In this section we define and prove formally all the results stated along this work

We start proving that solving the Horn-SAT problem $\phi_{\mathcal{T}}$ (see §3.1) under some assumptions we decide concept subsumption in $\mathcal{EL}^+$.

Theorem 1. *Given an $\mathcal{EL}^+$ TBox $\mathcal{T}$ in normal form, for every pair of concept names C, D in $\mathsf{PC}_{\mathcal{T}}$, $C \sqsubseteq_{\mathcal{T}} D$ if and only if the Horn propositional formula $\phi_{\mathcal{T}} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable.*

Proof. By construction the clause $p_{[C]} \rightarrow p_{[D]}$ (that is $\mathcal{EL}^+2sat(C \sqsubseteq D)$) is in $\phi_{\mathcal{T}}$ if and only if $C \sqsubseteq D$ is added to $\mathcal{A}$, that is, by the properties of the classification algorithm of §2.1, if and only if $C \sqsubseteq_{\mathcal{T}} D$. Thus, if $C \sqsubseteq_{\mathcal{T}} D$ then $\phi_{\mathcal{T}} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable.

Vice versa, if $\phi_{\mathcal{T}} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable, then it follows that $\phi_{\mathcal{T}} \models p_{[C]} \rightarrow p_{[D]}$. Thus, there must be a resolution derivation P of $p_{[C]} \rightarrow p_{[D]}$ from some subset of clauses $\psi_1, ..., \psi_n$ in $\phi_{\mathcal{T}}$. By construction, and since $\phi_{\mathcal{T}}$ is a definite Horn formula, every resolution step can be written in the form

$$\frac{(\bigwedge_i p_{[X_i]}) \rightarrow p_{[X_k]} \qquad (p_{[X_k]} \wedge \bigwedge_j p_{[X_j]}) \rightarrow p_{[X_n]}}{(\bigwedge_i p_{[X_i]} \wedge \bigwedge_j p_{[X_j]}) \rightarrow p_{[X_n]}} \tag{10}$$

s.t. all X's are concepts. Each corresponding derivation step

$$\frac{(\bigsqcap_i X_i) \sqsubseteq X_k \qquad (X_k \sqcap \bigsqcap_j X_j) \sqsubseteq X_n}{(\bigsqcap_i X_i \sqcap \bigsqcap_j X_j) \sqsubseteq X_n} \tag{11}$$

is valid in $\mathcal{EL}^+$; moreover, by construction, each ψ_i is $\mathcal{EL}^+2SAT(a_i)$ for some a_i which is either in $\mathcal{T}$ or has been derived from $\mathcal{T}$. Hence, there is a valid derivation of $C \sqsubseteq D$ from $\mathcal{T}$ in $\mathcal{EL}^+$, so that $C \sqsubseteq_{\mathcal{T}} D$. (Notice that from the latter fact we can also infer that $p_{[C]} \rightarrow p_{[D]}$ belongs to $\phi_{\mathcal{T}}$, that is, that the derivation P consists in no step.) □

Next, we prove that running DPLL on $\phi^{all}_{\mathcal{T}}$ under the assumption of the query literals and of the axiom selector variables still decides concept subsumption.

Theorem 2. *Given an $\mathcal{EL}^+$ TBox $\mathcal{T}$ in normal form, for every pair of concept names C, D in $\mathsf{PC}_{\mathcal{T}}$, $C \sqsubseteq_{\mathcal{T}} D$ if and only if the Horn propositional formula $\phi^{all}_{\mathcal{T}} \wedge \bigwedge_{a_i \in \mathcal{T}} s_{[a_i]} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable.*

Theorem 2 is a straightforward corollary of the following stronger result.

Theorem 3. *Given an $\mathcal{EL}^+$ TBox $\mathcal{T}$ in normal form, for every $\mathcal{S} \subseteq \mathcal{T}$ and for every pair of concept names C, D in $\mathsf{PC}_\mathcal{T}$, $C \sqsubseteq_\mathcal{S} D$ if and only if the Horn propositional formula $\phi^{all}_\mathcal{T} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable.*

Proof. Due to the soundness and completeness of the classification algorithm of §2.1, $C \sqsubseteq_\mathcal{S} D$ if and only if there exists a sequence of rule applications $r_1, \ldots, r_k$ generating $C \sqsubseteq D$ from $\mathcal{S}$. If and only if this is the case, by construction, $\phi^{all}_\mathcal{T}$ contains the clause

$$(s_{[C \sqsubseteq D]} \rightarrow (p_{[C]} \rightarrow p_{[D]})) \tag{12}$$

of type (8) and all the clauses of type (9) corresponding to all the rule applications $r_1, \ldots, r_k$.

Thus, on the one hand, if $C \sqsubseteq_\mathcal{S} D$, then $\bigwedge_{a_i \in \mathcal{S}} s_{[a_i]}$ and all the clauses of type (9) corresponding to all the rule applications $r_1, \ldots, r_k$ force $s_{[C \sqsubseteq D]}$ to be true and $p_{[C]} \wedge \neg p_{[D]}$ forces $p_{[C]}$ and $\neg p_{[D]}$ to be true, which falsifies the clause (12) in $\phi^{all}_\mathcal{T}$. Thus $\phi^{all}_\mathcal{T} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable.

On the other hand, suppose $\phi^{all}_\mathcal{T} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable. Since $\phi^{all}_\mathcal{T}$ is a definite Horn formula, $\phi^{all}_\mathcal{T} \wedge \bigwedge_{a_i \in \mathcal{S}}$ is satisfiable. Let $\phi^*_\mathcal{T}$ be the result of assigning in $\phi^{all}_\mathcal{T}$ all $s_{[a_i]}$ in $\mathcal{S}$ to $\top$ and unit-propagation the values. We notice that, by construction of $\phi^{all}_\mathcal{T}$ and $\phi^*_\mathcal{T}$, all and only the variables $s_{[a_i]}$ s.t. a_i can be derived from $\mathcal{S}$ are unit-propagated in this process. (Notice that $\phi^*_\mathcal{T}$ cannot be $\bot$ since $\phi^{all}_\mathcal{T} \wedge \bigwedge_{a_i \in \mathcal{S}}$ is satisfiable.) Thus $\phi^*_\mathcal{T}$ consists only on clauses in the forms:
(i) $\mathcal{EL}^+2SAT(a_i)$ s.t. a_i is derived from $\mathcal{S}$ (assertion clauses (8)),
(ii) $(s_{[a_i]} \rightarrow \mathcal{EL}^+2SAT(a_i))$ s.t. a_i is not derived from $\mathcal{S}$ (assertion clauses (8)), and
(iii) $(\bigwedge_i s_{[a_i]}) \rightarrow s_{[a_j]}$ s.t. a_i, a_j are not derived from $\mathcal{S}$ (rule clauses (9)).
Since $\phi^{all}_\mathcal{T} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable, $\phi^*_\mathcal{T} \wedge p_{[C]} \wedge \neg p_{[D]}$ is also unsatisfiable, i.e., $\phi^*_\mathcal{T} \models p_{[C]} \rightarrow p_{[D]}$. Thus, like in the proof of Theorem 1, there must be a resolution derivation P of $p_{[C]} \rightarrow p_{[D]}$ from some subset of clauses $\psi_1, ..., \psi_n$ in $\phi^*_\mathcal{T}$. We notice that all ψ_i's must be in form (i), because only the $\mathcal{EL}^+2SAT(a_i)$'s can contain $p_{[C]}, p_{[D]}$ and there would be no way of resolving away the selector variables $s_{[a_i]}$ from clauses (ii) and (iii) without reintroducing other ones. Thus, since $\phi^*_\mathcal{T}$ is a definite Horn formula, every resolution step in P can be written in the form

$$\frac{(\bigwedge_i p_{[X_i]}) \rightarrow p_{[X_k]} \qquad (p_{[X_k]} \wedge \bigwedge_j p_{[X_j]}) \rightarrow p_{[X_n]}}{(\bigwedge_i p_{[X_i]} \wedge \bigwedge_j p_{[X_j]}) \rightarrow p_{[X_n]}} \tag{13}$$

s.t. all X's are concepts. Each corresponding derivation step

$$\frac{(\bigsqcap_i X_i) \sqsubseteq X_k \qquad (X_k \sqcap \bigsqcap_j X_j) \sqsubseteq X_n}{(\bigsqcap_i X_i \sqcap \bigsqcap_j X_j) \sqsubseteq X_n} \tag{14}$$

is valid in $\mathcal{EL}^+$; moreover, by construction, each ψ_i is $\mathcal{EL}^+2SAT(a_i)$ for some a_i which is either in $\mathcal{S}$ or has been derived from $\mathcal{S}$. Hence, there is a valid derivation of $C \sqsubseteq D$ from $\mathcal{S}$ in $\mathcal{EL}^+$, so that $C \sqsubseteq_\mathcal{S} D$.

Notice also that from the latter fact we can also infer that $p_{[C]} \rightarrow p_{[D]}$ belongs to $\phi^*_{\mathcal{T}}$. Thus, (12) belongs to $\phi^{all}_{\mathcal{T}}$. □

The same result holds straightforwardly also for $\phi^{all}_{\mathcal{T}(po)}$ (see §3.2).

Theorem 4. *Given an $\mathcal{EL}^+$ TBox $\mathcal{T}$ in normal form, for every $\mathcal{S} \subseteq \mathcal{T}$ and for every pair of concept names C, D in $\mathsf{PC}_{\mathcal{T}}$, $C \sqsubseteq_{\mathcal{S}} D$ if and only if the Horn propositional formula $\phi^{all}_{\mathcal{T}(po)} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge \neg s_{[C \sqsubseteq D]}$ is unsatisfiable.*

Proof. Due to the soundness and completeness of the classification algorithm of §2.1, $C \sqsubseteq_{\mathcal{S}} D$ if and only if there exists a sequence of rule applications $r_1, \ldots, r_k$ generating $C \sqsubseteq D$ from $\mathcal{S}$. If and only if this is the case, by construction, $\phi^{all}_{\mathcal{T}(po)}$ contains all the clauses of type (9) corresponding to all the rule applications $r_1, \ldots, r_k$.

Thus, on the one hand, if $C \sqsubseteq_{\mathcal{S}} D$, then $\bigwedge_{a_i \in \mathcal{S}} s_{[a_i]}$ and all the clauses of type (9) corresponding to all the rule applications $r_1, \ldots, r_k$ force $s_{[C \sqsubseteq D]}$ to be true, which falsifies the unit clause $\neg s_{[C \sqsubseteq D]}$. Thus $\phi^{all}_{\mathcal{T}(po)} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge \neg s_{[C \sqsubseteq D]}$ is unsatisfiable.

On the other hand suppose $\phi^{all}_{\mathcal{T}(po)} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge \neg s_{[C \sqsubseteq D]}$ is unsatisfiable. Let $\phi^*_{\mathcal{T}}$ be the result of assigning in $\phi^{all}_{\mathcal{T}(po)}$ all $s_{[a_i]}$ in $\mathcal{S}$ to $\top$ and unit-propagation the values. (Notice that $\phi^*_{\mathcal{T}}$ cannot be $\bot$ since $\phi^{all}_{\mathcal{T}(po)} \wedge \bigwedge_{a_i \in \mathcal{S}}$ is satisfiable.) We notice that, by construction of $\phi^{all}_{\mathcal{T}(po)}$ and $\phi^*_{\mathcal{T}}$, all and only the variables $s_{[a_i]}$ s.t. a_i can be derived from $\mathcal{S}$ are unit-propagated in this process. Thus, if by absurd $C \sqsubseteq D$ could not be derived from $\mathcal{S}$, then $s_{[C \sqsubseteq D]}$ would not be unit-propagated in this process, so that $\phi^*_{\mathcal{T}} \wedge \neg s_{[C \sqsubseteq D]}$ would be satisfiable; hence $\phi^{all}_{\mathcal{T}(po)} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge \neg s_{[C \sqsubseteq D]}$ would be satisfiable, violating the hypothesis. □

Notice that, in one direction, the result of Theorem 3 holds also for $\phi^{one}_{\mathcal{T}}$ (see §3.2). In the other direction, instead, the result of Theorem 3 doesn't hold for every subset $\mathcal{S}$ of $\mathcal{T}$, but only for some of these subsets (since not all the possible sequence of rule applications deducing the same subsumption relation are encoded in $\phi^{one}_{\mathcal{T}}$).

Corollary 1. *Given an $\mathcal{EL}^+$ TBox $\mathcal{T}$ in normal form, for every pair of concept names C, D in $\mathsf{PC}_{\mathcal{T}}$, the following facts hold:*

(i) for every $\mathcal{S} \subseteq \mathcal{T}$, if $\phi^{one}_{\mathcal{T}} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable then $C \sqsubseteq_{\mathcal{S}} D$;

(ii) if $C \sqsubseteq_{\mathcal{T}} D$ then there exists at least a (possibly proper) subset $\mathcal{S} \subseteq \mathcal{T}$ such that $\phi^{one}_{\mathcal{T}} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable.

Proof.

(i) If $\phi^{one}_{\mathcal{T}} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable, then also if $\phi^{all}_{\mathcal{T}} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable, so that $C \sqsubseteq_{\mathcal{S}} D$ by Theorem 3.

(ii) If $C \sqsubseteq_{\mathcal{T}} D$, then the classification algorithm of §2.1 finds a sequence of rule applications $r_1, \ldots, r_k$ generating $C \sqsubseteq D$ from $\mathcal{S}$. If this is the case, by construction, $\phi_{\mathcal{T}}^{one}$ contains the assertion clause (12) and all the rule clauses corresponding to all the rule applications $r_1, \ldots, r_k$. Thus, $\bigwedge_{a_i \in \mathcal{T}} s_{[a_i]}$ and all the rule clauses corresponding to the rule applications $r_1, \ldots, r_k$ force $s_{[C \sqsubseteq D]}$ to be true and $p_{[C]} \wedge \neg p_{[D]}$ forces $p_{[C]}$ and $\neg p_{[D]}$ to be true, which falsifies the clause (12) in $\phi_{\mathcal{T}}^{one}$. Thus $\phi_{\mathcal{T}}^{one} \wedge \bigwedge_{a_i \in \mathcal{S}} s_{[a_i]} \wedge p_{[C]} \wedge \neg p_{[D]}$ is unsatisfiable for some $\mathcal{S} \subseteq \mathcal{T}$. □