ASP-based Axiom Pinpointing for Description Logics

Ignacio Huitzil[1], Giuseppe Mazzotta[2], Rafael Peñaloza[1] and Francesco Ricca[2]

[1]University of Milano-Bicocca, Italy
[2]University of Calabria, Italy

Abstract

Axiom pinpointing is the task of identifying the axiomatic causes for a consequence to follow from an ontology. Different approaches have been proposed in the literature for finding one or all justifications: the subset-minimal subontologies that preserve a description logic consequence. We propose an approach that leverages the capabilities of answer set programming—through minimal unsatisfiable subset enumeration—for transparent axiom pinpointing. Our approach is general in that it allows one to seamlessly specify the ontology and the reasoning rules, thus being applicable to different logics without modification to the core method. We showcase this generality by introducing methods for $\mathcal{EL}$ and for Horn-$\mathcal{ALC}$, and explaining how to apply it to other logics. A preliminary experiment shows the practicality of the approach.

Keywords

axiom-pinpointing, non-standard reasoning, ASP, consequence-based methods, Horn DLs

1. Introduction

Axiom pinpointing [1] refers to the task of identifying the axioms in an ontology that are responsible for a consequence to follow. It has been extensively studied in description logics (DLs) [2] and, under many different names, in other areas [3, 4]. Its basic goal is to enumerate one or all so-called *justifications*: the subset-minimal subontologies which still preserve the consequence of interest.

To-date, the most successful approach to axiom pinpointing which does not rely on repeated (black-box) calls to a standard reasoner is a reduction to the enumeration of the minimal unsatisfiable subsets (MUS) of clauses of a propositional formula [5, 6, 7]. In this approach, developed for the light-weight DL $\mathcal{EL}$, propositional variables are used to represent GCIs in normal form, and the propositional formula encodes the full execution of completion algorithm over the normalised TBox. This idea was improved upon in [8], where a more efficient encoding and a different enumeration process are used.

The main disadvantage of these approaches is that they require, as a pre-processing step, the construction of a huge propositional formula, which makes the reasoning steps explicit. In

DL 2023: 36th International Workshop on Description Logics, September 2–4, 2023, Rhodes, Greece
ignacio.huitzil@unimib.it (I. Huitzil); giuseppe.mazzotta@unical.it (G. Mazzotta); rafael.penaloza@unimib.it (R. Peñaloza); francesco.ricca@unical.it (F. Ricca)
https://rpenalozan.github.io/ (R. Peñaloza); https://www.mat.unical.it/ricca/ (F. Ricca)
0000-0002-7581-0345 (I. Huitzil); 0000-0003-0125-0477 (G. Mazzotta); 0000-0002-2693-5790 (R. Peñaloza); 0000-0001-8218-3178 (F. Ricca)

other words, the encoding step requires an implementation of the $\mathcal{EL}$ completion algorithm (or a consequence-based algorithm) with a modification to generate the propositional clauses. This in particular means that the approach cannot be easily adapted to different logics, as new implementations of the full decision procedure would be necessary. On the other hand, they have the advantage that any MUS enumerator can be applied on the derived formula, thus leveraging the fast developments from SAT [9].

Our goal is to keep the advantages of the reduction to MUS enumeration, while at the same time allowing a higher flexibility to handle different kinds of languages, and in particular, different DLs beyond $\mathcal{EL}$. We thus propose a new approach to axiom pinpointing based on a translation to Answer Set Programming (ASP) [10, 11]. The basic idea is to produce an ASP program which encodes the reasoning rules of the consequence based algorithm—rather than executing them directly. Within this setting, the MUSes of the generated ASP program [12] have a one-to-one correspondence with the justifications of the ontology. Our approach is general in the sense that it can be applied to any possible logical language which allows for a "modular" ASP representation. In a nutshell, a logic has a modular ASP representation if every axiom is translatable to a set of rules plus potentially a fixed finite program. The modular representation has the advantage of serving as a simple reasoner for the logic and, at a second step, as a tool for enumerating justifications. In addition, we can leverage the MUS enumerator to find justifications with special properties; e.g., with minimal cardinality, or to find the intersection of justifications.

As an example of its applicability, we instantiate our approach to deal with axiom pinpointing in Horn-$\mathcal{ALC}$ and its sublogic $\mathcal{EL}$. We also explain the changes to handle the very inexpressive $\mathcal{HL}$, which essentially corresponds to hypergraph reachability. We also implemented a tool called OWL2ASP, which translates (for now) $\mathcal{EL}$ ontologies to our ASP format, and calls a variant of the ASP solver WASP to enumerate the MUSes and thus the justifications. We present a first preliminary analysis of its performance on realistic ontologies.

The seed ideas of this paper were originally presented in [13]. This work extends that short paper by including the encoding for Horn-$\mathcal{ALC}$, by improving the call for the ASP MUS enumerator, and by describing the implemented encoder and the first (preliminary) empirical results.

2. Preliminaries

We start by briefly introducing the notions of answer set programming, DLs, and axiom pinpointing that are needed for understanding our approach.

2.1. Answer Set Programming

Answer set programming (ASP) [10, 14, 15] is a declarative programming paradigm based on stable models. Syntactically, it is based on rules and constraints.

In ASP, a *variable* is a string starting with uppercase letter; a *constant* is an integer or a string starting with lowercase letter; and an *atom* is an expression of the form $p(t_1, \ldots, t_n)$ where p is a predicate of arity n and $t_1, \ldots, t_n$ are terms. An atom is *ground* if it contains no variables. A *literal* is a atom a or its negation $not\ a$ where not denotes negation as failure. A

literal ℓ is *negative* if it is of the form $not\ a$, otherwise it is *positive*. The complement of positive (resp. negative) literal $\ell = a$ (resp. $\ell = not\ a$), denoted by $\bar{\ell}$, is the literal $not\ a$ (resp. a). A (normal) *rule* is an expression of the form $h \leftarrow b_1, \ldots, b_n$ where $b_1, \ldots, b_n$ is a conjunction of literals, called the *body*, $n \geq 0$, and h is an atom called the *head*. All variables in a rule must occur in some positive literal of the body (i.e., are safe cf. [16]). If r is a rule, then $head(r)$ is the head atom, and $body(r)$ is the set of all literals in the body. A *fact* is a rule with an empty body (i.e. $n = 0$). A *constraint* is a rule with an empty head; it is a shorthand for the rule $p \leftarrow b_1, \cdots, b_n, not\ p$, where p is a ground atom not occurring anywhere else. We will also use a special kind of *choice rule* which is of the form $\{a\}$ where a is an atom. In this case, we call a a *choice atom*. A *program* is a finite set of rules.

Given a program P, the *Herbrand Universe* U_P denotes the set of constants in P; the *Herbrand Base* B_P denotes the set of standard ground atoms that can be obtained from predicate in P and constants in U_P. Given a rule $r \in P$, $ground(r)$ denotes the set of possible rule instantiations that can be obtained by replacing variables in r with constants in U_P. The ground instantiation of the program P, denoted by $ground(P)$, is the union of ground instantiations of rules in P. An *interpretation* I is a subset of B_P. Given an interpretation I, a positive (resp. negative) literal ℓ is true w.r.t. I, if $\ell \in I$ (resp. $\bar{\ell} \notin I$); it is false if $\ell \notin I$ (resp. $\bar{\ell} \in I$). A conjunction of literals is true w.r.t. I if all the literals are true w.r.t. I. An interpretation I is a *model* of P if for each $r \in ground(P)$, the head of r is true whenever the body of r is true. The choice rule $\{a\}$ simply guesses whether a is made true or not. Given a program P and an interpretation I, the (Gelfond-Lifschitz) reduct [14]—P^I—is the program obtained from $ground(P)$ by (i) removing all those rules having in the body a false negative literal w.r.t. I, and (ii) removing negative literals from the body of remaining rules. Given a program P, the model I of P is a *stable model* or *answer set* if there is no $I' \subset I$ such that I' is a model of P^I. A program P is *coherent* if it admits at least one answer set, otherwise it is *incoherent*. ASP solvers decide whether a program is coherent and, in case, enumerate one or all its answer sets.

In the following sections, all ASP symbols are often denoted with a `typewriter` font, to distinguish them from other mathematical symbols, and rules use `:-` instead of $\leftarrow$.

2.2. Horn Description Logics

We assume familiarity with description logics (DLs) [2] but for clarity we introduce $\mathcal{EL}$ [17] and Horn-$\mathcal{ALC}$ [18]. These are used as prototypical examples, based on the fact that most axiom pinpointing tools focus on $\mathcal{EL}$. However, our goal is to describe a general approach that can be easily applied to other logics as well.

Starting from two infinite, disjoint sets N_C and N_R or *concept names* and *role names*, respectively, *$\mathcal{ALC}$-concepts* are constructed through the grammar rule

$$C ::= A \mid \top \mid \bot \mid \neg C \mid C \sqcap C \mid C \sqcup C \mid \exists r.C \mid \forall r.C,$$

where $A \in N_C$ and $r \in N_R$. A *GCI* is an expression of the form $C \sqsubseteq D$ where C and D are concepts. A *TBox* is a finite set of GCIs.

To define Horn-$\mathcal{ALC}$ we need the notion of *polarity* for each occurrence of a concept in an expression, which is defined recursively as follows. Each concept C occurs positively in C; if C occurs positively (resp. negatively) in C', then (i) it also occurs positively (resp. negatively)

Table 1
Consequence-based rules for Horn-$\mathcal{ALC}$. The first four rules cover the cases for $\mathcal{EL}$.

if $A \sqsubseteq C, C \sqsubseteq B \in \mathcal{T}$	then add $A \sqsubseteq B$
if $A \sqsubseteq C_1, A \sqsubseteq C_2, C_1 \sqcap C_2 \sqsubseteq B \in \mathcal{T}$	then add $A \sqsubseteq B$
if $A \sqsubseteq \exists r.B, B \sqsubseteq C \in \mathcal{T}$	then add $A \sqsubseteq \exists r.C$
if $A \sqsubseteq \exists r.C, \exists r.C \sqsubseteq B \in \mathcal{T}$	then add $A \sqsubseteq B$
if $A \sqsubseteq \exists r.C_1, A \sqsubseteq \forall r.C_2, C_1 \sqcap C_2 \sqsubseteq B \in \mathcal{T}$	then add $A \sqsubseteq \exists r.B$
if $A \sqsubseteq \exists r.B, B \sqsubseteq \bot \in \mathcal{T}$	then add $A \sqsubseteq \bot$

in $C' \sqcap D$, $C' \sqcup D$, $\exists r.C'$, $\forall r.C'$ and $D \sqsubseteq C'$ (considering commutativity of $\sqcap, \sqcup$) and (ii) it occurs negatively (resp. positively) in $\neg C'$ and $C' \sqsubseteq D$. Horn-$\mathcal{ALC}$ is the sublogic of $\mathcal{ALC}$ in which axioms with positive occurrences of $C \sqcup D$ or negative occurrences of $\neg C$ or $\forall r.C$ are disallowed. $\mathcal{EL}$ is another sublogic of $\mathcal{ALC}$, where the constructors $\neg$, $\sqcup$, $\forall$, and $\bot$ are not allowed. In particular, $\mathcal{EL}$ is also a fragment of Horn-$\mathcal{ALC}$. As an additional example, we consider also the fragment $\mathcal{HL}$ (also known as $\mathcal{L}_0$) of $\mathcal{EL}$ in which only concept names and conjunctions ($\sqcap$) are allowed.

The semantics of $\mathcal{ALC}$ (and its fragments) is defined through *interpretations*, which are pairs $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $\Delta^{\mathcal{I}}$ a non-empty set called *domain* and $\cdot^{\mathcal{I}}$ is the *interpretation function* mapping every $A \in N_C$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and every $r \in N_R$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended to concepts setting $\top^{\mathcal{I}} := \Delta^{\mathcal{I}}$, $(\neg C)^{\mathcal{I}} := \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$, $(C \sqcap D)^{\mathcal{I}} := C^{\mathcal{I}} \cap D^{\mathcal{I}}$, and $(\exists r.C)^{\mathcal{I}} := \{\delta \mid \exists \eta \in C^{\mathcal{I}}.(\delta, \eta) \in r^{\mathcal{I}}\}$. The other constructors are treated analogously. The interpretation $\mathcal{I}$ *satisfies* the GCI $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. It is a *model* of $\mathcal{T}$ iff it satisfies all GCIs in $\mathcal{T}$. We often call GCIs *axioms*, which allows us to seamlessly connect between DL and arbitrary representation languages. The TBox $\mathcal{T}$ *entails* the GCI $C \sqsubseteq D$ ($\mathcal{T} \models C \sqsubseteq D$) iff every model of $\mathcal{T}$ satisfies $C \sqsubseteq D$. In this case we say that $C \sqsubseteq D$ is a *consequence* of $\mathcal{T}$.

We focus on the problem of deciding *atomic* subsumption relations $A \sqsubseteq B$ where $A, B \in N_C$. The consequence-based algorithm for Horn-$\mathcal{ALC}$ [18] first transforms the TBox to *normal form*. A GCI is in normal form if it has one of the following shapes:

$$A \sqsubseteq C, \quad A \sqcap A' \sqsubseteq C, \quad A \sqsubseteq \exists r.B, \quad \exists r.A \sqsubseteq B, \quad A \sqsubseteq \forall r.B$$

where $r \in N_R$ and $A, A', B \in N_C \cup \{\top\}, C \in N_C \cup \{\bot\}$. Once the TBox $\mathcal{T}$ is in normal form, the consequence-based algorithm applies the *completion rules* in Table 1 to add new consequences until saturation. It is shown in [19] that the method is sound and complete for all atomic subsumptions over the concept names appearing in the original TBox.

In the case of $\mathcal{EL}$, the last kind of normal form axiom will never appear, and the concept C appearing in the right-hand side of the first two kinds can never be $\bot$. Hence, the first four rules in Table 1 suffice for atomic subsumption in this logic. For practical reasons, in this case we will unify two rules into one with a larger set of preconditions, which will suffice for deciding atomic subsumptions. It can be seen that $\mathcal{HL}$ is a syntactic variant of directed hypergraphs. Specifically, a GCI $A_1 \sqcap \cdots \sqcap A_m \sqsubseteq B_1 \sqcap \cdots \sqcap B_n$ represents a directed hypergraph connecting nodes $A_1, \ldots, A_m$ with nodes $B_1, \ldots, B_n$, and the entailment problem is nothing more than reachability in this hypergraph.

Beyond standard reasoning, it is sometimes important to understand which axioms are responsible for a consequence to follow from a TBox. This goal is often instantiated as the task of identifying *justifications*.

Definition 1. *A* justification *for a consequence α w.r.t. the TBox $\mathcal{T}$ is a set $\mathcal{M} \subseteq \mathcal{T}$ such that (i) $\mathcal{M} \models \alpha$ and (ii) for every $\mathcal{N} \subset \mathcal{M}$, $\mathcal{N} \not\models \alpha$.*

Briefly, a justification is a subset-minimal sub-TBox that still entails the consequence. Importantly, justifications refer to the GCIs in the original TBox before normalisation, which means an additional minimisation step if the pinpointing method works on normalised TBoxes [20].

Most work focuses on computing one or all justifications. While the former problem remains polynomial in $\mathcal{EL}$, the latter necessarily needs exponential time, as the number of justifications may be exponential on the size of the TBox. Despite some potential uses, which have been identified for non standard reasoning [21], only very recently have specific algorithms for computing the unions and intersection of justifications been developed [22, 23]. To the best of our knowledge, no previous work has considered computing the justifications of *minimal cardinality* directly.

A similar problem is observed in ASP. When a program is incoherent, one may want to find the minimal sub-programs that are still incoherent. This notion, analogous to justifications in DLs, is called a *minimal unsatisfiable subset* (MUS) in the ASP community [12]. Rather than giving the same weight to all facts and rules in the program, one variant is to limit the search to a subclass of facts of interest and assume that the rest of the program is fixed and hence, irrelevant for the explanation. We exploit this analogy and solve axiom pinpointing through MUS enumeration.

3. Reasoning Through Rules

Before presenting our approach to axiom pinpointing using ASP, we briefly describe how to reduce reasoning in Horn-$\mathcal{ALC}$ to ASP. The approach simulates the completion algorithm sketched in Section 2.2 through a small set of rules, while the TBox axioms (in normal form) are represented through facts.

Consider a TBox $\mathcal{T}$ in normal form, and let $\mathcal{C}(\mathcal{T})$ and $\mathcal{R}(\mathcal{T})$ be the sets of concept names and role names appearing in $\mathcal{T}$, respectively. For each $A \in \mathcal{C}(\mathcal{T})$ we use a constant `a`, and for each $r \in \mathcal{R}(\mathcal{T})$ we use a constant `r`. We also use the numerical constants `0` and `1` for $\bot$ and $\top$, respectively. We identify the four shapes of normal form axioms via a different predicate `s0`, ..., `s4` as shown in Figure 1 (left). Hence, `s0(a,b)` stands for the GCI $A \sqsubseteq B$ and so on. For each axiom in normal form appearing in $\mathcal{T}$, we write the associated fact. As previously mentioned, the reasoning process is simulated through rules. In the specific case of Horn-$\mathcal{ALC}$, these rules are shown in the right-hand side of Figure 1. As before, if the TBox belongs to $\mathcal{EL}$, then the first four rules suffice (and the last two can be removed from the program). For efficiency reasons, in the $\mathcal{EL}$ encoding we merge the third and fourth rules into one. To decide whether the atomic subsumption $A \sqsubseteq B$ is a consequence of the TBox, we need only ask the query `s0(a,b)`; i.e., check whether this fact is derivable from the initial facts, applying the rules. Since the original TBox may not be in normal form, the facts obtained this way are the result of

GCI	ASP fact
$A \sqsubseteq C$	`s0(a,c).`
$A_1 \sqcap A_2 \sqsubseteq C$	`s1(a1,a2,c).`
$A \sqsubseteq \exists r.B$	`s2(a,r,b).`
$\exists r.A \sqsubseteq B$	`s3(r,a,b).`
$A \sqsubseteq \forall r.B$	`s4(a,r,b).`

```
s0(X,Y)   :- s0(X,Z), s0(Z,Y).
s0(X,Y)   :- s0(X,Z1), s0(X,Z2), s1(Z1,Z2,Y).
s2(X,R,Y) :- s2(X,R,Z), s0(Z,Y).
s0(X,Y)   :- s2(X,R,Z), s3(R,Z,Y).
```

```
s2(X,R,Y) :- s2(X,R,Z1), s4(X,R,Z2), s1(Z1,Z2,Y).
s0(X,0)   :- s2(X,R,Y), s0(Y,0).
```

Figure 1: Expressing Horn-$\mathcal{ALC}$ GCIs in normal form as ASP facts (left) and the reasoning rules (right).

the normalisation step over the original GCIs. As we will see later, we can use additional rules to preserve the origin information for each normalised GCI, which will be useful for axiom pinpointing.

In the case of $\mathcal{HL}$, one can produce a more direct reduction, which takes into account the hyperedges without the need for normalisation or general derivation rules. We again represent each concept name A through a constant `a`, and associate a new constant `gi` for each GCI in $\mathcal{T}$. Then the GCI $A_1 \sqcap \cdots \sqcap A_m \sqsubseteq B_1 \sqcap \cdots \sqcap B_n$ is translated to the set of rules $\{$`gi :- a1,...,am.,b1 :- gi.,...,bn :- gi.`$\}$. To decide whether $A \sqsubseteq B$ is a consequence, we add the fact `a.` and verify the query `b`. The correctness of these approaches follows from the results in [19, 24, 25].

4. Axiom Pinpointing Through ASP

We present a general approach for solving axiom pinpointing tasks through an ASP solver. The approach is applicable to any logic (including other DLs) which allows for a *modular* ASP encoding. Roughly, an encoding is a function that maps TBoxes (or more generally, knowledge bases) to ASP programs, and it is modular if each axiom in $\mathcal{T}$ translates to a set of rules, such that an ASP encoding $\Pi_\mathcal{T}$ of $\mathcal{T}$ is obtained by the union of the encodings of its axioms, possibly together with some additional rules (independent of the specific axioms in $\mathcal{T}$) needed to simulate reasoning in ASP.

Definition 2. *An encoding in ASP $\Pi_\mathcal{T}$ is* modular *iff for every TBox $\mathcal{T}$ it holds that (i) for each $\alpha \in \mathcal{T}$ there is an ASP program Π_α, and (ii) there is a (possibly empty) set of rules R such that $\Pi_\mathcal{T} = \bigcup_{\alpha\in\mathcal{T}} \Pi_\alpha \cup R$*

The encodings from Section 3 for Horn-$\mathcal{ALC}$, $\mathcal{EL}$ and $\mathcal{HL}$ are all modular. In the first two cases, R is the set of rules in Figure 1 (right), while in the latter $R = \emptyset$. For any (arbitrary) Horn-$\mathcal{ALC}$ GCI α, Π_α is the ASP representation of its translation to normal form. We now formulate the problem of computing all justifications in ASP. First, we apply an *adornment* step, which allows to identify and keep track of the rules of a module corresponding to a given axiom, necessary for pinpointing.

Definition 3. *Let P be an ASP program, and δ be an atom not occurring in P. The δ-adornment for P is the program $\Delta_\delta(P) = \{r_\delta \mid r \in P\}$, where r_δ is such that $head(r_\delta) = head(r)$, and $body(r_\delta) = body(r) \cup \delta$.*

In words, the δ-adornment adds a new identifying atom δ to the body of each rule of the program. This guarantees that the rules *trigger* only when δ is true. In the case of Horn-$\mathcal{ALC}$, which means that axioms in normal form will be considered only if the original axiom that produces them is considered.

Definition 4. *The adorned ASP encoding of the TBox $\mathcal{T}$ is the program*

$$\delta(\Pi_{\mathcal{T}}) = \bigcup_{\alpha \in \mathcal{T}} \Delta_{\delta_\alpha}(\Pi_\alpha) \cup R \cup C$$

where for each $\alpha \in \mathcal{T}$, δ_α is a fresh atom not occurring in $\Pi(\mathcal{T})$, and C is the ASP program containing a choice rule $\{\delta_\alpha\}$ for each $\alpha \in \mathcal{T}$.

In the case of Horn-$\mathcal{ALC}$ and $\mathcal{EL}$, the adornment will change each fact (corresponding to a GCI in normal form) `si(_).` into the rule `si(_) :- xj`, where `xj` is the chosen constant for the original (DL) axiom α_j. Importantly, this approach handles the original axioms in the TBox, and not those already normalised as done e.g. in [20].

We now describe an ASP program that can be used for axiom pinpointing. Note that $A \sqsubseteq B$ is a consequence of $\mathcal{T}$ if and only if the program $\delta(\Pi_{\mathcal{T}})$ entails the atom `s0(a,b).` Thus, if we add the constraint `:-s0(a,b).` the resulting program will be incoherent. To find out the justifications for $A \sqsubseteq B$ (the minimal subset of axioms that entail it), it suffices to find the minimal subsets of adornments—which refer to the original axioms in $\mathcal{T}$—that preserve incoherence. In other words, the MUSes.

Proposition 1. *Let $\mathcal{T}$ be a TBox, c an atom modelling a consequence of $\mathcal{T}$, and P the program $P = \delta(\Pi_{\mathcal{T}}) \cup \{\leftarrow c\}$. $\mathcal{M} \subseteq \mathcal{T}$ is a justification for c iff there is a MUS M of P w.r.t. $\{\delta_\alpha \mid \alpha \in \mathcal{T}\}$ such that $\{\delta_\alpha \mid \alpha \in \mathcal{M}\} = M \cap \{\delta_\alpha \mid \alpha \in \mathcal{T}\}$.*

Justifications that are cardinality minimal (and thus also subset minimal) can be directly computed using an ASP program with weak constraints. In a nutshell, a *weak constraint* $:\sim$a tries to exclude the atom a from any model, but can be introduced if necessary. In this case, the MUS enumerator tries to minimise the number of weak constraints that are violated which corresponds to minimising the cardinality of the associated justification.

Proposition 2. *Let $\mathcal{T}$ be a TBox, c an atom modelling a consequence of $\mathcal{T}$, and P the program $P = \delta(\Pi_{\mathcal{T}}) \cup \{\leftarrow c\} \cup \{:\sim \delta_\alpha \mid \alpha \in \mathcal{T}\}$. $\mathcal{M} \subseteq \mathcal{T}$ is a cardinality-minimal justification for c iff there exists an optimal MUS M of P such that $\{\delta_\alpha \mid \alpha \in \mathcal{M}\} = M \cap \{\delta_\alpha \mid \alpha \in \mathcal{T}\}$.*

Before describing our implementation and empirical evaluation, we note that the translation permits computing the *intersection* of all justifications, and consequences derived from it, through the application of *cautious reasoning* [15]. In ASP, a cautious consequence is one that holds in every answer set. Since the program P from Proposition 1 provides a one-to-one correspondence between answer sets and sub-ontologies deriving a consequence, cautious reasoning refers to reasoning over the intersection of all those sub-ontologies, and in particular over the subset-minimal ones; that is, over the justifications. Unfortunately, an analogous result does not exist for the *union* of all justifications. Indeed, every axiom would be available for *brave reasoning* (consequences which hold in at least one answer set) [15] over the same program P, but not all axioms belong to some justification.

5. OWL2ASP

To verify the practicality of our approach, we implemented a tool called OWL2ASP, which translates an $\mathcal{EL}$ ontology into the ASP syntax described in Section 3 and then calls an ASP solver (after an adequate adorning) to obtain all the justifications for a given query. For the first part of the translation, the tool reuses the normalisation module of jcel [26] and improves upon the functionality of OwlToProlog[1] by correcting some issues on the use of $\top$ and standardising the output syntax. The parsing and manipulation of axioms and concepts uses the elements available through the OWL API. We implemented OWL2ASP in Java. The system and other relevant information is publicly available at https://board.unimib.it/datasets/r6xcggwvjp/1.

The translator takes an $\mathcal{EL}$ TBox and produces five files which include all the information about the original axioms, the normalised GCIs they produce, and a mapping from concept and role names (in DL) and constants (in ASP). As it is standard, we use integers as ASP constants to reduce the consumption of memory and other resources. Specifically, for a given TBox $\mathcal{T}$, OWL2ASP generates (i) a file asp.txt which contains the ASP translation of the normalisation of $\mathcal{T}$ together with the information of which original axioms produced them using a new predicate symbol `o(_)` to better capture the origin; (ii) originalAxioms.txt containing an (integer) ID for each OWL-represented axiom in the original ontology; (iii) owlElements.txt relating the concept and role names with unique integer IDs; (iv) axiom_facts.asp enumerating the IDs of original axioms for aiding the ASP MUS enumerator; and (v) encoding.asp enumerating the reasoning rules and the query of interest.

Note that in the current implementation, the file encoding.asp is the same for any possible input TBox modulo the requested query: it always contains the first four rules shown in Figure 1 (right).[2] Yet, here lies the flexibility in our approach: to handle other formalisms, one needs only to change this file with the corresponding rules, and—perhaps the largest effort—implement the normalisation procedure for the new language. The rest of the approach remains unchanged, which means that ASP features can be used without issues.

The resulting files can be fed to an ASP solver to verify that the consequence expressed in the query follows from the TBox. But our goal is rather to enumerate the justifications for this consequence. Thus, the tool adds a choice rule for each atom representing an *original* GCI from $\mathcal{T}$, and calls the MUS enumerator of WASP[3] [27] to obtain one or all MUSes. These MUSes are encoded using the constants from the ASP program. The information in the file originalAxioms.txt can be used to map this information back to a justification in the original OWL syntax of the TBox. The output of the MUS enumerator is available for a direct visualization by the user, or in a file which can be fed to other tools in e.g., a repair pipeline if necessary. A full execution example is available at the public repository for OWL2ASP. As mentioned already, through modifications in the call to WASP, it is possible to obtain also the justifications of minimal *cardinality* and their intersection. We can also take advantage of optimisations developed for ASP applications, as we see in the next section.

[1] https://github.com/Gallax619/OwlToProlog
[2] Recall that we merge the third and fourth rules for efficiency reasons.
[3] http://alviano.github.io/wasp/

6. Preliminary Experiment

We executed a preliminary experiment to empirically assess the potential of our method. For this experiment we consider the well-known biomedical ontology NOT-GALEN, which is among the *de facto* benchmarks for axiom pinpointing in $\mathcal{EL}$. More precisely, we consider the version provided in jcel,[4] which features 4379 axioms, 2748 concepts, and 413 roles, and chose a random sample of 25 queries on this ontology already considered in the literature. The task we consider is the enumeration of all justifications for each query.

Our tool, called OWL2ASP, employs the encoding from Figure 1 targeted for $\mathcal{EL}$, with the third and fourth rules combined into one. The resulting program was first processed through the magic sets technique [28] using a direct call to the DLV2 system [29, 30]. The adornment process (see Section 4) was applied to the resulting program, and WASP [31] was called for MUS enumeration [12].

We compare with other glass-box axiom pinpointing tools for $\mathcal{EL}$. In particular, we run three methods based on a translation to SAT originally proposed by Sebastiani and Vescovi—namely EL2SAT [5, 32], BEACON [6], and SATPIN [7]—and the resolution-based PULi [8] (configured with option SAT and strategy THRESHOLD). The experiment was executed on an Intel(R) Xeon(R) CPU E7-8880 v4 @ 2.20GHz running Debian Linux (4.9.0-19-amd64), with memory and time limits of 16GB and 800 seconds, respectively.

The results of the execution are reported in Table 2, which shows the solving time in seconds for each query and tool. For the cells denoted by **< 0.01**, the execution time was below the sampling range of the observation wrapper we used. For our tool (OWL2ASP) we report the time needed by WASP to perform MUS enumeration without considering the pre-processing time to obtain propositional programs. This is to create a fair comparison with the SAT-based tools whose input is already a SAT formula encoding the full standard reasoning process. On the one hand, we report that OWL2ASP requires 0.8s on average (with a standard deviation of 0.15) to pre-process the input over all the 25 queries compared, which is a positive result. On the other hand, we observe that the experiment highlights the viability of the proposed approach; indeed, OWL2ASP results to be the fastest in terms of solving time for this sample, providing answers almost instantaneously.

7. Conclusions

We presented a general approach for axiom pinpointing based on a reduction to ASP. Similar to existing SAT-based approaches, we construct an ASP program which, if extended with a constraint encoding a consequence of interest, becomes incoherent; that is, has no models. Then, an ASP MUS enumerator is used to find justifications efficiently. Importantly, even though the reasoning step requires normalisation, our approach enumerates directly on the original axioms in the TBox, thus avoiding an additional (potentially expensive) recovery step [20].

As a proof of concept, we have shown how the reduction works for the DL Horn-$\mathcal{ALC}$, and its sublogics $\mathcal{EL}$ and $\mathcal{HL}$. The same idea is applicable for any logic with a modular translation to ASP, for instance any DL with a consequence-based reasoning algorithm [18, 33] should

[4] http://julianmendez.github.io/jcel/

Table 2
Tool evaluation: Execution times in seconds for 25 queries on NOT-GALEN.

QUERY	BEACON	EL2SAT	OWL2ASP	PULI	SATPIN
1.not.galen	0.21	timeout	**< 0.01**	1.16	**< 0.01**
2.not.galen	0.27	timeout	**< 0.01**	1.21	0.12
3.not.galen	0.52	timeout	**< 0.01**	1.37	0.15
4.not.galen	0.56	timeout	**< 0.01**	1.12	0.16
5.not.galen	0.57	timeout	**< 0.01**	1.36	**< 0.01**
6.not.galen	0.27	timeout	**< 0.01**	1.34	**< 0.01**
7.not.galen	0.27	timeout	**< 0.01**	1.09	**< 0.01**
8.not.galen	0.26	timeout	**< 0.01**	1.06	**< 0.01**
9.not.galen	0.24	timeout	**< 0.01**	0.78	**< 0.01**
10.not.galen	0.50	timeout	**< 0.01**	1.54	**< 0.01**
11.not.galen	0.51	timeout	**< 0.01**	1.60	0.01
12.not.galen	0.49	timeout	**< 0.01**	1.60	**< 0.01**
13.not.galen	0.53	timeout	**< 0.01**	1.65	**< 0.01**
14.not.galen	0.52	timeout	**< 0.01**	1.61	0.02
15.not.galen	0.50	timeout	**< 0.01**	1.38	0.05
16.not.galen	0.53	timeout	**< 0.01**	1.34	0.08
17.not.galen	0.53	timeout	**< 0.01**	1.36	0.04
18.not.galen	0.57	timeout	**< 0.01**	1.32	0.04
19.not.galen	0.53	timeout	**< 0.01**	1.50	0.08
20.not.galen	0.50	timeout	**< 0.01**	1.38	0.15
21.not.galen	0.52	timeout	**< 0.01**	1.39	0.13
22.not.galen	0.53	timeout	**< 0.01**	1.41	0.13
23.not.galen	0.52	timeout	**< 0.01**	1.45	0.17
24.not.galen	0.24	timeout	**< 0.01**	1.40	0.14
25.not.galen	0.24	timeout	**< 0.01**	1.43	0.15

enjoy such a translation. The challenge in these cases is to find an adequate expression of the consequence-based rules which can be handled by the ASP solver. For instance, the consequence-based method for (full) $\mathcal{ALC}$ manipulates sets, which are not native to ASP. Dealing with further constructors imposes additional challenges. On the other hand, once such an encoding has been devised, our approach can be seamlessly applied, without further implementation issues. In contrast, existing methods based on SAT [5, 6, 7], need to explicitly execute all the reasoning steps to produce their encoding.

To test the applicability of our method, we implemented OWL2ASP, a tool that produces the corresponding encoding for $\mathcal{EL}$ TBoxes, and compare it with other glass-box methods for pinpointing in this logic. Our results are very promising, they show that our tool is competitive with state-of-the-art alternatives in a preliminary experiment considering 25 queries executed on NOT-GALEN. For future work, we plan to extend the evaluation to a larger sample of queries and ontologies, following existing benchmarks. We will also extend the implementation to handle Horn-$\mathcal{ALC}$.

An advantage of using an off-the-shelf ASP solver is that we can directly take advantage of all the services that these solvers provide without any additional cost. In particular, through a simple modification to the MUS enumerator call, one can obtain justifications of minimal cardinality. In the future, we plan to study the implementation of other axiom pinpointing services based on ASP constructs and evaluate their practical applicability. Moreover, we plan to further reduce the (already acceptable) pre-processing times by exploiting the modularity properties of the ontology in input, in order to specialise the reasoning task only to those modules that are involved in the input query.

Acknowledgments

This work was partially supported by MUR for the Department of Excellence DISCo at the University of Milano-Bicocca and under PRIN project PINPOINT Prot. 2020FNEB27, CUP H23C22000280006 and H45E21000210001. We would like to thank Carmine Dodaro for his support using the MUS enumerator.

References

[1] S. Schlobach, R. Cornet, Non-standard reasoning services for the debugging of description logic terminologies, in: Proceedings of IJCAI'03, Morgan Kaufmann Publishers Inc., 2003, pp. 355–360.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation, and Applications, second ed., Cambridge University Press, 2007.

[3] A. Kalyanpur, B. Parsia, E. Sirin, J. A. Hendler, Debugging unsatisfiable classes in OWL ontologies, Journal of Web Semantics 3 (2005) 268–293. doi:10.1016/j.websem.2005.09.005.

[4] R. Peñaloza, Explaining axiom pinpointing, in: C. Lutz, U. Sattler, C. Tinelli, A. Turhan, F. Wolter (Eds.), Description Logic, Theory Combination, and All That - Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday, volume 11560 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 475–496. URL: https://doi.org/10.1007/978-3-030-22102-7_22. doi:10.1007/978-3-030-22102-7_22.

[5] R. Sebastiani, M. Vescovi, Axiom Pinpointing in Large EL+ Ontologies via SAT and SMT Techniques, DISI Technical report, University of Trento, 2015.

[6] M. F. Arif, C. Mencía, A. Ignatiev, N. Manthey, R. Peñaloza, J. Marques-Silva, BEACON: an efficient SAT-based tool for debugging *EL+* ontologies, in: Proceedings of SAT 2016., volume 9710 of *LNCS*, Springer, 2016, pp. 521–530.

[7] N. Manthey, R. Peñaloza, S. Rudolph, SATPin: Axiom pinpointing for lightweight description logics through incremental SAT, Künstliche Intelligenz 34 (2020) 389–394. doi:https://doi.org/10.1007/s13218-020-00669-4.

[8] Y. Kazakov, P. Skocovský, Enumerating justifications using resolution, in: D. Galmiche, S. Schulz, R. Sebastiani (Eds.), Proceedings of the 9th International Joint Conference on

Automated Reasoning (IJCAR 2018), volume 10900 of *LNCS*, Springer, 2018, pp. 609–626. doi:10.1007/978-3-319-94205-6_40.

[9] M. H. Liffiton, A. Previti, A. Malik, J. Marques-Silva, Fast, flexible MUS enumeration, Constraints 21 (2016) 223–250. URL: https://doi.org/10.1007/s10601-015-9183-0. doi:10.1007/s10601-015-9183-0.

[10] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103.

[11] V. Lifschitz, Answer set planning, in: D. D. Schreye (Ed.), Logic Programming: The 1999 International Conference, MIT Press, 1999, pp. 23–37.

[12] M. Alviano, C. Dodaro, S. Fiorentino, A. Previti, F. Ricca, ASP and subset minimality: Enumeration, cautious reasoning and MUSes, Artif. Intell. 320 (2023) 103931. URL: https://doi.org/10.1016/j.artint.2023.103931. doi:10.1016/j.artint.2023.103931.

[13] R. Peñaloza, F. Ricca, Pinpointing axioms in ontologies via ASP, in: Proc. of the 16th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2022, volume 13416 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 315–321. doi:10.1007/978-3-031-15707-3_24.

[14] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Generation Comput. 9 (1991) 365–386.

[15] C. Baral, Knowledge Representation, Reasoning and Declarative Problem Solving, Cambridge University Press, 2010.

[16] S. Ceri, G. Gottlob, L. Tanca, Logic Programming and Databases, Surveys in computer science, Springer, 1990.

[17] F. Baader, S. Brandt, C. Lutz, Pushing the $\mathcal{EL}$ envelope, in: Proceedings of IJCAI'05, Professional Book Center, 2005, pp. 364–369.

[18] F. Simančík, Y. Kazakov, I. Horrocks, Consequence-based reasoning beyond Horn ontologies, in: Proc. IJCAI'11, AAAI Press, 2011, pp. 1093–1098.

[19] Y. Kazakov, Consequence-driven reasoning for horn SHIQ ontologies, in: C. Boutilier (Ed.), Proceedings of the 21st International Joint Conference on Artificial Intelligence, 2009, pp. 2040–2045. URL: http://ijcai.org/Proceedings/09/Papers/336.pdf.

[20] F. Baader, R. Peñaloza, B. Suntisrivaraporn, Pinpointing in the description logic EL+, in: Proc. of KI'07, volume 4667 of *LNCS*, Springer, 2007, pp. 52–67. doi:10.1007/978-3-540-74565-5_7.

[21] M. Bienvenu, R. Rosati, Tractable approximations of consistent query answering for robust ontology-based data access, in: F. Rossi (Ed.), Proceedings of IJCAI'13, AAAI Press/IJCAI, 2013, pp. 775–781.

[22] J. Chen, Y. Ma, R. Peñaloza, H. Yang, Union and intersection of all justifications, in: Proc. of ESWC 2022, volume 13261 of *LNCS*, Springer, 2022, pp. 56–73.

[23] R. Peñaloza, C. Mencía, A. Ignatiev, J. Marques-Silva, Lean kernels in description logics, in: Proceeding of the 14th Semantic Web Conference (ESWC 2017), volume 10249 of *LNCS*, 2017, pp. 518–533. doi:10.1007/978-3-319-58068-5_32.

[24] R. Peñaloza, B. Sertkaya, Understanding the complexity of axiom pinpointing in lightweight description logics, Artif. Intell. 250 (2017) 80–104.

[25] İ. İ. Ceylan, J. Mendez, R. Peñaloza, The Bayesian ontology reasoner is BORN!, in: M. Dumontier, B. Glimm, R. S. Gonçalves, M. Horridge, E. Jiménez-Ruiz, N. Matentzoglu,

B. Parsia, G. B. Stamou, G. Stoilos (Eds.), Proceedings of the 4th International Workshop on OWL Reasoner Evaluation (ORE-2015), volume 1387 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015, pp. 8–14. URL: http://ceur-ws.org/Vol-1387/paper_5.pdf.

[26] J. Mendez, jcel: A modular rule-based reasoner, in: I. Horrocks, M. Yatskevich, E. Jiménez-Ruiz (Eds.), Proceedings of the 1st International Workshop on OWL Reasoner Evaluation (ORE-2012), Manchester, UK, July 1st, 2012, volume 858 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2012.

[27] M. Alviano, C. Dodaro, S. Fiorentino, A. Previti, F. Ricca, Enumeration of minimal models and MUSes in WASP, in: Proceedings of the 16th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2022, volume 13416 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 29–42. doi:10.1007/978-3-031-15707-3_3.

[28] F. Bancilhon, D. Maier, Y. Sagiv, J. D. Ullman, Magic sets and other strange ways to implement logic programs, in: PODS, ACM, 1986, pp. 1–15.

[29] M. Alviano, F. Calimeri, C. Dodaro, D. Fuscà, N. Leone, S. Perri, F. Ricca, P. Veltri, J. Zangari, The ASP system DLV2, in: LPNMR, volume 10377 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 215–221.

[30] M. Alviano, N. Leone, P. Veltri, J. Zangari, Enhancing magic sets with an application to ontological reasoning, Theory Pract. Log. Program. 19 (2019) 654–670.

[31] M. Alviano, G. Amendola, C. Dodaro, N. Leone, M. Maratea, F. Ricca, Evaluation of disjunctive programs in WASP, in: LPNMR, volume 11481 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 241–255.

[32] R. Sebastiani, M. Vescovi, Axiom pinpointing in lightweight description logics via Horn-SAT encoding and conflict analysis, in: R. A. Schmidt (Ed.), Automated Deduction – CADE-22, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 84–99.

[33] D. T. Cucala, B. C. Grau, I. Horrocks, Consequence-based reasoning for description logics with disjunction, inverse roles, and nominals, in: Proceedings of DL 2017, Montpellier, France, July 18-21, 2017., 2017.