A Dependency-Graph based Approach for Finding Justification in OWL 2 EL[☆]

Zhangquan Zhou[*], Guilin Qi

School of Computer Science, Southeast University, Nanjing, China

Abstract

The ontology language OWL 2 EL, is designed for knowledge modeling and has been widely used in real applications. However, modeling knowledge as OWL ontologies is an error-prone process, where logical errors or contradictions would be imported. Further, it is almost impossible to manually find errors occurring in large-scale ontologies. *Finding justification*, an important service for error pinpointing, has then attracted much attention of researchers and developers. However, current methods of finding justifications suffer from an issue: *high-coupling of reasoners*, i.e., there is a tight relation between reasoners and the task of finding justifications. This makes the performance of finding justifications be highly influenced by the utilized reasoner, and, it is also restricted to optimize the algorithms effectively. In order to tackle this problem, we consider giving a method such that the task of finding justifications is independent from the utilized reasoner. Specifically, we propose a kind of graph called *Explanation Dependency Graph* (EDG) which guides to compute justifications from the reasoning results directly. We further give several optimizing strategies and prove the correctness of our method. We implement our method and evaluate it on real ontologies, including SNOMED CT. The experimental results show that our method is practical and performs better than current methods.

Keywords:
OWL 2 EL, Classification, Justification, Graph

[☆]This paper is an extension of our previous work [1].

[*]Corresponding author

Email addresses: quanzz1129@gmail.com (Zhangquan Zhou), gqi@seu.edu.cn (Guilin Qi)

1. Introduction

The Web Ontology Language OWL has been designed as one of the major standards for formal knowledge representation and automated reasoning in the Semantic Web. The most recent version of OWL is OWL 2[1]. One of the main advantages of employing OWL in real applications is that reasoning services can be used to optimize query answering, ontology diagnosis and debugging. Among different OWL sub-languages, OWL 2 EL (OWL EL for short) has attracted much attention of researchers and developers. This is because OWL EL has a polynomial computational complexity of reasoning and a sufficient expressive power in medicine applications. For example, some large medical ontologies like SNOMED CT [2] and Gene Ontology[2] (GO) can be expressed in OWL EL. Recently, OWL EL has also been used for traffic congestion diagnosing [3]. In these applications based on OWL EL, *classification* is the main reasoning service, which is the task of computing a subsumption hierarchy between concepts.

As OWL has begun to be used in many real applications, a lot of developers and users have participated in ontology building and editing. However, the development and maintenance of large-scale ontologies are complicated and error-prone. Some "unwanted" reasoning results (or *entailments*) may be derived from an ontology containing logic errors or contradictions. Given an unwanted entailment for a large-scale ontology like SNOMED CT with nearly six hundred thousand *axioms* (statements for concepts and properties), it is almost impossible to find the erroneous axioms responsible for it manually. Thus, it is necessary to provide explanation services to automatically find the erroneous axioms. *Finding justifications* is a service of this kind, which helps users or developers to understand an entailment by presenting minimal subsets (called justifications) of the target ontology which are responsible for the entailment.

The existing methods of finding justifications can be classified into two categories: *glass-box* methods and *black-box* methods. The glass-box methods trace how an entailment is derived based on reasoning information from the internals of a reasoner. This can be done by modifying the implemen-

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

[2] http://geneontology.org/

tations of the reasoning algorithms for tracing the derivation of entailments (see [4, 5, 6]). Since the utilized reasoners need to be modified for glass-box methods, some key optimizing strategies (e.g., pruning techniques, optimized encoding) cannot be further used [7]. The black-box methods treat a reasoner as an "oracle" and use it to check the *satisfiability* of an ontology or the derivability of an entailment (see [6, 8, 7]). In this way, justifications can also be computed. However, black-box algorithms typically require a bunch of calls of reasoners. This makes it difficult to find justifications in limited time for large ontologies even when some optimization strategies are utilized [8, 9, 10]. In summary, both of black-box methods and glass-box methods are highly-coupled with reasoners, i.e., there is a tight relation between reasoners and the task of finding justifications. This makes the performance of finding justifications be highly influenced by the utilized reasoner, and, it is also restricted to optimize the algorithms effectively.

Motivated by the above issue, we consider extending our previous work [1] and giving a method of finding justifications for OWL EL in the way the task of finding justifications is independent from the utilized reasoner. It is challenging to give such a method due to two requirements: 1) reasoners are disallowed to be modified to extract internal information of the reasoning procedure; 2) reasoners cannot be used during the task of finding justifications. Current methods cannot work under the above requirements. To satisfy the requirements, we propose a kind of graph called *Explanation Dependency Graph* (EDG) where each node in the graph is a set of ontology axioms and entailments that can derive the target entailment. Intuitively, EDG can replace reasoners to provide reasoning information that can further guide to find justifications. We show how to find justifications based on EDG without using a reasoner. In this way, the utilized reasoner needs a single run for the purpose of classification and does not need to be modified. We also give several optimizing strategies and prove the correctness of our method. We implement this method and evaluate it on real ontologies, including SNOMED CT. The experimental results show that our algorithm is practical and performs better than current implementations.

The rest of the paper is organized as follows. We first introduce basic notions of the description logic that underpins OWL EL in Section 2. We discuss related works in Section 3. We then introduce the method based on EDG in Section 4 and 5. We further give optimization strategies in Section 6. After that, we present implementation and evaluation results in Section 7. Finally, we conclude this paper in Section 8.

2. Background Knowledge

In this section, we introduce the notions that are used in this paper.

2.1. The Description Logic: $\mathcal{EL}^+$

We first introduce the syntax and semantics of $\mathcal{EL}^+$ [11] which is the basic description logic (DL) that underpins OWL EL. Let **CN** and **RN** be two disjoint sets of *atomic concepts* and *atomic roles* respectively. We can further define *complex concepts* inductively by using the constructors ($\exists$ and $\sqcap$) shown in the left upper part of Table 1. We use the symbols $C_{(i)}$, $D_{(i)}$ and $E_{(i)}$ for complex concepts, $A_{(i)}$ and $B_{(i)}$ for atomic concepts, $r_{(i)}$, $s_{(i)}$ and $t_{(i)}$ for atomic roles. The symbol $\top$ denotes the *top concept* (or universal concept). A DL ontology is always defined as a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is called a TBox containing a finite set of *general concept inclusion* (GCI) axioms of the form $C \sqsubseteq D$, *role inclusion* (RI) axioms of the form $r \sqsubseteq s$ and *role chain* (RC) axioms of the form $r_1 \circ r_2 \sqsubseteq s$; $\mathcal{A}$ is called an ABox that contains assertions about individuals (e.g., $\texttt{hasMum}(\texttt{Jack}, \texttt{Helen})$, `hasMum` is a role, `Jack` and `Helen` are individuals). Since individual is not our focus, *we assume that an $\mathcal{EL}^+$ ontology is a TBox in this paper.*

Table 1: syntax and semantics of $\mathcal{EL}^+$

Name	Syntax	Semantics
top concept	$\top$	$\triangle^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$\{x \in \triangle^{\mathcal{I}} \mid \exists y \in \triangle^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \wedge r \in C^{\mathcal{I}}\}$
GCI	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
RI	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
RC	$r_1 \circ r_2 \sqsubseteq s$	$\forall x, y, z$, if $(x,y) \in r_1^{\mathcal{I}}$ and $(y,z) \in r_2^{\mathcal{I}}$, then $(x,z) \in s^{\mathcal{I}}$ holds.

$\mathcal{EL}^+$ has a Tarski-style semantics. An *interpretation* $\mathcal{I} = (\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of $\triangle^{\mathcal{I}}$ and $\cdot^{\mathcal{I}}$, where $\triangle^{\mathcal{I}}$ is the domain of $\mathcal{I}$ which is a non-empty set, and $\cdot^{\mathcal{I}}$ is a function mapping each atomic concept A (resp. atomic role r) to a subset $A^{\mathcal{I}}$ (resp. a subset $r^{\mathcal{I}}$) of $\triangle^{\mathcal{I}}$ (resp. $\triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$). The interpretation function can be extended to arbitrary concept or role as shown in right part of Table 1. $\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ (written as $\mathcal{I} \models \mathcal{O}$) if $\mathcal{I}$ satisfies all axioms in $\mathcal{O}$. We use $\mathcal{O} \models \alpha$ for some axiom α if each model of $\mathcal{O}$ satisfies α.

We say that, for an axiom α, if α is occurring in $\mathcal{O}$, α is an *original axiom*; if α is not occurring in $\mathcal{O}$ but $\mathcal{O} \models \alpha$ holds, α is an *entailment*. We also say that a concept C is *subsumed* by D with respect to $\mathcal{O}$ if $\mathcal{O} \models C \sqsubseteq D$.

2.2. Finding Justifications

A justification is a minimal set of axioms in an ontology that is responsible for a target axiom. Finding all justifications is to compute all such minimal axiom sets with respect to a given axiom. We follow the definition of justification for highly expressive description logics [7], and give the formal definition in our case as follows:

Definition 1. ***(Justification)*** *Given a DL ontology $\mathcal{O}$. Suppose $\mathcal{O} \vDash \alpha$ for some axiom α. A subset $\mathcal{O}'$ of O is a justification for α in $\mathcal{O}$, if $\mathcal{O}' \vDash \alpha$, and $\mathcal{O}'' \nvDash \alpha$ for every $\mathcal{O}'' \subset \mathcal{O}'$.*

Example 1. *Given an $\mathcal{EL}^+$ ontology $\mathcal{O}_{ex1}$ consisting of the following axioms:*

$$
\begin{array}{lrcl}
(\alpha_1) & \texttt{Apple} & \sqsubseteq & \exists \texttt{beInvestedBy}.(\texttt{Fidelity} \sqcap \texttt{BlackStone}) \\
(\alpha_2) & \exists \texttt{beFundedBy}.\texttt{Fidelity} & \sqsubseteq & \texttt{InnovativeCompanies} \\
(\alpha_3) & \exists \texttt{beFundedBy}.\texttt{BlackStone} & \sqsubseteq & \texttt{InnovativeCompanies} \\
(\alpha_4) & \texttt{beInvestedBy} & \sqsubseteq & \texttt{beFundedBy}
\end{array}
$$

The above axioms describe investment relationships between companies: α_1 says that the company Apple is invested by both of the investment companies Fidelity and BlackStone; α_2 and α_3 say that the companies funded by Fidelity and BlackStone are innovative companies respectively; α_4 states the subsumption between the two roles `beInvestedBy` *and* `beFundedBy`*. We consider the justifications of* $\texttt{Apple} \sqsubseteq \texttt{InnovativeCompanies}$ *with respect to the ontology $\mathcal{O}_{ex1}$. It can be checked that there are two justifications of* $\texttt{Apple} \sqsubseteq \texttt{InnovativeCompanies}$*. They are $\mathcal{J}_1 = \{\alpha_1, \alpha_2, \alpha_4\}$ and $\mathcal{J}_2 = \{\alpha_1, \alpha_3, \alpha_4\}$. For simplicity, in what follows, we use* `bib` *to replace* `beInvestedBy`*,* `bfb` *to* `beFundedBy`*,* `Fidel` *to* `Fidelity`*,* `Black` *to* `BlackStone` *and* `InnoCmp` *to* `InnovativeCompanies`*.*

2.3. The Classification Task

The task of finding justifications is tightly related to the corresponding reasoning task. In this part, we introduce the task of *classification* which is an important reasoning task of $\mathcal{EL}^+$. The goal of the classification task

is to compute all concept subsumptions. Intuitively, finding justifications can be viewed an inverse process of classification. In other words, for some subsumption $A \sqsubseteq B$ (it is in the classification results), to find justifications of $A \sqsubseteq B$ is actually to obtain the original axioms that entail $A \sqsubseteq B$.

A state-of-the-art reasoning system (referred as CEL) [12] is given to perform classification of $\mathcal{EL}^+$ ontologies. The classification algorithm used in CEL first transforms the given ontology to a normal form, where all concept inclusions are of the form $A_1 \sqcap A_2 \sqsubseteq B$, $A \sqsubseteq \exists r.B$ or $\exists r.B \sqsubseteq A$ ($A_{(i)}, B$ are atomic concepts), and all role inclusions are of the form $r_1 \circ r_2 \sqsubseteq s$ or $r \sqsubseteq s$. The normalization can be done in linear time. *In the following, we assume that an input ontology $\mathcal{O}$ is in normal form.* The classification algorithm then works on a group of rules. We call these rules the CEL rules (see Table 2). The concepts occurring in the CEL rules denote atomic concepts. A set $\mathcal{C}_\mathcal{O}$ is used in the CEL rules. Initially, let $\mathcal{C}_\mathcal{O} := \mathcal{O} \cup \bigcup_{\forall A \in \mathbf{CN}} \{A \sqsubseteq A, A \sqsubseteq \top\}$. The classification algorithm then extends $\mathcal{C}_\mathcal{O}$ by exhaustively applying all the CEL rules. After classification, $\mathcal{C}_\mathcal{O}$ contains all subsumptions. We call $\mathcal{C}_\mathcal{O}$ of this stage the set of *classification results* with respect to $\mathcal{O}$. Without special statements, we use $\mathcal{C}_\mathcal{O}$ to denote the set of classification results in following paper.

Table 2: The CEL Rules for $\mathcal{EL}^+$ Classification

$\mathbf{R}_\sqcap$	If $X \sqsubseteq A_1$, $X \sqsubseteq A_2 \in \mathcal{C}_\mathcal{O}$, $A_1 \sqcap A_2 \sqsubseteq B \in \mathcal{O}$, and $X \sqsubseteq B \notin \mathcal{C}_\mathcal{O}$, then $\mathcal{C}_\mathcal{O} := \mathcal{C}_\mathcal{O} \cup \{X \sqsubseteq B\}$.
$\mathbf{R}^+_\exists$	If $X \sqsubseteq A \in \mathcal{C}_\mathcal{O}$, $A \sqsubseteq \exists r.B \in \mathcal{O}$, and $X \sqsubseteq \exists r.B \notin \mathcal{C}_\mathcal{O}$, then $\mathcal{C}_\mathcal{O} := \mathcal{C}_\mathcal{O} \cup \{X \sqsubseteq \exists r.B\}$.
$\mathbf{R}^-_\exists$	If $X \sqsubseteq \exists r.A, A \sqsubseteq B \in \mathcal{C}_\mathcal{O}$, $\exists r.B \sqsubseteq C \in \mathcal{O}$, and $X \sqsubseteq C \notin \mathcal{C}_\mathcal{O}$, then $\mathcal{C}_\mathcal{O} := \mathcal{C}_\mathcal{O} \cup \{X \sqsubseteq C\}$.
$\mathbf{R}_\mathtt{H}$	If $A \sqsubseteq \exists r.B \in \mathcal{C}_\mathcal{O}$, $r \sqsubseteq s \in \mathcal{O}$, and $A \sqsubseteq \exists s.B \notin \mathcal{C}_\mathcal{O}$, then $\mathcal{C}_\mathcal{O} := \mathcal{C}_\mathcal{O} \cup \{A \sqsubseteq \exists s.B\}$.
$\mathbf{R}_\circ$	If $A \sqsubseteq \exists r.B, B \sqsubseteq \exists s.C \in \mathcal{C}_\mathcal{O}$, $r \circ s \sqsubseteq t \in \mathcal{O}$, and $A \sqsubseteq \exists t.C \notin \mathcal{C}_\mathcal{O}$, then $\mathcal{C}_\mathcal{O} := \mathcal{C}_\mathcal{O} \cup \{A \sqsubseteq \exists t.C\}$.

Example 2. *We consider using the method of CEL to classify the ontology $\mathcal{O}_{ex1}$. In the ontology $\mathcal{O}_{ex1}$ only the axiom (α_1) is needed to be transformed to a norm form. Specifically, (α_1) is transformed to three axioms:* (α_5) $\mathtt{Apple} \sqsubseteq \exists \mathtt{bib}.\mathtt{FB}$, ($\alpha_6$) $\mathtt{FB} \sqsubseteq \mathtt{Fidel}$, *and* ($\alpha_7$) $\mathtt{FB} \sqsubseteq \mathtt{Black}$, *where* FB *is a new introduced concept name. We give a fragment of the procedure of*

classifying $\mathcal{O}_{ex1}$ using the CEL rules as follows:

$$\texttt{Apple} \sqsubseteq \texttt{Apple} \qquad \textit{initialization} \qquad (1)$$

$$\texttt{FB} \sqsubseteq \texttt{FB} \qquad \textit{initialization} \qquad (2)$$

$$\texttt{Apple} \sqsubseteq \exists\texttt{bib.FB} \qquad \textit{by } \mathbf{R}_{\overset{+}{\exists}} \textit{ to } (1) \textit{ and } (\alpha_5) \qquad (3)$$

$$\texttt{FB} \sqsubseteq \texttt{Fidel} \qquad \textit{by } \mathbf{R}_{\sqsubseteq} \textit{ to } (2) \textit{ and } (\alpha_6) \qquad (4)$$

$$\texttt{FB} \sqsubseteq \texttt{Black} \qquad \textit{by } \mathbf{R}_{\sqsubseteq} \textit{ to } (2) \textit{ and } (\alpha_7) \qquad (5)$$

$$\texttt{Apple} \sqsubseteq \exists\texttt{bfb.FB} \qquad \textit{by } \mathbf{R}_{\texttt{H}} \textit{ to } (3) \textit{ and } (\alpha_4) \qquad (6)$$

$$\texttt{Apple} \sqsubseteq \texttt{InnoCmp} \qquad \textit{by } \mathbf{R}_{\overset{-}{\exists}} \textit{ to } (6), (4) \textit{ and } (\alpha_2) \qquad (7)$$

$$\texttt{Apple} \sqsubseteq \texttt{InnoCmp} \qquad \textit{by } \mathbf{R}_{\overset{-}{\exists}} \textit{ to } (6), (5) \textit{ and } (\alpha_3) \qquad (8)$$

The classification starts with some tautological conclusions for concept names (see (1) and (2)) and then repeatedly applies the CEL rules. The entailments (3)-(8) are added to the set of classification results $\mathcal{C}_{\mathcal{O}}$ after applying these rules.

3. Related Work

In this section, we introduce the related issues and notions of finding justifications, including what is the problem of finding justifications originated from, and the major approaches proposed for tackling it. Finally, we discuss the advantages and drawbacks of current approaches.

Issues. Constructing and maintaining ontologies is a difficult and error-prone process [13, 14, 15]. Logical errors or contradictions are quite common in real ontologies, specially, when the target ontologies are large in scale [14, 16]. At first, logical errors or contradictions denote the problems of *incoherence* for an ontology. Specifically, a TBox is *incoherent* [5] whenever there are *unsatisfiable concepts*: concepts which are interpreted as empty sets in all models of the TBox. The authors of [5] first give the formal definitions of unsatisfiability and propose the notions of MUPS (minimal unsatisfiability-preserving sub-TBoxes) and MIPS (minimal incoherence-preserving sub-TBoxes). A MUPS, with respect to a TBox $\mathcal{T}$ and an unsatisfiable concept A, is a smallest subset of $\mathcal{T}$ in which A is unsatisfiable. A MIPS is a smallest subset of an original TBox preserving unsatisfiability of at least one concept. The MIPS is used to identify the smallest sets of TBox that cause the original TBox to be incoherent. Thus, to tackle the

problem of incoherence (or to do ontology debugging), an important task is to compute MUPS and MIPS for an incoherent ontology. '*Justification*' is first used in [17] as a generalized notion of MUPS. It can be computed for any conclusion in addition to the contradicting ones. Actually, in some cases, users or developers wish to know how a conclusion can be obtained from the original ontology, though this ontology is well-constructed and contains no logical error. The techniques of finding justifications can be used in ontology debugging [17, 18, 19], entailment understanding [20, 21, 22] and even verifying reasoner correctness [23, 24].

In the following, we organize the discussion of the major approaches by two lines based on highly-coupled reasoners: *black-box* and *glass-box* methods.

Black-Box Methods. In black-box methods, a reasoner is used as a black-box (or an oracle) for certain reasoning tasks, e.g., classification, testing satisfiability, and etc. A typical black-box method focus on detecting dependencies between unsatisfiable concepts, i.e., identifying the root concept that causes the incoherence and the concepts derived from it [19, 25, 4]. This method suffers from low performance when computing all justifications (or MUPS). Researchers then adapt the techniques used in the contexts of model-based *diagnosis* [26] to the case of ontology debugging. Specifically, a diagnose specifies the minimal set of axioms that have to be ignored in order to turn the terminology coherent. Diagnoses can be computed by constructing a tree that is called *hitting set tree* (HST). HST is first used to compute MUPS in [27]. In [7] and [28], this method is used in highly expressive OWL languages, and evaluated on small ontologies. There is also works that uses HST techniques to translate the problem of ontology debugging to SAT problem [29, 30]. Some other optimizing strategies are further applied to accelerate the computation based on black-box methods, such as module extraction [8, 9] and other heuristic approaches [31, 32, 33].

Glass-Box Methods. In glass-box methods, reasoning information from the internals of a reasoner is extracted and presented to the developers for tracing how a conclusion is derived. That is why these methods are called glass-box methods. The work of [5] takes the first attempt to compute MUPS in the description logic $\mathcal{ALC}$ based on a DL reasoner. The used reasoner is based on a tabular algorithm, which materializes ontologies by obtaining all possible individual assertions. Specifically, the method proposed in [5] requires to modify the implementation of the reasoner RACER such that each assertion is labeled by a set of axioms that are responsible for it. After

materialization, these labels are backtracked to compute MUPS. For the tractable fragments of description logics, i.e., $\mathcal{EL}$ and DL-lite, the similar methods are studied based on high performance reasoners (cf. [6, 34]). Some other techniques are proposed to optimize glass-box methods [35, 36].

Discussion. In following paragraphs, we discuss the advantages and drawbacks of black-box and glass-box methods. For black-box methods, they have one advantage over glass-box ones: no need for a specialized, explanation generating reasoner. However, to compute MUPS or justifications, a number of calls for reasoners is always necessary. Since reasoning tasks are inherently expensive in terms of running time, black-box methods can hardly work on large ontologies. This is also shown by current experimental results.

For glass-box methods, the biggest difference from black-box ones lies in that they just need single run of reasoners. However, glass-box methods still suffer from some problems: 1) A glass-box method requires to modify the implementation of reasoner. Some optimizations (like pruning techniques, optimized encoding) for reasoners have to be dropped in order to record special information for finding justifications. Furthermore, different modifications are needed when applying different reasoners. 2) Extra data structures should be maintained. This brings additional space and computation consumption. 3) It is hard to introduce some heuristic blocking techniques to speed up the computation of finding justifications. This is because, when backtracking attached tags, it is difficult for us to acquire global information that can guide us to avoid failed branches.

In summary, both of black-box and glass-box methods suffer from a common issue: they are highly-coupled with reasoners, i.e., the performance of finding justifications is highly influenced by that of the utilized reasoner. Thus, some optimizations cannot work as discussed above. In order to tackle this problem, we consider giving a method such that the task of finding justifications can be conducted independently from the utilized reasoner, and thus, cannot be influenced by the performance of the reasoner. In the following sections, we discuss this method in detail.

4. Explanation and EDG

Our target is to give a method of finding justifications in the way that the task of finding justifications is independent from the utilized reasoner. Thus, two requirements should be satisfied: first, reasoners are disallowed to be modified to extract internal information of the reasoning procedure;

second, reasoners cannot be used during the task of finding justifications. In other words, we should give a method of finding justifications with the original ontology and the set of classification results as the only information that we can use.

Driven by the above requirements, we attempt to find a kind of data-structure that has two features: 1) it can be computed from the original ontology and the set of classification results; 2) it can be further processed to compute justifications. To this end, we first introduce a notion of *explanation*. An explanation for some entailment α is a subset of the classification results, $\mathcal{E} \subseteq \mathcal{C}_{\mathcal{O}}$, where $\mathcal{C}_{\mathcal{O}}$ is the set of classification results and $\mathcal{E} \models \alpha$. An explanation has the above two features. On one hand, an explanation can be computed from the set of classification results $\mathcal{C}_{\mathcal{O}}$ even through roughly checking each subset of $\mathcal{C}_{\mathcal{O}}$. On the other hand, given all possible explanations, justifications can be further identified by preserving minimal subsets of original axioms according to the definition of justification. In the following, we first give the formal definition of *explanation* and then specify our ideas of finding justifications.

Definition 2. ***(Explanation)*** *Given a normalized $\mathcal{EL}^+$ ontology $\mathcal{O}$ and an axiom α. $\mathcal{C}_{\mathcal{O}}$ is the set of classification results with respect to $\mathcal{O}$. An explanation $\mathcal{E}$ of α is a set of axioms such that $\mathcal{E} \subseteq \mathcal{C}_{\mathcal{O}}$ and $\mathcal{E} \models \alpha$. We use $\mathcal{E}^{\mathtt{ea}}$ to denote the subset of $\mathcal{E}$ which contains all entailments in $\mathcal{E}$ and $\mathcal{E}^{\mathtt{oa}}$ to denote the subset which contains all original axioms in $\mathcal{E}$. We use $\mathbf{E}_{\mathcal{O},\alpha}$ to denote the set of all explanations with respect to α and $\mathcal{C}_{\mathcal{O}}$.*

Example 3. *Let $\mathcal{O}_{ex1}^{\mathtt{norm}}$ denote the normalized form of the ontology $\mathcal{O}_{ex1}$ in Example 1. Let $\mathcal{C}_{\mathcal{O}_{ex1}}$ be the set of classification results with respect to $\mathcal{O}_{ex1}^{\mathtt{norm}}$. One can check that: ($e_1$) $\{\mathtt{Apple} \sqsubseteq \mathtt{InnoCmp}\}$, ($e_2$) $\{\mathtt{Apple} \sqsubseteq \exists \mathtt{bfb.FB}, \mathtt{FB} \sqsubseteq \exists \mathtt{Fidel}, \exists \mathtt{bfb.Fidel} \sqsubseteq \mathtt{InnoCmp}\}$, ($e_3$) $\{\mathtt{Apple} \sqsubseteq \exists \mathtt{bfb.FB}, \mathtt{FB} \sqsubseteq \exists \mathtt{Black}, \exists \mathtt{bfb.Black} \sqsubseteq \mathtt{InnoCmp}\}$, ($e_4$) $\{\mathtt{Apple} \sqsubseteq \exists \mathtt{bib.FB}, \mathtt{bib} \sqsubseteq \mathtt{bfb}, \mathtt{FB} \sqsubseteq \exists \mathtt{Fidel}, \exists \mathtt{bfb.Fidel} \sqsubseteq \mathtt{InnoCmp}\}$ are some explanations of $\mathtt{Apple} \sqsubseteq \mathtt{InnoCmp}$, and are the subsets of $\mathcal{C}_{\mathcal{O}_{ex1}}$.*

It is obvious that for an entailment α, the singleton set $\{\alpha\}$ and all justifications of α are also the explanations of α.

Based on explanations, we can organize our method of finding justifications as two steps: for some entailment α, Step 1: compute all explanations of α; Step 2: identify the real justifications of α by filtering the explanations obtained in the first step.

We next discuss how to complete Step 1. That is, for some entailment, to compute its explanations. We conduct our work based on an observation: explanations can be computed by using the CEL rules. Recall Example 3. Given the explanation (e_2), we can get the explanation (e_4) through an application of Rule $\mathbf{R}_{\mathtt{H}}$ over $\mathtt{Apple} \sqsubseteq \exists \mathtt{bib}.\mathtt{FB}$ and $\mathtt{bib} \sqsubseteq \mathtt{bfb}$. It seems that we 'unfold' (e_2) based on Rule $\mathbf{R}_{\mathtt{H}}$ and get (e_4). Inspired by this observation, we can obtain all explanations by starting from a root explanation $\{\mathtt{Apple} \sqsubseteq \mathtt{InnoCmp}\}$ and unfolding all other explanations through different rules. To formulate this idea, we first define a relation, called *deductive dependency*, between two explanations, and further give a kind of graph that guides the work of unfolding.

Definition 3. ***(Deductive Dependency)*** *Given two explanations $\mathcal{E}_1$ and $\mathcal{E}_2$. We say that $\mathcal{E}_2$* deductively depends *on $\mathcal{E}_1$ if for any axiom $\alpha \in \mathcal{E}_2$: (1) $\alpha \in \mathcal{E}_1$, or (2) there exists a subset $\mathcal{E}' \subseteq \mathcal{E}_1$ s.t. α holds by applying some CEL rule on $\mathcal{E}'$ once.*

In Example 3, we can say that (e_2) deductively depends on (e_4). We use $\mathbf{D}_{\mathcal{O},\alpha} \subseteq \mathbf{E}_{\mathcal{O},\alpha} \times \mathbf{E}_{\mathcal{O},\alpha}$ to denote all possible deductive dependencies over the explanation set $\mathbf{E}_{\mathcal{O},\alpha}$. Specifically, for each $(\mathcal{E}_1, \mathcal{E}_2) \in \mathbf{D}_{\mathcal{O},\alpha}$, we have that $\mathcal{E}_1$ deductively depends on $\mathcal{E}_2$. By grouping deductive dependencies between explanations, we actually have a directed graph that is defined as follows.

Definition 4. ***(Explanation Dependency Graph (EDG))*** *Given a normalized $\mathcal{EL}^+$ ontology $\mathcal{O}$ and an axiom α of $\mathcal{O}$, an EDG with respect to $\mathcal{O}$ and α is a directed graph $\mathcal{G} = (V, E)$, where $V = \{V_{\mathcal{E}} : \mathcal{E} \in \mathbf{E}_{\mathcal{O},\alpha}\}$ is the set of nodes in $\mathcal{G}$ and each node $V_{\mathcal{E}}$ corresponds to an explanation set $\mathcal{E}$. $E = \{(V_{\mathcal{E}_1}, V_{\mathcal{E}_2}) : V_{\mathcal{E}_1}, V_{\mathcal{E}_2} \in V$ and $\mathcal{E}_1$ deductively depends on $\mathcal{E}_2\}$.*

Example 4. *We give an EDG $\mathcal{G}_{ex1}$ of $\mathtt{Apple} \sqsubseteq \mathtt{InnoCmp}$ in Figure 1. For simplicity, we use the corresponding identifiers to denote axioms. One can check that, the labels of each node in $\mathcal{G}_{ex1}$ is an explanation of $\mathtt{Apple} \sqsubseteq \mathtt{InnoCmp}$; the edges represent deductive dependencies between these explanations.*

Without special statements, we do not distinguish the nodes in an EDG and the explanations. We say that a *complete EDG* w.r.t $\mathcal{O}$ and α is an EDG containing all justifications of α as its nodes. We also say that the node $V_{\{\alpha\}}$ is the *root node* in the EDG $\mathcal{G}_{\mathcal{O},\alpha}$. We use the following lemma to show that,

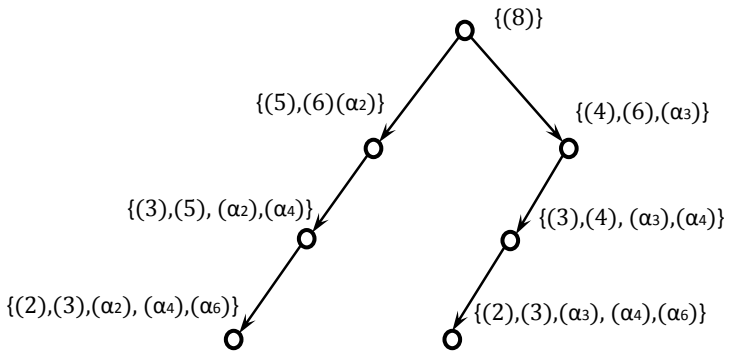

Figure 1: An example of EDG.

there always exists an EDG, such that, we can travel from a root note to any explanation through the edges.

Lemma 1. *There exists an EDG $\mathcal{G}_\alpha$ with respect to an axiom α, s.t., for any explanation $\mathcal{E}$ of α, there is a path starting from the node $V_{\{\alpha\}}$ and ending at $V_{\{\mathcal{E}\}}$.*

Proof. Suppose $\mathcal{E}$ is an explanation of α. This also implies that $\mathcal{E} \models \alpha$. In other words, α can be derived by applying the CEL rules in Table 2. One can identify a sequence of explanations $(\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_{k-1}, \mathcal{E}_k)$ $(k > 1)$ where $\mathcal{E}_1 = \mathcal{E}$ and $\mathcal{E}_k = \{\alpha\}$ (corresponding to the root node), such that for $1 \leq i \leq k-1$, $\mathcal{E}_{i+1}$ deductively depends on $\mathcal{E}_i$. Thus, we can find such an EDG containing the nodes $V_{\mathcal{E}_i}$ and $V_{\mathcal{E}_k}$ and edges $(V_{\mathcal{E}_i}, V_{\mathcal{E}_{i+1}})$ where $1 \leq i \leq k-1$. □

Backing to Step 1, we have that, given an EDG $\mathcal{G}$, we can obtain any explanation by traveling in $\mathcal{G}$ from the root node according to the above lemma. Further, if the target EDG is a complete EDG, we can also obtain all justifications by carefully checking each explanation (corresponding to Step 2). We introduce the detailed procedure in the next section.

5. Finding Justifications based on EDG

According to the discussion of the previous section, if a complete EDG can be constructed, then we can obtain all explanations (Step 1), where justifications can be further identified (Step 2). In this section, we specify Step 1 and Step 2 based on EDG and give algorithms of finding justifications.

Traveling Patterns. Recall that deductive dependencies are defined based on the rules in Table 2. This also means that deductive dependencies between explanations can be further distinguished by different rules in Table 2. Consider explanations (e_2) and (e_4) in Example 3. (e_2) deductively depends on (e_4) through Rule $\mathbf{R}_{\mathtt{H}}$. Based on the correspondences between deductive dependencies and classification rules, we can construct EDGs by traveling from the root node, and visiting other explanations through different rules. Specifically, we give different *traveling patterns* based on the rules in Table 2 as follows:

• Suppose $V_{\mathcal{E}}$ is a node in an EDG, and $A \sqsubseteq B$ is an entailment in $\mathcal{E}$. Since $A \sqsubseteq B$ can be derived by applying rules $\mathbf{R}_{\sqsubseteq}$, $\mathbf{R}_{\sqcap}$ or $\mathbf{R}_{\overset{-}{\exists}}$, we have the following patterns to reach a child node $V_{\mathcal{E}'}$ of $V_{\mathcal{E}}$:

($\mathbf{P}_0$) If $B \equiv A$, then $\mathcal{E}' = \mathcal{E} \setminus \{A \sqsubseteq B\}$;

($\mathbf{P}_\top$) If $B \equiv \top$, then $\mathcal{E}' = \mathcal{E} \setminus \{A \sqsubseteq B\}$;

($\mathbf{P}_{\sqsubseteq}$) If there exists a concept X s.t., $A \sqsubseteq X \in \mathcal{C}_{\mathcal{O}}$ and $X \sqsubseteq B \in \mathcal{O}$, then $\mathcal{E}' = \mathcal{E} \cup \{A \sqsubseteq X, X \sqsubseteq B\} \setminus \{A \sqsubseteq B\}$;

($\mathbf{P}_{\sqcap}$) If there exists axioms $A \sqsubseteq X_1, A \sqsubseteq X_1 \in \mathcal{C}_{\mathcal{O}}$ and $X_1 \sqcap X_2 \sqsubseteq B \in \mathcal{O}$, then $\mathcal{E}' = \mathcal{E} \cup \{A \sqsubseteq X_1, A \sqsubseteq X_1, X_1 \sqcap X_2 \sqsubseteq B\} \setminus \{A \sqsubseteq B\}$;

($\mathbf{P}_{\overset{-}{\exists}}$) If there exists axioms $A \sqsubseteq \exists r.X, X \sqsubseteq Y \in \mathcal{C}_{\mathcal{O}}$ and $\exists r.Y \sqsubseteq B \in \mathcal{O}$, then $\mathcal{E}' = \mathcal{E} \cup \{A \sqsubseteq \exists r.X, X \sqsubseteq Y, \exists r.Y \sqsubseteq B\} \setminus \{A \sqsubseteq B\}$.

In the above patterns, we say that $A \sqsubseteq B$ is a *bud* in $\mathcal{E}$.

• Suppose $V_{\mathcal{E}}$ is a node in an EDG, and $A \sqsubseteq \exists r.B$ is an entailment in $\mathcal{E}$. Since $A \sqsubseteq \exists r.B$ can derived by applying the rules $\mathbf{R}_{\overset{+}{\exists}}$, $\mathbf{R}_{\mathtt{H}}$ or $\mathbf{R}_{\circ}$, we have the following three patterns to reach a child node $V_{\mathcal{E}'}$ of $V_{\mathcal{E}}$:

($\mathbf{P}_{\overset{+}{\exists}}$) If there exists a concept X s.t., $A \sqsubseteq X \in \mathcal{C}_{\mathcal{O}}$ and $X \sqsubseteq \exists r.B \in \mathcal{O}$, then $\mathcal{E}' = \mathcal{E} \cup \{A \sqsubseteq X, X \sqsubseteq \exists r.B\} \setminus \{A \sqsubseteq \exists r.B\}$;

($\mathbf{P}_{\mathtt{H}}$) If there exists a role s s.t., $A \sqsubseteq \exists s.B \in \mathcal{C}_{\mathcal{O}}$ and $s \sqsubseteq r \in \mathcal{O}$, then $\mathcal{E}' = \mathcal{E} \cup \{A \sqsubseteq \exists s.B, s \sqsubseteq r\} \setminus \{A \sqsubseteq \exists r.B\}$;

($\mathbf{P}_{\circ}$) If there exists axioms $A \sqsubseteq \exists s.X, X \sqsubseteq \exists t.B \in \mathcal{C}_{\mathcal{O}}$ and $s \circ t \sqsubseteq r \in \mathcal{O}$, then $\mathcal{E}' = \mathcal{E} \cup \{A \sqsubseteq \exists s.X, X \sqsubseteq \exists t.B, s \circ t \sqsubseteq r\} \setminus \{A \sqsubseteq \exists r.B\}$.

In the above patterns, we say that $A \sqsubseteq \exists r.B$ is a *bud* in $\mathcal{E}$. $\square$

Patterns $\mathbf{P}_0$-$\mathbf{P}_\circ$ are applied when traveling from a node $\mathcal{E}$ to one of its child $\mathcal{E}'$ in an EDG through some bud. Patterns $\mathbf{P}_0$ and $\mathbf{P}_\top$ say that for an entailment of the form $A \sqsubseteq A$ or $A \sqsubseteq \top$ occurring in $\mathcal{E}$, this entailment is directly eliminated from $\mathcal{E}$, and, $\mathcal{E}'$ is obtained. This is because that entailments of the form $A \sqsubseteq A$ or $A \sqsubseteq \top$ are tautological axioms which hold in any ontology containing the concept A. Patterns $\mathbf{P}_\sqsubseteq$ to $\mathbf{P}_\circ$ correspond to Rules $\mathbf{R}_\sqsubseteq$ to $\mathbf{R}_\circ$ in Table 2 respectively. Intuitively, these patterns can be viewed as inverse processes of corresponding rules. We take Pattern $\mathbf{P}_{\overline{\exists}}$ for an example. Suppose $\mathcal{E}$ contains an entailment of the form $A \sqsubseteq \exists r.B$, and we also have that two axioms $A \sqsubseteq \exists r.X$ and $X \sqsubseteq Y$ hold in the set of classification results $\mathcal{C}_\mathcal{O}$ and $\exists r.Y \sqsubseteq B$ occurs in the original ontology. It also means that $A \sqsubseteq \exists r.B$ can be derived from $A \sqsubseteq \exists r.X$, $X \sqsubseteq Y$ and $\exists r.Y \sqsubseteq B$ by applying Rule $\mathbf{R}_{\overline{\exists}}$. We get a child node of $\mathcal{E}$: $\mathcal{E}'(= \mathcal{E} \cup \{A \sqsubseteq \exists r.X, X \sqsubseteq Y, \exists r.Y \sqsubseteq B\} \setminus \{A \sqsubseteq \exists r.B\})$ by treating $A \sqsubseteq \exists r.B$ as a bud. Based on the above discussion, if $\mathcal{E}$ is an explanation of the target axiom α, $\mathcal{E}'$ is also an explanation of α. The other patterns can be explained similarly.

An Algorithm of Constructing EDGs. Based on the traveling patterns, we can construct a complete EDG and obtain all explanations for a given entailment α. An intuitive process of constructing EDGs is as follows: the process starts by adding a root node in an empty EDG and iteratively generates child nodes of newly added nodes; since one node in an EDG may contain multiple entailments which can all be selected as buds, there could be multiple child nodes by selecting different buds; when all nodes are generated, the construction of a complete EDG is finished. We give Algorithm 1 to construct a complete EDG.

At the beginning of Algorithm 1, the algorithm checks whether the axiom $A \sqsubseteq B$ is in the original ontology $\mathcal{O}$. If $A \sqsubseteq B$ is an original axiom, Algorithm 1 terminates and returns $\{A \sqsubseteq B\}$ as the only explanation (lines 1-3). Otherwise an empty queue Q is created and the root node $V_{\{A \sqsubseteq B\}}$ is put in Q (line 4 and 5). Then the algorithm uses the queue Q to construct an EDG and obtain the set of explanations of $\{A \sqsubseteq B\}$ (line 6 - 14). The basic procedure is that, if the queue Q is not empty, a new node $V_\mathcal{E}$ in Q is visited and the function `expand(·)` is called to expand $V_\mathcal{E}$ and generate new child nodes of $V_\mathcal{E}$ based on the traveling patterns $\mathbf{P}_0$-$\mathbf{P}_\circ$. In line 8, a condition BC1 (we discuss it in detail in following paragraphs) is used to guarantee the

termination of Algorithm 1. The program may not terminate in some cases, where an entailment in one node $V_{\mathcal{E}}$ is also in an child node $V_{\mathcal{E}'}$ of $V_{\mathcal{E}}$. We call this a *non-termination problem*. We use an example to illustrate this problem.

Algorithm 1: Constructing a Complete EDG

Input:
$A \sqsubseteq B$: an entailment;
$\mathcal{O}$: a normalized $\mathcal{EL}^+$ ontology;
$\mathcal{C}_{\mathcal{O}}$: the classification result after classifying $\mathcal{O}$.
Output:
J: the set of explanations for $A \sqsubseteq B$.

```
1  if A ⊑ B ∈ O then
2      put(J, {A ⊑ B});
3      return;
4  create a queue Q;
5  put a root node V_{A⊑B} in Q;
6  while Q is not empty do
7      V_E ← Q.pop();
8      if V_E satisfies BC1 then
9          skip following procedure and continue the while iteration;
10     if E^ea is empty then
11         put(J, E);
12     else
13         get an entailment α from E^ea;
14         expand(A ⊑ B, V_E, Q);
```

Example 5. *Given a node $V_{\mathcal{E}}$ in an EDG for some entailment $X \sqsubseteq B$ (see Figure 2), whose corresponding explanation $\mathcal{E}$ is $\{X \sqsubseteq A_1, X \sqsubseteq A_2, A_1 \sqcap A_2 \sqsubseteq B\}$. The axiom $X \sqsubseteq A_1$ is selected as a bud to expand the current EDG, and there exist two entailments $X \sqsubseteq B$ and $B \sqsubseteq A_1$ in the set of classification results, a new node $V_{\mathcal{E}'}$ is generated according to Pattern $\boldsymbol{P}_{\sqsubseteq}$ and $\mathcal{E}' = \{X \sqsubseteq B, B \sqsubseteq A_1, X \sqsubseteq A_2, A_1 \sqcap A_2 \sqsubseteq B\}$. In the next step, the program further generates a child node $V_{\mathcal{E}''}$ of $V_{\mathcal{E}'}$ where $\mathcal{E}'' = \{X \sqsubseteq A_1, B \sqsubseteq A_1, X \sqsubseteq A_2, A_1 \sqcap A_2 \sqsubseteq B\}$ by applying Pattern $\boldsymbol{P}_{\sqcap}$. Here $X \sqsubseteq A_1$ is re-added in $\mathcal{E}''$. In addition, $V_{\mathcal{E}}$ is an ancestor of $V_{\mathcal{E}''}$. Thus, it is obvious that $X \sqsubseteq A_1$ will be repeatedly added in one of the descendant nodes of $V_{\mathcal{E}''}$. This then*

leads to a non-terminated process.

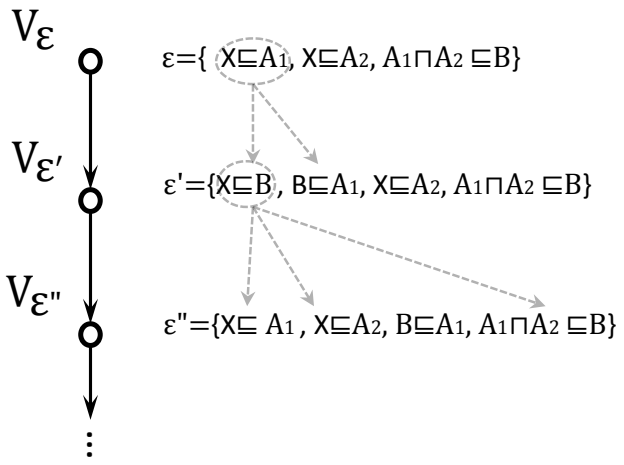

Figure 2: A case where the program will not terminate

To avoid the non-termination problem, we follow an intuitive idea: if a node $V_{\mathcal{E}}$ is generated and there exists an entailment α in $\mathcal{E}$ that also appears in one ancestor node of $V_{\mathcal{E}}$, $\mathcal{E}$ is blocked for further expanding. As shown in Figure 2, if the node $V_{\mathcal{E}''}$ is blocked, none of its child node can be further generated. Formally, we give the following *blocking condition* BC1:

BC1 Suppose a node $V_{\mathcal{E}}$ is being visited, if there exists an entailment α in $\mathcal{E}$ which is also a bud in one ancestor node $V_{\mathcal{E}'}$ of $V_{\mathcal{E}}$, then $V_{\mathcal{E}}$ is blocked.

Intuitively, by applying BC1, in each path of an EDG constructed by Algorithm 1, all nodes have different buds. Since the number of entailments with respect to an ontology is fixed in the size of the given ontology, EDGs constructed by Algorithm 1 have to be finite in size.

At this point, Step 1 has been finished, i.e., we can use Algorithm 1 to obtain all explanations for an entailment. We give the following theorem to show the correctness of Algorithm 1,. That is: (1) BC1 guarantees the termination of Algorithm 1; (2) Algorithm 1 can always construct a complete EDG.

Theorem 1. *Given a normalized $\mathcal{EL}^+$ ontology $\mathcal{O}$ and the set of classification results $\mathcal{C}_{\mathcal{O}}$, $A \sqsubseteq B$ is an entailment of $\mathcal{O}$, we have that Algorithm 1 terminates with $A \sqsubseteq B$ as the input, and constructs a complete EDG of $A \sqsubseteq B$.*

The above theorem can be proved based on Lemma 2 and Lemma 3.

Lemma 2. *Given a normalized $\mathcal{EL}^+$ ontology $\mathcal{O}$, α is an entailment of $\mathcal{O}$ and $\mathcal{J}$ is a justification with respect to $\mathcal{O}$ and α. $\mathcal{G}$ is an EDG for α with respect to $\mathcal{J}$, which is constructed by Algorithm 1. We have that $V_{\mathcal{J}}$ is a node in $\mathcal{G}$.*

Proof. Since $\mathcal{J}$ is a justification of α, one can identify a sequence of explanations $(\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_{k-1}, \mathcal{E}_k)$ $(k > 1)$ where $\mathcal{E}_1 = \mathcal{J}$ and $\mathcal{E}_k = \{\alpha\}$, such that for $1 \leq i \leq k-1$, $\mathcal{E}_{i+1}$ deductively depends on $\mathcal{E}_i$, and these explanations satisfy BC1. It can be easily checked that each node of the form $V_{\mathcal{E}_i}$ $(1 \leq i \leq k)$ has to be added to $\mathcal{G}$ by Algorithm 1. □

Lemma 3. *Given two normalized $\mathcal{EL}^+$ ontologies $\mathcal{O}$ and $\mathcal{O}'$, α is an entailment of both $\mathcal{O}$ and $\mathcal{O}'$. $\mathcal{G} = (V, E)$ (resp. $\mathcal{G}' = (V', E')$) is an EDG for α with respect to $\mathcal{O}$ (resp. $\mathcal{O}'$), which is constructed by Algorithm 1. If $\mathcal{O}' \subseteq \mathcal{O}$, we have that $\mathcal{G}'$ is a subgraph of $\mathcal{G}$, specifically, $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{E}' \subseteq \mathcal{E}$.*

Proof. The above lemma can be proved based on the conclusion that $\mathcal{C}_{\mathcal{O}'} \subseteq \mathcal{C}_{\mathcal{O}}$ where $\mathcal{C}_{\mathcal{O}'}$ (resp., $\mathcal{C}_{\mathcal{O}}$) is the set of classification results of $\mathcal{O}'$ (resp., $\mathcal{O}$). This conclusion can be proved by an induction on the CEL rules in Table 2. It can also be proved by the semantics of $\mathcal{EL}^+$. That is, for each entailment β, $\mathcal{O}' \models \beta$ implies $\mathcal{O} \models \beta$. We do not give the details here.

To finish this lemma, we first define the depths of nodes in an EDG $\mathcal{H}$ as follows. Let the depth of the root node V_{root} be 0, denoted by $\mathtt{d}(V_{root}) = 0$; for each node V and its parent V' in $\mathcal{H}$, if $\mathtt{d}(V') = k$, let $\mathtt{d}(V) = k + 1$. We conduct the proof of this lemma by an induction on the depths of nodes in the EDG $\mathcal{H}$ constructed by Algorithm 1.

(*Basic case*) Suppose α is the target axiom. The root node V_α has to exist in both of $\mathcal{G}$ and $\mathcal{G}'$. Thus, the basic case holds.

(*Inductive cases*) We have an induction hypothesis that, for an depth i $(i \geq 0)$, each node V of depth j $(1 \leq j \leq i)$ in $\mathcal{G}'$ is also in $\mathcal{G}$. We then need to prove that, for the depth $i + 1$, each node V of depth j' $(1 \leq j' \leq i + 1)$ in $\mathcal{G}'$ has to be in $\mathcal{G}$. In Algorithm 1, new nodes are generated by just applying the code in line 14 where the function $\mathtt{expand}(\cdot)$ is called. In the function $\mathtt{expand}(\cdot)$, the traveling patterns $\mathbf{P}_0$-$\mathbf{P}_\circ$ are applied respectively. In the following, we continue this proof by distinguishing different cases based

on different traveling patterns.

Suppose V is in $\mathcal{G}'$, V' is the parent of V, and $\mathtt{d}(V) = j$ $(1 \leq j \leq i+1)$. We should prove that V is in $\mathcal{G}$ as well.

(*Case* $\mathbf{P}_0$) Suppose V is generated by using this pattern. We have that V' is in the form of $V \cup \{A \sqsubseteq B\}$ where $A \equiv B$. According to the induction hypothesis, V' is in $\mathcal{G}$. By applying this pattern in Algorithm 1, V can also be generated and added in $\mathcal{G}$.

(*Case* $\mathbf{P}_\top$) This case is similar to the previous case.

(*Case* $\mathbf{P}_\sqsubseteq$) Suppose V is generated by using this pattern. We have that V' is in the form of $V \cup \{A \sqsubseteq B\} \setminus \{A \sqsubseteq X, X \sqsubseteq B\}$ where $A \sqsubseteq X \in \mathcal{C}_{\mathcal{O}'}$ and $X \sqsubseteq B \in \mathcal{O}'$. Since we have that $\mathcal{C}_{\mathcal{O}'} \subseteq \mathcal{C}_{\mathcal{O}}$ and $\mathcal{O}' \subseteq \mathcal{O}$, $A \sqsubseteq X \in \mathcal{C}_{\mathcal{O}}$ and $X \sqsubseteq B \in \mathcal{O}$ also hold. According to the induction hypothesis, V' is in $\mathcal{G}$. By applying this pattern on $\mathcal{O}$ in Algorithm 1, V can also be generated and added to $\mathcal{G}$.

(*Case* $\mathbf{P}_\sqcap$) Suppose V is generated by using this pattern. We have that V' is in the form of $V \cup \{A \sqsubseteq B\} \setminus \{A \sqsubseteq X_1, A \sqsubseteq X_1, X_1 \sqcap X_2 \sqsubseteq B\}$ where $A \sqsubseteq X_1, A \sqsubseteq X_1 \in \mathcal{C}_{\mathcal{O}'}$ and $X_1 \sqcap X_2 \sqsubseteq B \in \mathcal{O}'$. Since we have that $\mathcal{C}_{\mathcal{O}'} \subseteq \mathcal{C}_{\mathcal{O}}$ and $\mathcal{O}' \subseteq \mathcal{O}$, $A \sqsubseteq X_1, A \sqsubseteq X_1 \in \mathcal{C}_{\mathcal{O}}$ and $X_1 \sqcap X_2 \sqsubseteq B \in \mathcal{O}$ also hold. According to the induction hypothesis, V' is in $\mathcal{G}$. By applying this pattern on $\mathcal{O}$ in Algorithm 1, V can also be generated and added to $\mathcal{G}$.

(*Case* $\mathbf{P}^-_\exists$) Suppose V is generated by using this pattern. We have that V' is in the form of $V \cup \{A \sqsubseteq B\} \setminus \{A \sqsubseteq \exists r.X, X \sqsubseteq Y, \exists r.Y \sqsubseteq B\}$ where $A \sqsubseteq \exists r.X, X \sqsubseteq Y \in \mathcal{C}_{\mathcal{O}'}$ and $\exists r.Y \sqsubseteq B \in \mathcal{O}'$. Since we have that $\mathcal{C}_{\mathcal{O}'} \subseteq \mathcal{C}_{\mathcal{O}}$ and $\mathcal{O}' \subseteq \mathcal{O}$, $A \sqsubseteq \exists r.X, X \sqsubseteq Y \in \mathcal{C}_{\mathcal{O}}$ and $\exists r.Y \sqsubseteq B \in \mathcal{O}$ also hold. According to the induction hypothesis, V' is in $\mathcal{G}$. By applying this pattern on $\mathcal{O}$ in Algorithm 1, V can also be generated and added to $\mathcal{G}$.

(*Case* $\mathbf{P}^+_\exists$) Suppose V is generated by using this pattern. We have that V' is in the form of $V \cup \{A \sqsubseteq \exists r.B\} \setminus \{A \sqsubseteq X, X \sqsubseteq \exists r.B\}$ where $A \sqsubseteq X \in \mathcal{C}_{\mathcal{O}'}$ and $X \sqsubseteq \exists r.B \in \mathcal{O}'$. Since we have that $\mathcal{C}_{\mathcal{O}'} \subseteq \mathcal{C}_{\mathcal{O}}$ and $\mathcal{O}' \subseteq \mathcal{O}$, $A \sqsubseteq X \in \mathcal{C}_{\mathcal{O}}$ and $X \sqsubseteq \exists r.B \in \mathcal{O}$ also hold. According to the induction hypothesis, V' is

in $\mathcal{G}$. By applying this pattern on $\mathcal{O}$ in Algorithm 1, V can also be generated and added to $\mathcal{G}$.

(*Case* $\mathbf{P}_{\mathtt{H}}$) Suppose V is generated by using this pattern. We have that V' is in the form of $V \cup \{A \sqsubseteq \exists r.B\} \setminus \{A \sqsubseteq \exists s.B, s \sqsubseteq r\}$ where $A \sqsubseteq \exists s.B \in \mathcal{C}_{\mathcal{O}'}$ and $s \sqsubseteq r \in \mathcal{O}'$. Since we have that $\mathcal{C}_{\mathcal{O}'} \subseteq \mathcal{C}_{\mathcal{O}}$ and $\mathcal{O}' \subseteq \mathcal{O}$, $A \sqsubseteq \exists s.B \in \mathcal{C}_{\mathcal{O}}$ and $s \sqsubseteq r \in \mathcal{O}$ also hold. According to the induction hypothesis, V' is in $\mathcal{G}$. By applying this pattern on $\mathcal{O}$ in Algorithm 1, V can also be generated and added to $\mathcal{G}$.

(*Case* $\mathbf{P}_{\circ}$) Suppose V is generated by using this pattern. We have that V' is in the form of $V \cup \{A \sqsubseteq \exists r.B\} \setminus \{A \sqsubseteq \exists s.X, X \sqsubseteq \exists t.B, s \circ t \sqsubseteq r\}$ where $A \sqsubseteq \exists s.X, X \sqsubseteq \exists t.B \in \mathcal{C}_{\mathcal{O}'}$ and $s \circ t \sqsubseteq r \in \mathcal{O}'$. Since we have that $\mathcal{C}_{\mathcal{O}'} \subseteq \mathcal{C}_{\mathcal{O}}$ and $\mathcal{O}' \subseteq \mathcal{O}$, $A \sqsubseteq \exists s.X, X \sqsubseteq \exists t.B \in \mathcal{C}_{\mathcal{O}}$ and $s \circ t \sqsubseteq r \in \mathcal{O}$ also hold. According to the induction hypothesis, V' is in $\mathcal{G}$. By applying this pattern on $\mathcal{O}$ in Algorithm 1, V can also be generated and added to $\mathcal{G}$.

The proof of Lemma 3 is finished. We now prove Theorem 1. For each justification $\mathcal{J}$ of the entailment $A \sqsubseteq B$, we have that $\mathcal{J} \subseteq \mathcal{O}$. Let $\mathcal{G}_{\mathcal{J}}$ be the EDG constructed by Algorithm 1 on $\mathcal{J}$ and $A \sqsubseteq B$. There has to be a node $V_{\mathcal{J}}$ in $\mathcal{G}_{\mathcal{J}}$ according to Lemma 2. By Lemma 3, $\mathcal{G}_{\mathcal{J}} \subseteq \mathcal{G}_{\mathcal{O}}$ holds. Thus, we have that $V_{\mathcal{J}}$ is in $\mathcal{G}_{\mathcal{O}}$. It also indicates that $\mathcal{G}_{\mathcal{O}}$ is a complete EDG. □

We now discuss the scale of a complete EDG. In the worst cases, a complete EDG may contain exponential number of nodes in the size of the target ontology. Specifically, in the *while* iteration, Algorithm 1 constructs the target EDG by starting from a node $V_{\mathcal{E}}$ through the out-edges to all possible child nodes for one bud in $\mathcal{E}$. Suppose $\mathcal{E}'$ is a newly added explanation. If all axioms in $\mathcal{E}'$ are from the original ontology, $\mathcal{E}'$ will be added as a candidate justification in J (see lines 10 and 11). When queue Q is empty, the construction stops. Consider that the worst case is to visit all possible nodes and links between adjacent nodes in the target EDG, the complexity is $O(|V|^2)$. It shows that the complexity of our algorithm depends on the number of nodes in a complete EDG. Since there exists exponential number of justifications in the worst case, there may be exponentially many nodes in an EDG. This also shows the hardness of the problem of finding justifications.

Computing Justifications from Explanations. Algorithm 1 returns explanations that may be supersets of justifications. In order to obtain all

justifications, we should further process the outputs of Algorithm 1 (corresponding to Step 2). We use Algorithm 2 to compute all justifications which takes the outputs of Algorithm 1 as inputs.

Algorithm 2: Computing All Justifications

Input:
J: a set of explanations for $A \sqsubseteq B$ (containing only original axioms).
Output:
J': the set of all justifications for $A \sqsubseteq B$.

```
1 for j_n ∈ J do
2     for j_m ∈ J do
3         if j_n ⊂ j_m then
4             Remove(J, j_m);
5         else
6             Remove(J, j_n);
7 J' ← J;
```

Algorithm 2 computes all justifications for an entailment $A \sqsubseteq B$ by removing all the supersets of each candidate justification in J. The cost of computing in this phase can be negligible compared to the cost of constructing an EDG. This is because that the number of explanations containing only original axioms is far less than the number of nodes generated in an EDG according to our preliminary experiments.

6. An Optimized Algorithm

In this section, we consider further optimizing Algorithm 1 in two ways. First, we propose other blocking conditions to prune a complete EDG constructed by Algorithm 1, such that not all nodes in this graph have to be checked. On the other hand, we apply parallel techniques to improve the efficiency of EDG construction. Based on these two optimizations, we give a new algorithm of constructing EDGs. Furthermore, we ensure that the optimized algorithm can always construct complete EDGs.

Blocking Conditions. In the work of [7, 8], the authors propose a method of finding justifications based on a kind of tree structure (known as *Hitting Set Tree*). A simple blocking condition is studied in this work to avoid visiting all branches of a hitting set tree when generating justifications. The basic idea is to use the visited branches to filter out the useless branches.

We follow this idea and study the blocking conditions in our case. That is, to give conditions to block the explanations that will not generate justifications during the construction of an EDG. Since an EDG is not a hitting set tree, the blocking condition given in [7, 8] cannot apply in our case. We give our own blocking conditions as follows.

Suppose a node $V_{\mathcal{E}}$ is being visited:

BC2 If there exists a node $V_{\mathcal{E}'}$ which has been visited such that $\mathcal{E}' \subseteq \mathcal{E}$, then $V_{\mathcal{E}}$ is blocked;

BC3 If there exists a candidate justification j such that $j \subseteq \mathcal{E}^{\mathtt{oa}}$, then $V_{\mathcal{E}}$ is blocked;

BC4 If there exists a child node $V_{\mathcal{E}'}$ expanded from an entailment α in $V_{\mathcal{E}}$, and $\mathcal{E}' \subseteq \mathcal{E}$, then the other branches of $V_{\mathcal{E}}$ expanded by α are blocked.

The condition BC2 is used to avoid repeated expansion of the same node. BC3 states that when a node $V_{\mathcal{E}}$ is visited and its original axiom set has already been a candidate justification, then it is not needed to be expanded further. The meaning of BC4 can be explained similarly. We use Example 6 to illustrate the usage of BC4.

Example 6. *In Figure 3, $V_{\mathcal{E}}$ is expanded to $V_{\mathcal{E}'}$, $V_{\mathcal{E}''}$ and $V_{\mathcal{E}'''}$ from $X \sqsubseteq B$ by using Patterns $\boldsymbol{P}_{\sqcap}$ or $\boldsymbol{P}_{\sqsubseteq}$. After expanding, we have that $\mathcal{E}'' \subseteq \mathcal{E}$. This means that $X \sqsubseteq B$ holds in $\mathcal{E}$. We also have $\mathcal{E}'' \subseteq \mathcal{E}'$ and $\mathcal{E}'' \subseteq \mathcal{E}'''$. According to BC2, $V_{\mathcal{E}'}$ and $V_{\mathcal{E}'''}$ can be blocked. Thus in this kind of cases, we block the other branches expanded from such entailments since the nodes in those branches will not generate justifications.*

Parallelizing EDG construction. Parallelizing the task of finding justifications is rarely studied due to its inherent high complexity. One parallel method of ontology debugging is given in the work of [37]. This method focus on partitioning the target ontology into smaller pieces and conducting ontology debugging in parallel. Since this method aims at parallelizing the target ontology but not the algorithm, it cannot be used in our case. For parallelizing EDG construction, we give our method based on the observation that the paths in an EDG are independent from each other. This indicates that the generation of each path in an EDG can be performed in a parallel

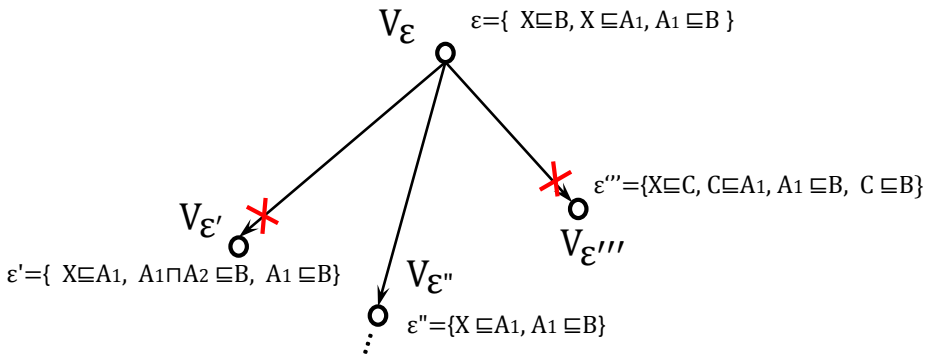

Figure 3: A case of using BC4 to block nodes in an EDG.

way. This is the basic motivation of our method. Specifically, we give Algorithm 3 which parallelizes EDG construction and utilizes the above blocking conditions as well.

The initialization part of Algorithm 3 is the same as that of Algorithm 1. The difference is that we parallelize the *while* iteration in Algorithm 3. In line 9, when a node $V_{\mathcal{E}}$ is generated from Q, the program launches a new routine (or a thread) R_1 to expand the EDG from $V_{\mathcal{E}}$. The execution logic of R_1 is given in lines 10 to 19. R_1 first checks whether $V_{\mathcal{E}}$ can be blocked based on the blocking conditions (line 11). If $V_{\mathcal{E}}$ can be blocked, R_1 terminates and the traversal in this branch stops. Otherwise, R_1 expands $V_{\mathcal{E}}$ using the traveling patterns and a child node of $V_{\mathcal{E}}$ denoted by $V_{\mathcal{E}'}$ is generated. If $\mathcal{E}'$ consists of original axioms, $\mathcal{E}'$ will be added to J (line 16); otherwise it will be put in Q (line 18). The program computes all justifications using Algorithm 2 in line 20. We give Theorem 2 to show the correctness of Algorithm 3.

Theorem 2. *Given a normalized $\mathcal{EL}^+$ ontology $\mathcal{O}$ and the classification result $\mathcal{C}_{\mathcal{O}}$, suppose $A \sqsubseteq B$ is an entailment of $\mathcal{O}$, Algorithm 3 terminates with $A \sqsubseteq B$ as the input, and constructs a complete EDG graph of $A \sqsubseteq B$.*

Proof. We first construct a new algorithm $\mathcal{A}_1^{\mathtt{opt}}$ based on Algorithm 1. Specifically, $\mathcal{A}_1^{\mathtt{opt}}$ generates new nodes as same as Algorithm 1, but checks these nodes according to BC2-BC4, i.e., if new generated nodes satisfy BC2, BC3 or BC4, $\mathcal{A}_1^{\mathtt{opt}}$ blocks them. It is obvious that $\mathcal{A}_1^{\mathtt{opt}}$ constructs complete EDGs,

since blocked nodes must not lead to justifications.

Algorithm 3: Computing Justifications in Parallel

Input:
$A \sqsubseteq B$: an entailment;
$\mathcal{O}$: a normalized $\mathcal{EL}^+$ ontology;
$\mathcal{C}_{\mathcal{O}}$: the set of classification result with respect to $\mathcal{O}$.
Output: J: the set of explanations for $A \sqsubseteq B$.

```
1  if A ⊑ B ∈ O then
2      put(J, {A ⊑ B});
3      return;
4  create a queue Q;
5  put a root node V_{A⊑B} in Q;
6  while Q is not empty do
7      V_E ← Q.pop();
8      allocate V_E to a new routine R_1;
9      R_1.execute():
10     {
11     if V_E can be blocked then
12         R_1.stop();
13     else
14         V_E' ← expand(V_E);
15         if E' contains no entailment then
16             put(J, E');
17         else
18             put(Q, V_E');
19     }
20     J ← computeAllJustifications(J);
```

We now prove the above theorem by checking that all nodes in the EDG $\mathcal{H}$ constructed by $\mathcal{A}_1^{\mathtt{opt}}$ have to be generated in the EDG $\mathcal{G}$ constructed by Algorithm 3 as well. We show this by an induction on the depths of nodes (see the definition of node depths in the proof of Lemma 3) in $\mathcal{H}$.

(*Basic case*) Suppose $\alpha \equiv A \sqsubseteq B$ is the target axiom. The root node V_α has to exist in both of $\mathcal{H}$ and $\mathcal{G}$. Thus, the basic case holds.

(*Inductive cases*) We have an induction hypothesis that, for an depth i ($i \geq 0$), each node V of depth j ($1 \leq j \leq i$) in $\mathcal{H}$ is also in $\mathcal{G}$. We then need to prove that, for the depth $i+1$, each node V of depth j' ($1 \leq j' \leq i+1$) in $\mathcal{H}$ has to be in $\mathcal{G}$. This can be proved by two *invariants* set in $\mathcal{A}_1^{\mathtt{opt}}$ and Algorithm 3: (1) each explanation has a total lexicographical order of axioms; (2) each selected bud has the minimum order in an explanation. These two invariants ensure that whenever a node is checked in Algorithm 3, it would generate same child nodes. Based on this observation and the induction hypothesis, the inductive cases can be proved. □

7. Evaluation and Analysis

In this section, we evaluate our method by conducting experiments on real ontologies. We further compare our method to glass-box and black-box implementations. Finally, we give analysis for the experimental results.

Implementation and Datasets. We implemented our algorithms by using Java multithread techniques. Since the queue in Algorithm 1 would be visited concurrently, we implemented it based on a thread-safe datatype `ConcurrentLinkedQueue`. To classify an ontology, we used **jCEL**[3] as the $\mathcal{EL}^+$ reasoner. Our running environment is an IBM X3850 server with a memory of 16 GiB and 8 cores. An auxiliary process is set to transform the set of classification results to appropriate data structures for our system.

We used five real ontologies to perform our experiments: Galen-core (a tailored version of Galen), Galen, NCI, GO and SNOMED CT.[4] These ontologies contain no noise and are usually used for evaluating the methods of ontology debugging [19, 7, 10, 9]. The source code of our system and the above ontologies can be found at the address.[5] Our method requires a single run of the reasoner and utilizes the set of classification results to construct EDGs. The number of axioms, the cost time of classification and transformation are given in Table 3. The classification and transformation need to be done only once for each ontology. Thus, we can set an offline phase to finish classification and transformation.

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

[4]SNOMED CT can be obtained from http://www.ihtsdo.org/snomed-ct.

[5]https://github.com/quanzz/findJusts/

Table 3: The classification and transformation time in seconds

ontology	#axioms	classification time	transformation time
Galen-core	7,540	2.482	0.188
NCI	422,953	2.466	1.031
GO	89,370	2.622	0.469
Galen	60,633	60.654	1.904
SNOMED CT	1,046,114	1133.575	10.765

Evaluation. For Galen-core, NCI and GO, we sampled 20% of all the entailments based on the method of simple random sampling [38]. For Galen and SNMOED CT, we sampled 200 entailments. The above samples do not include the entailments of the form $A \sqsubseteq A$, $A \sqsubseteq \top$ or $\bot \sqsubseteq A$ since they make no sense in the final experimental results. In addition, we set 10 minutes as the time limit for all experiments. One may note that reasoning time is unlimited. This is because that the task of reasoning is supposed to run in an offline way. However, the task of finding justifications is supposed to be a service online. Thus, we set 10 minutes to bond the response time.

Table 4: The experimental results of our method

ontology	avg. time/s	avg. #nodes	avg. space (KiB)	time in the worst case/s	#nodes in the worst case	max. space (KiB)	%time -out cases
Galen-core	0.76	20.39	2.79	78.13	644	87.51	0
NCI	2.34	10.96	1.49	191.42	308	42.10	0
GO	0.52	10.84	1.44	126.62	260	35.54	0
Galen	7.74	122.42	16.73	574.55	7301	998.18	4.5%
SNOMED CT	0.44	26.37	3.61	356.12	351,314	48031.21	33.2%

We ran our system on the sampled entailments, and collected the experimental results in Table 4. The fourth (resp., seventh) column of Table 4 collected the averagely (resp., maximal) occupied space for storage of a constructed EDG (a kilobyte per unit). From Table 4, we can see that there is no time-out case for Galen-core, NCI and GO due to their small sizes. By comparing the results of different ontologies, we can find that, the average computing time is close. This is similar to the results of the average numbers of expanded nodes. This is because the main proportion of samples tends

to be easily handled (we call them *the easy samples*), i.e., the EDGs of the easy samples contain a handful of nodes, and, can be constructed within a second. On the other hand, the results of the average cases are significantly better than that of the worst cases. This can also be explained by the large proportion for the easy samples.

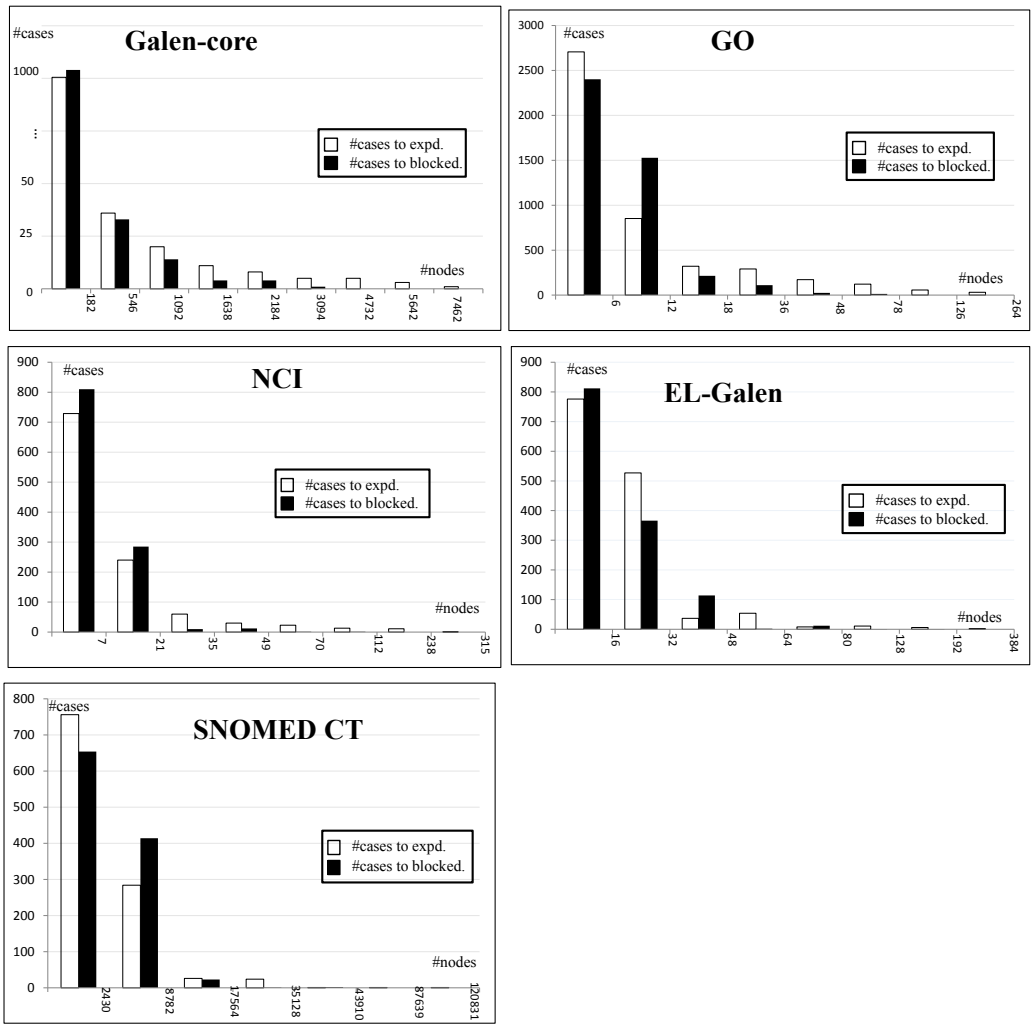

Figure 4: The statistics of constructing EDGs.

In order to investigate the effect of the blocking conditions given in Section 6, we give Figure 4 where five histograms are involved for the tested

ontologies respectively. In each histogram, the abscissa axis records the numbers of nodes in an EDG, the vertical axis records the numbers of samples, and, the white bars (resp., the black bars) denote the numbers of samples for which the numbers of finally expanded nodes (resp., the numbers of blocked nodes) fall in the corresponding intervals of the abscissa axis. From Figure 4, we can find that each histogram has relatively higher bars in its head and shorter bars in its tail. This indicates that the easy samples take a large proportion, i.e., the numbers of finally expanded nodes are small. On the other hand, the areas of black bars are close to that of white bars. This means that, the number of blocked nodes is close to the number of finally expanded nodes. According to Algorithm 1, visiting a node occupies a time unit. Thus, the computing time is actually reduced by nearly 50% compared to a naive algorithm without using the blocking conditions.

Comparison. In this part, we compare our method to traditional glass-box and black-box methods by evaluating the same samples. Since there is no available implementation of traditional glass-box method for finding justification in OWL EL, we follow the idea given in [6] and implement a system (denoted by *Sys-Glass-Box*) using Java. For black-box implementation, we choose a well-developed Protégé plugin, *the Explanation Workbench*, which can also handle OWL EL ontologies [39].

We use the sampled entailments in the previous experiment to evaluate Sys-Glass-Box and the Explanation Workbench and collected the results in Table 5 and Table 6 respectively. In Table 5, the third (resp., fifth) column collects the averagely (resp., maximal) occupied space for storage of tracing information (a kilobyte per unit). In Table 6, the third (resp., fifth) column collects the average (resp., maximal) number of reasoner calls.

Table 5: The experimental results of Sys-Glass-Box

ontology	avg. time/s	avg. #space (KiB)	worst case time/s	max. space (KiB)	%time-out cases
Galen-core	1.31	476.56	91.27	2382.87	3.4%
NCI	2.88	26731.95	423.01	80195.88	2.7%
GO	1.75	5648.46	338.64	28247.31	2.3%
Galen	13.10	2368.47	547.81	49818.92	10.4%
SNOMED CT	2.21	40863.83	345.07	528941.76	46.1%

Table 6: The experimental results of the Explanation Workbench

ontology	avg. time/s	avg. #calls	worst case time/s	#max. calls	%time-out cases
Galen-core	0.11	56	7.01	728	2.3%
NCI	1.92	1022	199.51	98964	1.1%
GO	0.56	576	289.10	16207	5%
Galen	5.20	189	425.23	896	16.7%
SNOMED CT	0.73	474	219.08	10126	41.2%

By comparing the results in Table 4 and Table 5, we can find that, the occupied space of our method is far less than that of Sys-Glass-Box. This is due to two reasons: 1) our method does not attach extra information to the classification results, while Sys-Glass-Box has to tag each entailment to trace reasoning procedure; 2) an EDG is only constructed for a target entailment. In other words, the unrelated axioms and entailments will not be considered during computing. On the other hand, some optimizing strategies, e.g., pruning techniques, cannot be used in Sys-Glass-Box. This is because that the tracing paths used in [6] are inter-dependent. This kind of inter-dependence makes it impossible to use block conditions and introduce parallelism techniques. The experimental results also show that our method performs better in terms of computing time. Further, the rate of time-out cases of our method stays lower.

From Table 4 and Table 6, we can see that for all test ontologies, the proportion of time-out cases in our system is less than that of the Explanation Workbench. The average time of computing justifications of these two systems is close. We observed that the time-out cases occurring in our system and the Explanation Workbench are not always the same. For example, when handling SNOMED CT by the Explanation Workbench, a time-out case is to compute justifications of the subsumption between the concept "`Entire inferior process of eleventh thoracic vertebra` " and the concept "`Musculoskeletal structure of thoracic spine`", which has 7 justifications, and takes nearly 20 seconds to finish the computation. In contrast, our system gives the results within one second. On the other hand, a time-out case in our system is to compute justifications between the concept "`Bisoprolol fumarate 10mg tablet`" and the concept "`Cardiovascular drug`" which has only one justification. For this case, the Explanation Workbench finishes immediately. The difference lies in that the efficiency of the black-box method is decided by the sizes of the target ontologies and the

number of reasoner calls, while the efficiency of our method depends on the number of nodes in the constructed EDGs.

8. Conclusions and Future Work

In this paper, we proposed a method of finding justifications of OWL EL ontologies based on the notion of explanation and Explanation Dependency Graph (EDG), which allows us to compute justifications from classification results directly. Since constructing EDGs can work independently from the task of reasoning, we do not need to modify the inner implementation of a reasoner like glass-box methods, or call reasoner a number of times like black-box methods, for the purpose of finding justification. Further, in order to make our algorithm more efficient, we proposed several blocking conditions and parallelize the construction of EDGs. The experimental results show that our algorithm is practical and performs better than glass-box and black-box implementations. In the future work, we plan to apply our method for other OWL languages, e.g., OWL QL and OWL RL. The basic idea is to define specific EDGs for different languages. However, there are also challenges. For example, OWL RL has more than twenty rules. This inevitably makes searching space large. We have to find more techniques to improve the performance.

Acknowledgments

This work was partially supported by NSFC grant 61672153 and the 863 program under Grant 2015AA015406.

References

[1] Z. Zhou, G. Qi, B. Suntisrivaraporn, A new method of finding all justifications in OWL 2 EL, in: Proc. of WI, 2013, pp. 213–220.

[2] K. A. Spackman, K. E. Campbell, R. A. Côté, SNOMED RT: a reference terminology for health care, in: Proc. of AMIA, 1997, pp. 334–339.

[3] F. Lécué, S. Tallevi-Diotallevi, J. Hayes, R. Tucker, V. Bicer, M. L. Sbodio, P. Tommasi, Smart traffic analytics in the semantic web with STAR-CITY: scenarios, system and lessons learned in dublin city, J. Web Sem. (2014) 26–33.

[4] A. Kalyanpur, B. Parsia, E. Sirin, J. A. Hendler, Debugging unsatisfiable classes in OWL ontologies, J. Web Sem. 3 (4) (2005) 268–293.

[5] S. Schlobach, R. Cornet, Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies, in: Proc. of IJCAI, 2003, pp. 355–362.

[6] F. Baader, R. Peñaloza, B. Suntisrivaraporn, Pinpointing in the Description Logic EL, in: Proc. of DL, 2007, pp. 15–24.

[7] A. Kalyanpur, B. Parsia, M. Horridge, E. Sirin, Finding all justifications of OWL DL entailments, in: Proc. of ISWC, 2007, pp. 267–280.

[8] F. Baader, B. Suntisrivaraporn, Debugging SNOMED CT Using Axiom Pinpointing in the Description Logic $\mathcal{EL}^+$, in: Proc. of KR-MED, 2008, pp. 245–254.

[9] B. Suntisrivaraporn, G. Qi, Q. Ji, P. Haase, A Modularization-Based Approach to Finding All Justifications for OWL DL Entailments, in: Proc. of ASWC, 2008, pp. 1–15.

[10] B. Suntisrivaraporn, Finding all justifications in SNOMED CT, J. ScienceAsia: Journal of the Science Society of Thailand 39 (1) (2013) 79–90.

[11] F. Baader, S. Brandt, C. Lutz, Pushing the $\mathcal{EL}$ Envelope, in: Proc. of IJCAI, 2005, pp. 364–369.

[12] F. Baader, C. Lutz, B. Suntisrivaraporn, Efficient Reasoning in $\mathcal{EL}^+$, in: Proc. Description Logics, 2006, pp. 158–170.

[13] A. Waterson, A. D. Preece, Verifying ontological commitment in knowledge-based systems, Knowl.-Based Syst. 12 (1) (1999) 45–54.

[14] Y. Ma, B. Jin, Y. Feng, Dynamic evolutions based on ontologies, Knowl.-Based Syst. 20 (1) (2007) 98–109.

[15] A. M. Khattak, K. Latif, S. Lee, Change management in evolving web ontologies, Knowl.-Based Syst. 37 (2013) 1–18.

[16] A. Kalyanpur, Debugging and Repair of OWL Ontologies, PhD thesis, The Graduate School of the University of Maryland 1 (4) (2006) 158–183.

[17] A. Kalyanpur, B. Parsia, B. C. Grau, Beyond Asserted Axioms: Fine-Grain Justifications for OWL-DL Entailments, in: Proc. of DL, 2006, pp. 75–84.

[18] A. Kalyanpur, B. Parsia, E. Sirin, B. C. Grau, Repairing unsatisfiable concepts in OWL ontologies, in: Proc. of ESWC, 2006, pp. 170–184.

[19] B. Parsia, E. Sirin, A. Kalyanpur, Debugging OWL Ontologies, in: Proc. of WWW, 2005, pp. 633–640.

[20] M. Horridge, B. Parsia, U. Sattler, Explaining inconsistencies in OWL ontologies, in: Proc. of SUM, 2009, pp. 124–137.

[21] M. Horridge, B. Parsia, U. Sattler, Justification Oriented Proofs in OWL, in: Proc. of ISWC, 2010, pp. 354–369.

[22] M. Horridge, Justification Based Explanation in Ontologies, PhD thesis, The School of Computer Science of the University of Manchester.

[23] M. Lee, N. Matentzoglu, B. Parsia, U. Sattler, A Multi-reasoner, Justification-Based Approach to Reasoner Correctness, in: Proc. of ISWC, 2015, pp. 393–408.

[24] M. Lee, N. Matentzoglu, U. Sattler, B. Parsia, Verifying reasoner correctness - A justication based method, in: Proc. of ORE, 2015, pp. 46–52.

[25] A. Kalyanpur, B. Parsia, E. Sirin, Black Box Techniques for Debugging Unsatisfiable Concepts, in: Proc. of DL, 2005, pp. 68–80.

[26] R. Reiter, A Theory of Diagnosis from First Principles, J. Artif. Intell. (1987) 57–95.

[27] S. Schlobach, Diagnosing Terminologies, in: Proc. of AAAI, 2005, pp. 670–675.

[28] S. Schlobach, Z. Huang, R. Cornet, F. van Harmelen, Debugging Incoherent Terminologies, J. Autom. Reasoning (2007) 317–349.

[29] 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: Proc. of SAT, 2016, pp. 521–530.

[30] N. Manthey, R. Peñaloza, S. Rudolph, Efficient axiom pinpointing in EL using SAT technology, in: Proc. of DL, 2016.

[31] H. Wang, M. Horridge, A. L. Rector, N. Drummond, J. Seidenberg, Debugging OWL-DL ontologies: A heuristic approach, in: Proc. of ISWC, 2005, pp. 745–757.

[32] M. Horridge, S. Bail, B. Parsia, U. Sattler, Toward cognitive support for OWL justifications, Knowl.-Based Syst. 53 (2013) 66–79.

[33] H. Stuckenschmidt, Debugging weighted ontologies, in: Proc. of CEUR-WS, 2013, pp. 1–8.

[34] L. Zhou, Dealing with inconsistencies in dl-lite ontologies, in: Proc. of ESWC, 2009, pp. 954–958.

[35] G. Qi, Z. Wang, K. Wang, X. Fu, Z. Zhuang, Approximating model-based abox revision in dl-lite: Theory and practice, in: Proc. of AAAI, 2015, pp. 254–260.

[36] X. Fu, G. Qi, Y. Zhang, Z. Zhou, Graph-based approaches to debugging and revision of terminologies in dl-lite, J. Knowl.-Based Syst. 100 (2016) 1–12.

[37] Y. Ma, R. Peñaloza, Towards parallel repair: An ontology decomposition-based approach, in: Proc. of IWDL, 2014, pp. 633–645.

[38] X. Meng, Scalable simple random sampling and stratified sampling, in: Proc. of ICML, 2013, pp. 531–539.

[39] M. Horridge, B. Parsia, U. Sattler, Explanation of OWL entailments in protege 4, in: Proc. of ISWC, 2008, pp. 65–74.