

Explanations in Logic-Based Reasoning Systems: A theoretical framework for generating explanations

Solomon Malesa
mlssol001@myuct.ac.za
University of Cape Town
Rondebosch, South Africa

ABSTRACT

Logic-Based Reasoning Systems generate conclusions from a set of premises asserted by a user. In today’s world, there is an increasing trend in adopting such systems to help make decisions. However, generating explanations for conclusions drawn by Logic-Based Reasoning Systems remains a challenge. The inability of domain experts (i.e. users) to efficiently debug incorrect conclusions or know why a system draws a conclusion indicates a usability issue in Logic-Based Reasoning Systems.

This paper investigates a theoretical framework used to build explanation tools. This includes providing background information about Logic-Based Reasoning Systems and a brief overview of Attributive Language with Concept Negation (\mathcal{ALC}) Description Logics (\mathcal{DL}). Furthermore, this paper shows how to find the minimal premises (justifications) of a conclusion (entailment) and how to generate explanations using justifications. After generating an explanation expressed in $\mathcal{ALC DL}$, this paper shows how to convert this explanation into a natural language equivalent, such as English. The framework presented in this paper allows OWL (Web Ontology Language) ontology engineers to build explanation tools from the ground up for existing Logic-Based Reasoning Systems.

CCS CONCEPTS

• **Computing methodologies** → *Description logics*; • **Information systems** → Expert systems; **Web Ontology Language (OWL)**;

KEYWORDS

OWL, Description Logics, Explanation

1 INTRODUCTION

The information age has changed the way humans make decisions. It has introduced the notion of outsourcing repetitive and computationally intensive activities to computers. This usually involves capturing the parameters of a real-world problem into a machine-readable form, writing algorithms to solve the problem, and then using the solution to augment decisions or inform areas of interest.

In order to capture the parameters of a real-world problem, an ontology needs to be developed. An ontology is defined as an engineering artefact[11] that documents a representation of “rich and complex knowledge about things, groups of things, and relations between things”[9]. The World Wide Web Consortium (W3C) maintains the standards of a state of the art ontology language called the Web Ontology Language (OWL).

There are many Logic-Based Reasoning Systems that have been successfully built using OWL. For instance, the medical field has built OWL Reasoning Systems for diagnosing medical conditions such as the SNOMED system [7] and the NCI Thesaurus expert system [8]. However good these systems are, domain experts are reluctant to adopt them because they do not provide users with enough information as to why a conclusion is made by the system.

Definition 1 (Entailment). An ontology O logically entails a proposition γ , denoted $O \models \gamma$, if and only if whenever O is interpreted to be true, so is γ .

Definition 2 (Justification). A justification \mathcal{J} is a minimal set of premises¹ that are required to be true (or present in an ontology O) in order for an entailment η to be true. The minimality of \mathcal{J} implies that if any premise $\alpha \in \mathcal{J}$ is removed from \mathcal{J} , then it is not possible to prove that $O \models \eta$.

A justification explanation is a type of explanation that provides the rationale for why an ontology draws an entailment[34]. The purpose of justification explanations is to aid domain experts with understanding entailments, debugging and repairing ontologies, and understanding ontologies in general [11]. This also includes allowing domain experts to learn what premises led to the inference of entailments and to discern why an ontology draws an inconsistent entailment.

Definition 3 (Inconsistency). An inconsistent ontology O is an ontology that contains a premise $\alpha \in O$, whereby α cannot be interpreted to be true if $O \setminus \{\alpha\}$ is interpreted to be true.

The increasing need to use Logic-Based Reasoning Systems to make decisions has led to an increase in the need to improve the usability of such systems[26]. This paper aims to provide a theoretical framework for generating justification explanations. This allows domain experts to be able to build explanation facilities from scratch for an existing Logic-Based Reasoning System built with OWL and restricted to $\mathcal{ALC DL}^2$.

An example implementation of this framework is illustrated in the supplementary information in section 10 (Figure 12). Moreover, there is a sister project by Cilliers Pretorius[25] which extends an Explanation Workbench created by Horridge[24] in Protégé[12, 13, 19]. Cilliers’s project improves an existing Logic-Based Reasoning System by applying the principles presented in this paper.

Remark 1. A *premise* is a type of proposition which is used to infer new propositions.

¹A premise is a proposition in which other propositions may be inferred from

²A decidable subset of First Order Logic (\mathcal{FOL}).

In this paper, Logic-Based Reasoning Systems are treated as black-boxes, which means that an explanation facility built with this framework is independent of the implementation of the Logic-Based Reasoning System. The structure of this paper is as follows: Section 2 outlines the background of Logic-Based Reasoning Systems. Section 3 establishes the preliminary knowledge required to understand this paper. Section 4 focuses on the techniques used to compute justifications. Section 5 focuses on how proof-trees³ are generated and selected. Section 6 focuses on how the selected proof trees are translated into English and presented to the user as an explanation for an entailment. Section 7 concludes this paper and the appendix contains an example of the implementation of this framework.

2 BACKGROUND AND RELATED WORK

This section is akin to a brief literature review. It outlines what has been done in the field of generating explanations for *ALC DL* and OWL Reasoning Systems and how does the background relate to this paper.

Early generations of Logic-Based reasoning systems attempted to generate explanations using unsuitable techniques such as the *canned text* approach. This approach requires the ontology engineer to anticipate all possible entailments and hard-code corresponding explanations[22]. Needless to say, it fails for ontologies as large as the SNOMED ontology which contains about 500,000 premises [31]– making this technique impractical.

Another approach deployed by early-generation systems is the *code translation* technique. This technique traces the actions of the underlying Reasoning System and translates them to English as an explanation for how a system draws an entailment. However, systems employing this technique fail to generate justification explanations. Examples of such systems include CENTAUR [5], Digitalis Advisor [32] and MYCIN [14] which assist in diagnosing medical conditions.

Swartout and Moore [21] have summarised the issues experienced with early-generation systems:

- **Narrow:** First-generation systems allowed a limited variety of queries.
- **Inflexible:** They only present explanations in textual form, they do not offer alternative forms of presentation such as graphs.
- **Insensitive:** They cannot take into account the user’s background knowledge
- **Unresponsive:** They do not allow users to ask follow-up questions.
- **Inextensible:** Users cannot extend first-generation systems’ explanation facilities to tailor them according to their needs.

In an attempt to solve the issues experienced by first-generation reasoning systems, a wave of second-generation systems emerged. These systems include NEOMYCIN [29], XPLAIN [33], PROSE

³A tree structure that has the entailment as the root, and the premises of the justification as the leaf nodes and the inferred propositions as the intermediate nodes

[6], and REX [38]. In addition to solving first-generation issues, Swartout and Moore [34] have identified the following characteristics as desirable in second-generation systems:

- **Fidelity:** The explanations generated accurately should reflect the actions of the system.
- **Understandability:** The explanations should be naturally understood by domain experts.
- **Low Construction Overhead:** The amount of effort put into developing explanation tools should be less than the amount of effort put into developing the corresponding reasoning system.
- **Efficiency:** The runtime efficiency of the reasoning system should not be degraded by the attached explanation tool.

In addition to the aforementioned characteristics, Tintarev and Masthoff [36] have identified the following explanation objectives that are desirable in Recommender Systems⁴:

- **Transparency:** Reasoning System should provide explanations of *how* conclusions were reached (i.e. how the system operates).
- **Scrutability:** Users must be able to inform the system that a particular explanation is incorrect.
- **Trustworthiness:** Users should be able to trust the system to make decisions on their behalf.
- **Effectiveness:** The decisions recommended by the systems should help users make better/optimal decisions.
- **Efficiency:** The time taken to make a decision should be minimised.
- **Satisfaction:** Users must be left satisfied with the decision recommended by the system.
- **Persuasiveness:** The system should persuade the user to accept the decision made by the system.

Another important contribution in this field was made by Nguyen et al. [37] on measuring the understandability of OWL inferences. They propose an algorithm to predict the understandability of OWL inferences. Furthermore, Nguyen’s thesis [23] implements an explanation generation tool for *SROIQ DL* (another decidable subset of *FOL*, but more expressive than *ALC DL*). His approach involves curating a set of common deductive rules and creating proof-trees from them in order to generate explanations in a natural language (English).

Other notable literature include Schlobach and Cornet’s [28] work which introduced methods used to debug incorrect entailments. Tied-up with the work by Baader and Hollunder [2] and a toolkit for justification entailments in OWL developed by Parsia and Ulrike Sattler [24], this gave rise to another technique: “*Minimal Unsatisfiable Preserving Sub-TBoxes (MUPS)*” [39]. This technique can prove that an entailment is false by finding the minimal subsets of an ontology that are justifications for unsatisfiable classes.

Kalyanpur et al. [16] were the earliest advocates for using justification explanations as a tool for debugging entailments. They

⁴A type of reasoning system which makes recommendations for online e-commerce products based on customers’ history and activities

Table 1: A table illustrating notation used for $\mathcal{ALC}\mathcal{DL}$

Symbol	Description	Example	Symbol	Description	Example
\top	\top is a class containing all instances	\top	\perp	\perp is an empty class	\perp
\sqcap	conjunction of classes	$C \sqcap D$	\sqcup	disjunction of classes	$C \sqcup D$
\neg	complement of a class	$\neg C$	\forall	universal restriction	$\forall R.C$
\exists	existential restriction	$\exists R.C$	\sqsubseteq	subclass	$C \sqsubseteq D$
\equiv	Class equivalence	$C \equiv D$	\doteq	Class definition	$C \doteq D$
$:$	Class assertion	$a : C$	$:$	Relation assertion	$(a, b) : R$

introduced this in the reasoning system Swoop [16, 18]. Furthermore, Kalyanpur’s thesis [15] demonstrates the value of using justifications for repairing and debugging OWL ontologies.

Today, justification explanations are used as the primary source of debugging and repairing OWL ontologies. This has inspired the research community to develop new algorithms [3, 4, 30, 31, 35] with justification explanations as the foundation of explanations in Logic-Based Reasoning Systems.

After conducting a literature review of related work, we assert that although current Logic-Based Reasoning Systems allow ontology engineers and domain experts to generate explanations, they are still riddled with the same issues that were identified by Swartout et al. [34] in 1993. This indicates that progress in this area is stagnant or optimising explanations in Logic-Based Reasoning Systems is difficult.

3 PRELIMINARIES

This section introduces the main ideas of $\mathcal{ALC}\mathcal{DL}$ and preliminary knowledge required throughout this paper.

3.1 Description Logics

\mathcal{DL} is a decidable⁵ subset of \mathcal{FOL} . There are many flavours of this logic with varying degrees of expressiveness. The $\mathcal{ALC}\mathcal{DL}$ is formally called the Attributive Language (\mathcal{AL}) with Class Complements Description Logics. Recall that in section 1, we defined an ontology as a representation of: “things”, “groups of things” and “relations between things”. In $\mathcal{ALC}\mathcal{DL}$, groups of things are referred to as *classes*, things are referred to as *instances* (of classes) and relations between things are referred to as *relations*.

Furthermore, ontologies built with $\mathcal{ALC}\mathcal{DL}$ are subdivided into the following categories: a terminological component (*TBox*) and an assertion component (*ABox*). The TBox contains propositions which define the domain of the ontology, this is usually seen as a set of class instances described in terms of their relations (i.e. properties). For example, All Cats are Animals. Counter to TBox propositions, the propositions of an ABox are typically of the form: A is an instance of B, for example, Fluffy is a Cat.

⁵Decidability in this context refers to the existence of effective methods to solve a problem. This means that inferences made using this logic will return a solution within a reasonable amount of time

Table 1 illustrates the notation used throughout this paper. Let C and D be classes, a and b be instances and R be a relation. Let N_C be a set of class names, N_R be a set of relation names, and N_O be a set of instance names.

Definition 4 (Class). A class is defined as:

- \top , A set of all instances.
- \perp , An empty set.
- $A \in N_C$ is a class.
- $C \sqcap D$, the conjunction of C and D is a class.
- $C \sqcup D$, the disjunction of C and D is a class.
- $\neg C$, the complement of C is a class.

Let $R \in N_R$.

- $\forall R.C$, the universal restriction of a class by a relation is a class.
- $\exists R.A$, the existential restriction of a class by a relation is a class.

Definition 5 (Knowledge Base). A knowledge base $\mathcal{KB} = (T, A)$ is an ordered pair containing a Tbox (T) and an Abox (A).

To give meaning to our \mathcal{KB} , we define the semantics of $\mathcal{ALC}\mathcal{DL}$, which map the interpretation of the symbols in the \mathcal{KB} to a possible world. In predicate logic, the interpretation function maps the symbols to binary values. However, $\mathcal{ALC}\mathcal{DL}$ maps the symbols to a domain of classes and instances of classes.

In the context of Logic, an interpretation is the assignment of meaning to the symbols in the \mathcal{KB} .

Definition 6 (Interpretation). We formally define an interpretation to be a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over a tuple (N_C, N_R, N_O) which consists of:

- A domain $\Delta^{\mathcal{I}} \neq \emptyset$
- An interpretation function $\cdot^{\mathcal{I}}$ which maps an instance a , class C and relation R as follows:
 - (1) $\cdot^{\mathcal{I}} : a \mapsto a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
 - (2) $\cdot^{\mathcal{I}} : C \mapsto C^{\mathcal{I}}$
 - (3) $\cdot^{\mathcal{I}} : R \mapsto Q$, where $Q \subset \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

such that:

- $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
- $\perp^{\mathcal{I}} = \emptyset$
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \sqcup D^{\mathcal{I}}$
- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \sqcap D^{\mathcal{I}}$
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
- $(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \text{for every } y, (x, y) \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}$

- $(\exists R.C)^I = \{x \in \Delta^I \mid \text{there exists } y, (x, y) \in R^I \text{ and } y \in C^I\}[1]$

Definition 7 (Entailment with respect to an interpretation). We define logical consequence as follows: Suppose we have two interpretations $Mod(\alpha)$ and $Mod(\beta)$ for α and β , respectively. Then α entails β (written $\alpha \models \beta$) if and only if $Mod(\alpha) \subseteq Mod(\beta)$.

Definition 8 (Terminological Box). A *TBox* \mathcal{T} is now formally defined with the following two conditions:

- $\mathcal{I} \models C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
- $\mathcal{I} \models \mathcal{T}$ iff $\mathcal{I} \models \Phi$ for every $\Phi \in \mathcal{T}$ [1]

Definition 9 (Assertion Box). An *ABox* \mathcal{A} is now defined with the following three conditions:

- $\mathcal{I} \models a : C$ if and only if $a^{\mathcal{I}} \in C^{\mathcal{I}}$
- $\mathcal{I} \models (a, b) : R$ if and only if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
- $\mathcal{I} \models \mathcal{A}$ if and only if $\mathcal{I} \models \phi$ for every $\phi \in \mathcal{A}$ [1]

We conclude the overview of description logics by refining our definition of \mathcal{KB} constructed with $\mathcal{ALC} \mathcal{DL}$. $\mathcal{KB} = (\mathcal{T}, \mathcal{A})$ if the following condition holds: $\mathcal{I} \models \mathcal{KB}$ iff $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{A}$.

Recall that a justification for an entailment is a minimal set of premises that are required to be true in order for an entailment to hold [20]. From the description $\mathcal{ALC} \mathcal{DL}$, we know that premises are instances of classes asserted by an ontology engineer.

We now update this definition with a more precise definition:

Definition 10 (Justification). Suppose an ontology \mathcal{O} and an entailment \mathcal{W} such that $\mathcal{O} \models \mathcal{W}$. Then \mathcal{J} is the justification for the entailment \mathcal{W} if:

- $\mathcal{J} \sqsubseteq \mathcal{O}$ and
- $\forall \mathcal{Z} \subsetneq \mathcal{J}, \mathcal{Z} \not\models \mathcal{W}$ [11]

Definition 11 (Satisfiability). We say that a class C is unsatisfiable with respect to an ontology \mathcal{O} , if $C^{\mathcal{I}} = \emptyset$ for every interpretation \mathcal{I} of the ontology \mathcal{O} . This implies that a class C is satisfiable in the ontology \mathcal{O} if there is an interpretation \mathcal{I} such that $C^{\mathcal{I}} \neq \emptyset$.

Definition 12 (Consistency and Inconsistency). A more precise definition of consistency is that an ontology \mathcal{O} is consistent if there exists an interpretation \mathcal{I} in \mathcal{O} . However, an ontology \mathcal{O} is inconsistent if there is no interpretation \mathcal{I} in \mathcal{O} that satisfies every premise in \mathcal{O} . This is described as $\mathcal{O} \models \top \sqsubseteq \perp$, meaning the ontology \mathcal{O} entails everything.

3.2 Services in Logic-Based Reasoning Systems

This subsection briefly outlines what it means for a logic-based system to reason.

Reasoning in Logic-Based Systems occurs when the system discerns if an entailment follows from its ontology. Since this paper treats reasoning systems as a black-boxes, the details of the implementation of a reasoner will not be discussed. However, a brief overview of the types of standard reasoning tasks is given in the following list:

- **Consistency Checking:** In this case, the system determines if there is at least a single interpretation \mathcal{I} of the ontology \mathcal{O} .
- **Satisfiability Checking:** In this case, the system determines if $C^{\mathcal{I}} \neq \emptyset$.
- **Subsumption Testing:** In this case, the system determines if given two classes C and D , is $\mathcal{O} \models C \sqsubseteq D$ true? Which essentially asks the question, is $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every interpretation \mathcal{I} of the ontology \mathcal{O} ?
- **Instance Checking:** In this case, the systems checks if the instance a belongs to a class C . This means that the system tries to find an interpretation \mathcal{I} such that $a^{\mathcal{I}} \in C^{\mathcal{I}}$

4 JUSTIFICATION GENERATION

Recall that we defined a justification \mathcal{J} as any minimal subset (a set of premises) of the ontology \mathcal{O} from which an entailment ϕ can be drawn [15]. The condition of minimality implies that if a premise $\alpha \in \mathcal{J}$ is removed from \mathcal{O} , then it would be impossible to infer ϕ from the ontology \mathcal{O} .

We present two algorithms developed by Horridge[11] for computing justifications for black-box Logic-Based Reasoning Systems. The first algorithm computes a single justification \mathcal{J} for an entailment η and the second algorithm computes all possible justifications $\{\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3, \dots, \mathcal{J}_n\}$ for η . Horridge[11] claims that it is a useful practise to distinguish between the two because the algorithm used for computing a single justification may have subroutines that can be used by the algorithm used for computing all justifications. Another reason for distinguishing between the two algorithms is that at times domain experts may need a single justification to debug an ontology- making it unnecessary to compute all possible justifications.

4.1 Generating A Single Justification

Algorithms used to compute justifications are generally called *premise pinpointing algorithms*. Algorithm 1 is adopted from Horridge's [11] algorithms for computing a single justification \mathcal{J} .

Algorithm 1 is a brute-force algorithm. It constructs \mathcal{J} , given \mathcal{O} for an entailment η . It has two phases, the expansion phase (illustrated in algorithm 2) and the contraction phase (illustrated in algorithm 3).

The first step is to determine if \mathcal{O} is empty. If so, then \mathcal{J} is \emptyset . We assume that $\eta \neq \emptyset$ because the justification for an empty set is \emptyset . The next step is to continually add premises α to \mathcal{S} until $\mathcal{S} \models \eta$. This is called the expansion phase and is illustrated by algorithm 2 in line 5 to 8. When $\mathcal{S} \models \eta$, then it is guaranteed that \mathcal{S} contains a justification \mathcal{J} .

Algorithm 1: Computing Single Justification

Data: Ontology O , Entailment η **Result:** Justification \mathcal{J}

```
1  $S \leftarrow \emptyset$ 
2 if  $O \equiv \emptyset$  then
3   return  $\emptyset$ 
4 else
5    $S \leftarrow \text{Expand}(S, O, \eta)$ 
6    $\mathcal{J} \leftarrow \text{Contract}(S, \eta)$ 
7 end
8 return  $\mathcal{J}$ 
```

Algorithm 2: Expand

Data: Ontology O , Entailment η , Set S **Result:** Set S

```
1 if  $O \neq \eta$  then
2   return  $\emptyset$ 
3 else
4    $\mathcal{P} \leftarrow O$ 
5   while  $S \neq \eta$  do
6      $\mathcal{P} \leftarrow \mathcal{P} \setminus \{\alpha\}$ 
7      $S \leftarrow S \cap \{\alpha\}$ 
8   end
9 end
10 return  $S$ 
```

Algorithm 3: Contract

Data: Entailment η , Set S **Result:** Set S

```
1  $\mathcal{P} \leftarrow \emptyset$ 
2 for  $i \leftarrow 0$  to  $|S|$  do
3   if  $S \setminus (\alpha_{ith} \in S) \models \eta$  then
4      $\mathcal{P} \leftarrow \mathcal{P} \cup \{\alpha_{ith}\}$ 
5      $S \leftarrow S \setminus \{\alpha_{ith}\}$ 
6   end
7 end
8 return  $S \setminus \mathcal{P}$ 
```

After the expansion phase, we know that $\mathcal{J} \subset S$. The contraction phase prunes the set S by removing all premises α which when removed, leaves $S \models \eta$ still true. This is shown in algorithm 3 in line 2 to 5, in which premises that do not form \mathcal{J} are inserted into set \mathcal{P} and then the algorithm 3 returns S with \mathcal{P} removed.

Although algorithm 1 does in fact return a correct justification, we notice that the runtime complexity of this algorithm, for the worst and average cases, is $O(n)$, where $n = |O|$. This is obviously inefficient for large ontologies. In the best case scenario, $|\mathcal{J}| = 1$ and the runtime complexity is $O(1)$ for algorithm 2 and 3. We note that the total number of unnecessary tests required, in the worst case scenario, is $2 \times (|O| - |\mathcal{J}|)$. We also observe that the space complexity in the worst and average cases is $O(n)$. In the best case

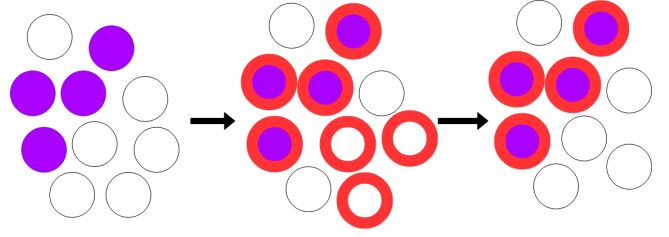


Figure 1: A diagram illustrating the application of algorithm 1, 2 and 3 pinpointing \mathcal{J}

scenario, $|\mathcal{J}| = 1$ and thus the space complexity is $O(1)$, in the best case scenario.

Figure 1 illustrates the basic idea of algorithm 1. The first set of premises represent O with purple circles representing the premises $\{\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n\} \equiv \mathcal{J}$. The second set of premises represent the resulting set S (represented by red circles) after applying algorithm 2. Finally, the last set of premises represent $S \equiv \mathcal{J}$ after applying algorithm 3.

To improve algorithm 1, we need to employ strategies which minimise the number of tests required by the contraction (algorithm 3) and expansion (algorithm 2) phase. Horridge [11] proposed multiple techniques to optimise the expansion phase and the contraction phase.

4.1.1 Expansion Optimisation: Expansion by Selection Function.

The main idea is to use a function $\sigma_{O, \eta} : \mathcal{P}(O) \rightarrow \mathcal{P}(O)$ where $\mathcal{P}(O)$ denotes the powerset of the ontology O . The function σ generates all possible subsets of O .

This reduces the amount of tests required in the expansion phase. Based on the work done by Huang and Harmelen [10], it is evident that this method is empirically superior than Algorithm 2. The chances of selecting S whereby $\mathcal{J} \subset S$ increases with an increasing size of \mathcal{J} . In the best case scenario, only a single test (i.e. does $S \models \eta$?) has to be done. Horridge[11] summarises the benefits of this method by claiming that it performs faster for smaller sets of premises and it results in fewer tests during the contraction phase. The best case runtime complexity of this strategy is $O(1)$, however, the space complexity for all cases of this strategy is $O(2^n)$.

4.1.2 Expansion Optimisation: Expansion by Modularisation.

A module M of an ontology O is a set M such that $M \subset O$. This strategy requires the ontology engineer to dissolve O into separate and independent fragments $\{M_1, M_2, M_3, \dots, M_n\}$ based on the similarity of the premises $\alpha \in O$. The process of fragmenting O (i.e. modularising O) is beyond the scope of this paper. However, Horridge [11] claims that the benefits of this approach are similar to those of expansion by a selection function. Moreover, computing modules does not affect the time taken to compute single justifications. The best case runtime complexity of this strategy is $O(1)$, however, the space complexity of this strategy is $O(n)$ if

$$\bigcap_{i=1}^n M_i \equiv O.$$

4.1.3 Contraction Optimisation: Sliding Window. The Sliding Window strategy is similar to algorithm 3, however, instead of selecting a single premise α to remove during each step, a set of premises $\mathcal{S}'' \subset \mathcal{S}$ is removed. The size of the window dictates the number of premises removed in each step. The advantage of this strategy is that more than one premise is removed from the set at each step. Kalyanpur et al. [17] have shown that for their ontology, a window of size 10 (i.e. removing 10 premises per step) provides the best performance. However, it is never made clear whether this performance depends on the type of ontology tested or if these results hold without any loss of generality [11]. The space complexity for all cases is $O(n)$, where $n = |O|$. The runtime complexity is $O(\frac{n}{|\mathcal{S}''|})$, however for the best case scenario, the runtime is $O(1)$.

4.1.4 Contraction Optimisation: Divide and Conquer. The divide and conquer strategy improves on the previous strategy by splitting the ontology O into multiple parts O_i , where $|O_i| = 1$. Then $\forall O_i \neq \eta$, O_i and O_{i+1} are merged to form a larger set and test if $O_i \cap O_{i+1} \models \eta$. This process is repeated by halving and merging subsets O_i until \mathcal{J} is found. The main advantage of this strategy is that the runtime complexity of this strategy is $O(\log n)$ better than algorithm 3 which has a runtime complexity $O(n)$.

4.2 Generating All Justifications

The necessity of multiple justifications for an entailment η stems from the need to aid the understanding of domain experts when they debug an ontology. This may be induced by the need for an alternative justification for an entailment, which ultimately produces a different explanation for the same entailment η .

Algorithm 4: Computing All Justifications (brute-force)

Data: Ontology O , Entailment η
Result: Set of Justifications $\{\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3, \dots, \mathcal{J}_n\}$

```

1  $\mathcal{J}' \leftarrow \emptyset$ 
2  $\mathcal{K} \leftarrow \emptyset$ 
3  $\mathcal{S} \leftarrow \mathcal{P}(O)$ 
4 for  $\beta \in \mathcal{S}$  do
5   if  $\beta \models \eta$  then
6      $\mathcal{J}' \leftarrow \mathcal{J}' \cup \{\text{ComputeSingleJustification}(\beta)\}$ 
7   end
8 end
9 for  $\delta \in \mathcal{J}'$  do
10  if  $\delta \notin \mathcal{K}$  then
11     $\mathcal{K} \leftarrow \mathcal{K} \cup \{\delta\}$ 
12  end
13 end
14 return  $\mathcal{K}$ 

```

Algorithm 4 is a brute-force strategy. We start by computing the power set of the ontology O and set it to \mathcal{S} . We then test each element $\iota \in \mathcal{S}$ for justifications of η . Then we create a new set $\mathcal{J}' = \{\iota \mid \forall \iota \in \mathcal{S}, \iota \models \eta\}$. In line 9 - 13 we remove all duplicate justifications in \mathcal{J}' .

This strategy works fine for small ontologies, however, for very large ontologies it is impractical since ComputeSingleJustification has to be called $2^{|O|}$ times. Thus the runtime and space complexity of this strategy is exponential in nature (i.e. $O(2^n)$).

To improve algorithm 4, we turn to ‘‘Reiter’s Hitting Set Tree’’ Algorithm [27]. We modify Horridge’s [11] algorithm that creates a tree that is finite in size and has justifications as the nodes of the tree and premises as the edges of the tree to produce algorithm 5.

Algorithm 5: Computing All Justifications

Data: Ontology O , Entailment η
Result: Set of Justifications $\{\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3, \dots, \mathcal{J}_n\}$

```

1  $v_{root} \leftarrow \text{ComputeSingleJustification}(\eta)$ 
2 if  $v_{root} \equiv \emptyset$  then
3   return  $\emptyset$ 
4 end
5  $\mathcal{K} \leftarrow v_{root}$ 
6  $\mathcal{S}_{stack}.\text{push}(v_{root})$ 
7 while  $\mathcal{S}_{stack} \neq \emptyset$  do
8    $v_{head} \leftarrow \mathcal{S}_{stack}.\text{pop}()$ 
9    $\mathcal{P} \leftarrow \emptyset$ 
10  for each child  $\iota$  of  $v_{head}$  do
11     $\mathcal{P} \leftarrow \mathcal{P} \cup \{\text{edge}(v_{head}, \iota)\}$ 
12  end
13   $\alpha \leftarrow \delta \in v_{head} \mid \alpha \notin \mathcal{P}$ 
14   $\mathcal{S} \leftarrow \text{PremisesTracedFromRootTo}(v_{head}) \cup \{\alpha\}$ 
15   $\mathcal{O}' \leftarrow \mathcal{O} \setminus \mathcal{S}$ 
16   $\mathcal{J} \leftarrow \emptyset$ 
17  if  $\mathcal{O}' \models \eta$  then
18     $\mathcal{J} \leftarrow \text{ComputeSingleJustification}(\mathcal{O}')$ 
19  end
20   $v_{new} \leftarrow \text{CreateNewNode}(\mathcal{J})$ 
21   $\text{SetChildNodeFor}(v_{head}, v_{new})$ 
22   $\text{SetEdge}(v_{head}, v_{new}, \alpha)$ 
23   $\mathcal{K} \leftarrow \mathcal{K} \cup v_{new}$ 
24   $\mathcal{S}_{stack}.\text{push}(v_{new})$ 
25 end
26 return  $\mathcal{K}$ 

```

We denote intermediate nodes with v and successor nodes v' , which are connected to v by an edge with a premise α such that α is in the justification contained in v but not in the justification contained in v' . When $v' \equiv \emptyset$, then it is a leaf node. Furthermore, for any node v'' , the set of premises that are in the path from the root to the node v'' do not intersect with the justification in v'' .

The following enumeration is a high level description of algorithm 5 (line 1 - 26) which generates a new justification for a new node v' from a justification in v .

- We begin by choosing a premise $\alpha \in v$ which is not in the edges of the children of node v .
- We then let \mathcal{S} be the disjunction of the chosen premises $\{\alpha\}$ and the set of premises found in the path traced from v to the root node of the tree. Then we set $\mathcal{O}' \leftarrow \mathcal{O} \setminus \mathcal{S}$.

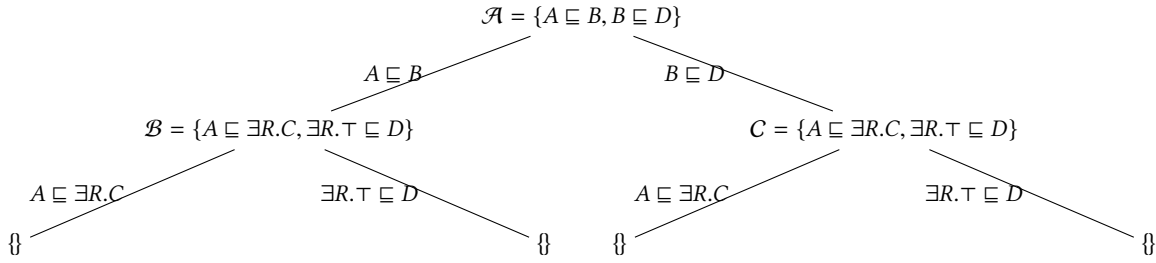


Figure 2: An example of a hitting set tree [11]

- We test if the new ontology \mathcal{O}' entails η . If that is the case, we compute a justification \mathcal{J} for η using the new ontology \mathcal{O}' . Whereas if the new ontology \mathcal{O}' does not entail η , we set the justification $\mathcal{J} = \emptyset$
- We then make a new node v' and set its justification to be \mathcal{J}
- We add an edge e between v and v' , and we label e with the premise α
- We reinsert all the premises in \mathcal{S} back into \mathcal{O}

The enumerated procedure is repeated until no new successor nodes can be generated. Upon termination, all the intermediate nodes form a new set of all the justifications for the entailment η .

The example tree in Figure 2 is generated from an ontology containing the premises $A \sqsubseteq B$, $B \sqsubseteq D$, $A \sqsubseteq \exists R.C$, and $\exists R.\top \sqsubseteq D$ for an entailment $A \sqsubseteq D$ [11]. It results in three justifications \mathcal{A} and $\mathcal{B} \equiv \mathcal{C}$.

Although computing the hitting trees is practical for small ontologies. In the worst case scenario, its space complexity is of the order $2^{n+1} - 1$. This means that if an entailment is incorrect, then $2^{n+1} - 1$ justifications have to be debugged. The number of justifications for an entailment, say $A_{i-1} \sqsubseteq A_n$ is exponential (i.e. $|\mathcal{J}| = 2^n$).

Nguyen [23] has conducted a study which shows that the typical size of justifications is $1 < |\mathcal{J}| < 192$ premises, with about 69% of the justifications containing ten or fewer premises.

Horridge [11] suggests two techniques to improve the algorithm for computing all justifications.

4.2.1 Optimisation: Early Termination. He suggests that there should be some mechanism that allows the termination of search for new justifications. He introduces the idea of open nodes and closed nodes in which the closed nodes are all nodes that cannot be extended to create new justifications and the open nodes are those that can be extended to create new justifications. This strategy increases the performance of the algorithm by avoiding unnecessary searches, for instance if two different paths could lead to different nodes containing the same justification (as shown in Figure 2), then only one of the paths is explored.

4.2.2 Optimisation: Justification Reuse. Another technique he suggested is that of reusing justifications when extending the tree.

This saves a lot of redundant calls to the reasoning system. For instance, the justification in v is generated by computing the entailment η with respect to $\mathcal{O} \setminus \mathcal{S}$ where \mathcal{S} is the set of premises found from tracing a path from v to the root. Then if there is a node v' containing a justification \mathcal{J} and \mathcal{J} does not intersect with \mathcal{S} , then \mathcal{J} can be inserted to the node v .

5 GENERATING EXPLANATIONS

So far, we have been able to determine which premises α from an ontology \mathcal{O} are contained in a justification \mathcal{J} for an entailment η . In this section, we illustrate how to generate explanations for an entailment η , given a justification \mathcal{J} .

A justification is analogous to a set of axioms for a theorem. This means that an explanation for the pertinent conclusion needs to be computed by forming new propositions that are entailed within the justification. More importantly, these propositions should form a logical sequence such that that these propositions form a logical chain that leads to the conclusion. We call these new propositions *intermediate entailments*.

Definition 13 (Explanation). We define an explanation $\Sigma(\eta)$ of an entailment η to be a pair $(\mathcal{J}, \{\epsilon_1, \epsilon_2, \epsilon_3, \dots, \epsilon_n\})$, where \mathcal{J} is a justification for an entailment η and $\{\epsilon_1, \epsilon_2, \epsilon_3, \dots, \epsilon_n\}$ is a minimal set of intermediate entailments that link the justification \mathcal{J} to the entailment η . The minimality of $\{\epsilon_1, \epsilon_2, \epsilon_3, \dots, \epsilon_n\}$ implies that if an entailment $\beta \in \{\epsilon_1, \epsilon_2, \epsilon_3, \dots, \epsilon_n\}$ is removed then $\{\epsilon_1, \epsilon_2, \epsilon_3, \dots, \epsilon_n\} \setminus \beta$ cannot logically link η with \mathcal{J} .

Figure 4 illustrates what a construction of an explanation for an entailment E looks like. In this case, $\eta = E$, $\mathcal{J} = \{ax1, ax2, ax3, ax4, ax5\}$, and $\{\epsilon_1, \epsilon_2, \epsilon_3, \dots, \epsilon_n\} = \{le1, le2\}$. To illustrate what we mean by linking \mathcal{J} to η , we observe that $\{ax1, ax2\} \models le1$, $\{ax3, ax4, ax5\} \models le2$, and $\{le1, le2\} \models E$. Hence without $le1$ and/or $le2$, it cannot be proven that $E \models E$.

Horridge [11] calls these intermediate nodes lemmas. The main purpose of a lemma is to make apparent the intermediate steps taken by a logic-based reasoning system to reach a conclusion that $\mathcal{O} \models \eta$. The tree presented in Figure 4 is called a proof-tree. In this section, we show an algorithm for constructing such trees.

This algorithm is based on two strategies: (1) The first strategy relies on the notion of lemmatisation (illustrated in algorithm 6), in

which a subset S of a justification is replaced by a simple premise that summarises the subset S (i.e. lemma). For example, $\{ax1, ax2\}$ are replaced by $le1$. (2) The second strategy relies on ordering a series of justifications that have been lemmatised to construct a proof tree (illustrated in algorithm 7).

5.1 Lemmatisation

Suppose we have an ontology O from which we derive a justification \mathcal{J} for an entailment η whereby $O \models \eta$. The goal is to lemmatise \mathcal{J} into \mathcal{J}' such that \mathcal{J}' is “less complex” than \mathcal{J} . Algorithm 6, adopted from Horridge [11], describes how to lemmatise a justification \mathcal{J} .

Algorithm 6: Lemmatising a Justification

Data: Justification \mathcal{J} , Entailment η
Result: Set \mathcal{J}_{result}

```

1 if  $\mathcal{J} \equiv \{\eta\}$  then
2   | return  $\{\eta\}$ 
3 end
4  $\mathcal{S}_{tidy} \leftarrow \text{ComputeDeductiveClosure}(\mathcal{J}, \eta) \setminus \{\eta\}$ 
5  $\mathcal{V} \leftarrow \text{ComputeAllJustifications}(\mathcal{S}_{tidy}, \eta)$ 
6  $a \leftarrow \text{ComputeComplexity}(\mathcal{J}, \eta)$ 
7  $\mathcal{J}_{result} \leftarrow \mathcal{J}$ 
8 for  $\beta \in \mathcal{V}$  do
9   | if  $\text{ComputeComplexity}(\beta, \eta) < a$  then
10    | |  $a \leftarrow \text{ComputeComplexity}(\beta, \eta)$ 
11    | |  $\mathcal{J}_{result} \leftarrow \beta$ 
12    | end
13 end
14 return  $\mathcal{J}_{result}$ 

```

The input to algorithm 6 is \mathcal{J} and η and it outputs \mathcal{J}_{result} . If there is no way to simplify the premises in \mathcal{J} , then the algorithm returns \mathcal{J} .

Definition 14 (Tidy Set). We define a tidy set of premises as a set of premises \mathcal{S} in which $\mathcal{S} \not\models \top \sqsubseteq \perp$, $\mathcal{S} \not\models A \sqsubseteq \perp$, and $\mathcal{S} \not\models \top \sqsubseteq A$. This means that a tidy set of premises is a set \mathcal{S} that does not contain classes that are equivalent to \perp if $\mathcal{S} \models A \sqsubseteq \perp$ and it does not contain classes that are equivalent to \top if $\mathcal{S} \models \top \sqsubseteq A$ [11].

Definition 15 (Deductive Closure). A deductive closure of \mathcal{J} is a set of all propositions that can be entailed from \mathcal{J} .

Horridge has shown that $|\text{ComputeDeductiveClosure}(x, y)| = \infty$ [11]. The implementation details of *ComputeDeductiveClosure* are beyond the scope of this paper, but for practical purposes we assume that the deductive closure of \mathcal{J} is finite.

Given that the calculation of the complexity of a premise is complex in itself, we do not illustrate how to compute it. However, the essence of the function *ComputeComplexity* is to assign scores to premises in a justification that reflects their complexity. These scores are then used in algorithm 6 to determine which set of premises should be replaced with a less complex set of premises. The runtime complexity of algorithm 6 is $O(n)$.

5.2 Generation of Proof Tree

In this subsection, we show an algorithm 7 which uses algorithm 6 as a subroutine to compute a proof-tree by lemmatisation.

Algorithm 7: Proof Tree Generation

Data: Justification \mathcal{J} , Entailment η
Result: Proof-Tree Σ

```

1  $\text{Root}(\Sigma) \leftarrow \eta$ 
2  $\zeta \leftarrow \text{Lemmatised}(\mathcal{J}, \eta)$ 
3 for  $\theta \in \zeta$  do
4   |  $\text{InsertChildNode}(\text{getRoot}(\Sigma), \theta)$ 
5 end
6 if  $\{\text{AllLeafNodes}(\Sigma)\} \equiv \mathcal{J}$  then
7   | return  $\Sigma$ 
8 end
9 while  $\text{ContainsLemmatisablepremise}(\{\text{AllLeafNodes}(\Sigma)\})$  do
10  |  $\alpha \leftarrow \text{getLemmatisableNode}(\{\text{AllLeafNodes}(\Sigma)\})$ 
11  |  $\beta \leftarrow \text{ComputeSingleJustification}(\mathcal{J}, \alpha)$ 
12  |  $\gamma \leftarrow \text{Lemmatised}(\beta, \alpha)$ 
13  | for  $\delta \in \{\text{AllLeafNodes}(\Sigma)\}$  do
14    | | if  $\delta \equiv \alpha$  then
15      | | | for  $\kappa \in \gamma$  do
16        | | | |  $\text{InsertChildNode}(\delta, \kappa)$ 
17      | | | end
18    | | end
19  | end
20 end
21 return  $\Sigma$ 

```

Suppose we are given a justification \mathcal{J} and an entailment η , as shown in Figure 5, we show the mechanics of an algorithm for generating a proof tree.

- (1) The first step is to lemmatise \mathcal{J} using the algorithm 6 to give a new lemmatised justification ζ , shown in Figure 6.
- (2) We then create an initial proof tree Σ using ζ , shown in Figure 7.
- (3) We check if ζ contains leaf nodes with premises that can be lemmatised.
 If so:
 - (a) We choose a lemma α in our proof tree Σ that can be lemmatised. We compute a justification β for α using \mathcal{J} , shown 8.
 - (b) We then lemmatise β in order to generate a new justification γ , shown in Figure 9.
 - (c) We insert γ to the proof tree Σ , shown in Figure 10.
 After this, we return to step 3.
 Else, continue to step 4.
- (4) At this point, the construction of the proof tree is now completed, as shown in Figure 11.

The analysis of the efficiency of this algorithm requires an empirical study (which is outside the purview of this paper). However, we assert that this algorithm is acceptable because it is general and does not require us to know the implementation details of the underlying logic-based reasoner. Another aspect that we observe

Table 2: A table illustrating a comprehensive list of OWL atomic premises, translated into a natural language

ID	OWL premise	Natural Language	ID	OWL premise	Natural Language
1	$A \sqsubseteq B$	“Every A is a B ”	15	$Irr(R_o)$	“Nothing R_o itself”
2	$A \equiv B$	“An A is equivalent to B ”	16	$Sym(R_o)$	“ $X R_o Y$ means the same as $Y R_o X$ ”
3	$Dis(A, B)$	“No A is a B ” or “Nothing is both an A and a B ”	17	$Asym(R_o)$	“The property R_o is asymmetric”
4	$Dom(R_o, A)$	“Anything that R_o something is an A ”	18	$Tra(R_o)$	“If $X R_o Y$ and $Y R_o Z$ then $X R_o Z$ ”
5	$Rng(R_o, A)$	“Anything that something R_o is an A ”	19	$R_d \sqsubseteq S_d$	“If $X R_d$ a value then $X S_d$ that value”
6	$Dom(R_d, A)$	“Anything that R_d some value is an A ”	20	$R_d \equiv S_d$	“The properties R_d and S_d are equivalent”
7	$Rng(R_d, D_r)$	“Any value that something R_d is a D_r ”	21	$Dis(R_d, S_d)$	“The properties R_d and S_d are disjoint”
8	$R_o \sqsubseteq S_o$	“If $X R_o Y$ then $X S_o Y$ ”	22	$Fun(R_d)$	“Everything R_d at most one value”
9	$R_o \equiv S_o$	“The properties R_o and S_o are equivalent”	23	$a \in A$	“ a is an A ”
10	$Dis(R_o, S_o)$	“The properties R_o and S_o are disjoint”	24	$R_o(a, b)$	“ $a R_o b$ ”
11	$InvS(R_o, S_o)$	“ $X R_o Y$ means the same as $Y R_o X$ ”	25	$Sam(a, b)$	“ a and b are the same instances”
12	$Fun(R_o)$	“Everything R_o at least one thing”	26	$Dif(a, b)$	“ a and b are different instances”
13	$InvFun(R_o)$	“If there is something X then at most one thing $R_o X$ ”	27	$\exists R_o . A$	“ R_o at least one A ”
14	$Ref(R_o)$	“Everything R_o itself”.	28	$\forall R_o . A$	“ R_o only A ”

is that this method of computing proof trees does not have any mechanism that prevents it from producing explanations that are too fine-grained. It is also possible that this method may produce explanations that are needlessly longer than the explanations produced by glass-box systems. The runtime complexity of algorithm 7 is $O(n^2)$.

6 TRANSLATION TO ENGLISH AND PRESENTATION

Given that we now have an explanation expressed in $\mathcal{ALC DL}$, in this section we show how to best present this explanation in a natural language (e.g. English)

6.1 OWL Atomic premises

In this subsection we give a list of common patterns for OWL atomic premises and we illustrate how they are translated into a natural language. We let A and B be class names, R_o and S_o be property names, R_d and S_d be data property names, a and b be instances of classes, and D_r a data range and D_t be a data type. Table 2 shows a comprehensive list of OWL atomic premises illustrating how to translate them into a natural language.

Definition 16 (atomic premise). An atomic premise is a premise containing a single OWL constructor.

6.2 Compound premises

In order to convert an explanation for an entailment η w.r.t an ontology O expressed in $\mathcal{ALC DL}$, we need to capture all lexemes of the ontology.

Definition 17 (lexeme). A lexeme is a minimal unit of a symbol or group of symbols in a lexicon with meaning. More succinctly, it is a minimal semantic unit of a language.

Definition 18 (Compound Premise). A compound premise, in the context of this paper, is any premise that contains two or more constructors. For example, $A \sqcap B$ is an atomic premise whereas

$A \sqcap B \sqcap C$ is a compound premise. More succinctly, a compound premise is a non-atomic premise.

In OWL, there are five distinct lexemes:

- **Classes.** Examples of classes include student and town. In English classes fall under the category of common **nouns**.
- **Instance of classes.** Examples of instances of classes include Cape Town, which is an instance of a class City. In English instances of classes fall under the category of proper **nouns**.
- **Relations.** Examples of relations include ‘buy’ and ‘owns’. In English these fall under **verbs**.
- **Data Types.** Data types describe the data used in an ontology O , for example string and integer. In English, data types fall under **nouns**.
- **Literals.** Examples of literals include: false and “2 metres”. In English, these are proper **nouns**.

Given that we are now able to translate atomic premises assembled with OWL constructors using the patterns in Table 2, we employ a parser that takes a statement (proposition) and converts it into an English sentence.

The following enumeration is a high-level description of algorithm 8 used to convert OWL propositions into natural language statements:

- (1) Given a proposition (a compound premise or a lemma), we generate a parse tree of the proposition. We assume that the proposition is expressed in a manner that does not introduce the issue of precedence. This is done by using brackets to denote precedence. For example, a proposition $A \sqsubseteq B \sqsubseteq C \sqcap D$ is asserted as follows: $A \sqsubseteq (B \sqsubseteq (C \sqcap D))$. This means that we assume a left-to-right association. Figure 3 illustrates an example of a parse tree whereby the nodes are the OWL atomic entities and the edges are labelled with OWL constructors.

- (2) The parse tree is then modified to remove all nodes that are labelled with an atomic entity containing \top . Hewlett et al. [27] suggests that this should be done by either “merging the node containing \top with a sister node containing a named class, or by promoting a class that the \top node is related to by an \sqsubseteq relation”.
- (3) We then modify the depth first search algorithm such that whenever a constructor is encountered, it is replaced with the appropriate English text found in Table 2. The resulting output of this is shown in Figure 3. Since the generation of a parse tree by a divide and conquer algorithm is $O(\log n)$, then the runtime complexity of algorithm 8 is $O(n \log n)$.

Algorithm 8: Translating OWL statements into a Natural Language

Data: Proof-Tree Σ
Result: Proof-Tree Σ

```

1 for node  $\pi \in \Sigma$  do
2    $\alpha \leftarrow \text{GenerateParseTree}(\pi)$ 
3    $\beta \leftarrow \text{ConvertToEnglish}(\alpha)$ 
4    $\text{Replace}(\pi, \beta, \Sigma)$ 
5 end
6 return  $\Sigma$ 

```

6.3 Presentation of an explanation

In this section, we show how algorithm 9 is used to present an explanation. It takes a natural language proof-tree generated by algorithm 8 and converts it into a step-by-step explanation for an entailment. An example of the presentation we refer to is shown in the output of Figure 12.

Algorithm 9: Generate a complete explanation

Data: English Proof-Tree Σ
Result: Sequence of statements τ

```

1  $\tau \leftarrow \emptyset$ 
2  $\psi \leftarrow \text{getRoot}(\Sigma)$ 
3 while  $\psi \neq \emptyset$  do
4    $\omega \leftarrow \psi.\text{get}(0)$ ;
5    $\psi \leftarrow \psi \setminus \psi.\text{get}(0)$ ;
6    $\tau \leftarrow \tau \cup \text{"The statement } \omega \text{ is implied because: \setminus n"}$ 
7   for childNode  $\rho \in \omega$  do
8     if  $\text{LastElement}(\rho, \omega)$  then
9        $\tau \leftarrow \tau \cup \text{"-" } \omega . \setminus n'$ 
10    else
11       $\tau \leftarrow \tau \cup \text{"-" } \omega \text{ and } \setminus n'$ 
12    end
13     $\psi \leftarrow \psi \cap \rho$ 
14  end
15 end
16 return  $\tau$ 

```

The explanations generated by our framework are **top-down** explanations. In summary, algorithm 9 traverses the proof tree

generated by algorithm 8 in a pre-order fashion. Whenever the algorithm visits a node, it prints the contents of the node as a statement and we indicate that this statement is implied because of the children of the pertinent node. We repeat this until we no longer have intermediate nodes. The runtime complexity of algorithm 9 is $O(n^2)$.

7 CONCLUSIONS

This paper has successfully provided a framework for generating explanations for entailments in $\mathcal{ALC} \mathcal{DL}$ ontology. Given the limited nature of this project, we restricted this framework to $\mathcal{ALC} \mathcal{DL}$. However, this framework can be extended to accommodate OWL ontologies.

There are multiple limitations of this framework. This includes the lack of optimisation of the algorithms presented in this paper. Another limitation is that of converting an explanation in \mathcal{ALC} Description Logics to English.

English is itself a complex language that is sometimes ambiguous, and when expressing complex propositions we cannot guarantee that these will be unambiguous which may lead to incorrect interpretation of explanations. Although this framework allows one to generate explanations, it does not assess whether these explanations are understood by users. We also observe that the prominent runtime and space complexity bottleneck of this framework is in generating justifications as illustrated in algorithm 1 and 5.

The main contribution of this paper is to curate existing state of the art explanation generation frameworks and present it in a summarised form. This allows novice domain experts and ontology engineers to build explanation generation tools from the ground up for black-box logic-based reasoning systems.

8 FUTURE WORK

The sister project [25] of this paper focused on improving annotations in OWL. In the future, we will investigate how to improve this framework by allowing the use of annotations.

Another investigation which we would like to conduct is the comparison of a black-box explanation generation facility versus a glass-box explanation generation facility. We would also investigate if the two could be integrated to form a hybrid explanation generation facility that perform better than glass-box and black-box explanation generation tools. Lastly, we would like to conduct an empirical evaluation of this framework when applied to an existing ontology.

9 ACKNOWLEDGEMENTS

First and foremost, I would like to thank my project supervisor Prof. Thomas Meyer and my second reader Prof. Sonia Berman for their guidance throughout the project. I would also like to thank Victoria Chama for providing me with resources that help me understand explanation facilities and Cilliers Pretorius for being a supportive project partner.

REFERENCES

- [1] 2019. Description logic. (Jul 2019). https://en.wikipedia.org/wiki/Description_logic
- [2] Franz Baader and Bernhard Hollunder. 1995. Embedding Defaults into Terminological Knowledge Representation Formalisms. In *Journal of Automated Reasoning*. Morgan Kaufmann, 306–317.
- [3] Franz Baader, Rafael Peñaloza, and Boontawe Sontisrivaraporn. 2007. Pinpointing in the Description Logic EL. https://doi.org/10.1007/978-3-540-74565-5_7
- [4] Franz Baader and Boontawe Sontisrivaraporn. 2008. Debugging Snomed ct Using Axiom Pinpointing in the Description Logic EL+, Vol. 410.
- [5] William Clancey. 1983. The Advantages of Abstract Control Knowledge in Expert System Design. [No source information available], 74–78.
- [6] Claudia Jimenez Dominguez. 1990. Sur l'explication dans les systemes a base de regles : le systeme prose.
- [7] Shaker El-Sappagh, Francesco Franda, Farman Ali, and Kyung Kwak. 2018. SNOMED CT standard ontology based on the ontology for general medical science. *BMC Medical Informatics and Decision Making* 18 (12 2018). <https://doi.org/10.1186/s12911-018-0651-5>
- [8] Jennifer Golbeck, Gilberto Fragoso, Frank W. Hartel, James A. Hendler, Jim Oberthaler, and Bijan Parsia. 2003. The National Cancer Institute's Thésaurus and Ontology. *J. Web Semant.* 1 (2003), 75–80.
- [9] OWL Working Group. 2019. Web Ontology Language (OWL). (2019). <https://www.w3.org/OWL/>
- [10] Christian Halaschek-Wiener and Yarden Katz. 2006. Belief Base Revision For Expressive Description Logics. *CEUR Workshop Proceedings* 216.
- [11] Matthew Horridge. 2011. Justification based explanation in ontologies.
- [12] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. 2008. Explanation of OWL Entailments in Protégé 4. In *International Semantic Web Conference*.
- [13] Matthew Horridge, Bijan Parsia, and Uli Sattler. 2008. Explanation of OWL Entailments in Protégé 4.
- [14] Brian J. Ellinoy. 1985. Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. *American Journal of Health-System Pharmacy* 42 (04 1985), 958–958. <https://doi.org/10.1093/ajhp/42.4.958>
- [15] Aditya Kalyanpur. 2006. *Debugging and Repair of Owl Ontologies*. Ph.D. Dissertation. College Park, MD, USA. Advisor(s) Hendler, James. AAI3222483.
- [16] Aditya Kalyanpur, Bijan Parsia, and James A. Hendler. 2005. A Tool for Working with Web Ontologies. *Int. J. Semantic Web Inf. Syst.* 1 (2005), 36–49.
- [17] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. 2007. Finding All Justifications of OWL DL Entailments. In *ISWC/ASWC*.
- [18] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, and James Hendler. 2006. Swoop: A Web Ontology Editing Browser. *Web Semantics: Science, Services and Agents on the World Wide Web* 4 (06 2006), 144–153. <https://doi.org/10.1016/j.websem.2005.10.001>
- [19] Holger Knublauch, Ray W. Ferguson, Natasha Noy, and Mark Musen. 2004. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. *Third International Semantic Web Conference* 3298, 229–243. https://doi.org/10.1007/978-3-540-30475-3_17
- [20] Solomon Malesa. 2019. An Overview of Explanations in Reasoning Systems. (2019).
- [21] Johanna Moore and William Swartout. 1989. Explanation in expert systems: A survey. (01 1989).
- [22] Sasikumar Mukundan, Srinivasan Ramani, S Muthu Raman, KSR Anjaneyulu, and Raman Chandrasekar. 2007. A Practical Introduction to Rule Based Expert Systems. (01 2007).
- [23] Tu Nguyen. 2013. Generating natural language explanations for entailments in ontologies.
- [24] Bijan Parsia and Ulrike Sattler. 2015. The OWL Explanation Workbench : A toolkit for working with justifications for entailments in OWL ontologies.
- [25] Cilliers Pretorius. 2019. Improved Explanations in the Protégé OWL Ontology Editor. (2019). <https://github.com/Pietersielie/Explanation-Workbench-More-Readable-Extension>
- [26] Frank Puppe. 1993. *Use and Usability of Expert Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 17–25. https://doi.org/10.1007/978-3-642-77971-8_3
- [27] Raymond Reiter. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32, 1 (1987), 57 – 95. [https://doi.org/10.1016/0004-3702\(87\)90062-2](https://doi.org/10.1016/0004-3702(87)90062-2)
- [28] Stefan Schlobach and Ronald Cornet. 2003. Non-standard Reasoning Services for the Debugging of Description Logic Terminologies. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 355–360. <http://dl.acm.org/citation.cfm?id=1630659.1630712>
- [29] David Schlossberg and Rafik Samuel. 2017. *NEO-FRADIN, NEOMYCIN (Neomycin)*. 259–260. <https://doi.org/10.1002/9781119220787.ch119>
- [30] Joey Sik Chun Lam, Wamberto Weber Vasconcelos, Frank Guerin, David Corsar, Alison Chorley, Timothy Norman, Javier Vázquez-Salceda, Sofia Panagiotidi, Roberto Confalonieri, I Gomez, Soraya Hidalgo, Sergio Álvarez Napa-gao, Juan Carlos Nieves, M Palau Roig, Luigi Ceccaroni, Huib Aldewereld, Virginia Dignum, Frank Dignum, Loris Penserini, and Kees Nieuwenhuis. 2009.

ALIVE: A Framework for Flexible and Adaptive Service Coordination. 236–239. https://doi.org/10.1007/978-3-642-10203-5_21

- [31] Kent Spackman, Keith Campbell, and RA Cote. 1997. SNOMED RT: a reference terminology for health care. *Proceedings : a conference of the American Medical Informatics Association / ... AMIA Annual Fall Symposium. AMIA Fall Symposium* 4 (02 1997), 640–4.
- [32] William Swartout. 1977. A Digitalis Therapy Advisor with Explanations. *Proceedings IJCAI* 5, 819–825.
- [33] William Swartout. 1983. XPLAIN: a System for Creating and Explaining Expert Consulting Programs. *Artificial Intelligence* 21 (08 1983), 285–325. [https://doi.org/10.1016/S0004-3702\(83\)80014-9](https://doi.org/10.1016/S0004-3702(83)80014-9)
- [34] William Swartout and Johanna Moore. 1993. Explanation in Second Generation Expert Systems. 543–585. https://doi.org/10.1007/978-3-642-77927-5_24
- [35] Richard Booth Thomas Meyer, Kevin Lee and Jeff Z. Pan. 2006. Finding Maximally Satisfiable Terminologies for the Description Logic ALC. *Proceedings of the National Conference on Artificial Intelligence* 1.
- [36] Nava Tintarev and Judith Masthoff. 2007. A Survey of Explanations in Recommender Systems. *IEEE 23rd International Conference on Data Engineering Workshop*, 801–810. <https://doi.org/10.1109/ICDEW.2007.4401070>
- [37] Paul Piwek Sandra Williams Tu Anh T. Nguyen, Richard Power. 2013. Predicting the Understandability of OWL Inferences. In *Extended Semantic Web Conference (ESWC 2013)*, Vol. 7882. 109–123.
- [38] Michael Wick and William B. Thompson. 1992. Reconstructive Expert System Explanation. *Artif. Intell.* 54 (03 1992), 33–70. [https://doi.org/10.1016/0004-3702\(92\)90087-E](https://doi.org/10.1016/0004-3702(92)90087-E)
- [39] L Zhou, H Huang, G Qi, Y Qu, and Q Ji. 2011. An algorithm for calculating minimal unsatisfiability-preserving subsets of ontology in DL-Lite. *Jisuanji Yanjiu yu Fazhan/Computer Research and Development* 48 (12 2011), 2334–2342.

10 SUPPLEMENTARY INFORMATION

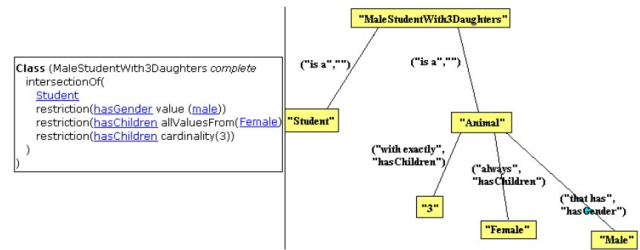


Figure 3: An example Parse Tree for OWL Class MaleStudentWith3Daughters

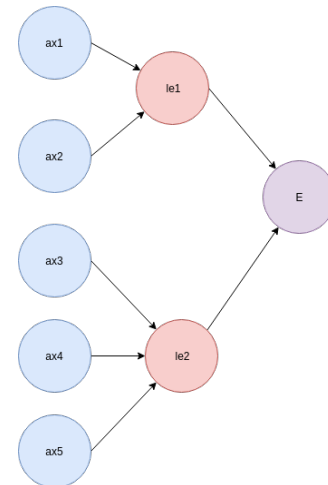


Figure 4: Structure of an explanation

For the purposes of this paper, Axiom is synonymous with premise/proposition

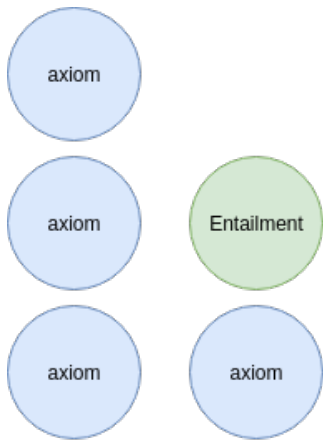


Figure 5: An input of premises and an entailment

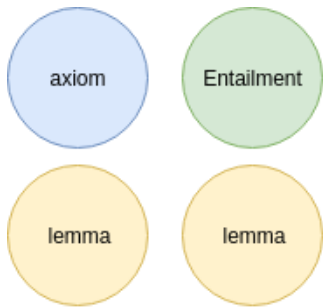


Figure 6: An input of premises and an entailment

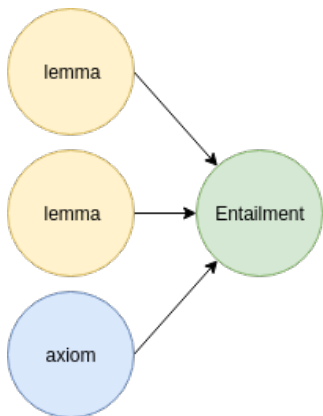


Figure 7: Initial proof tree

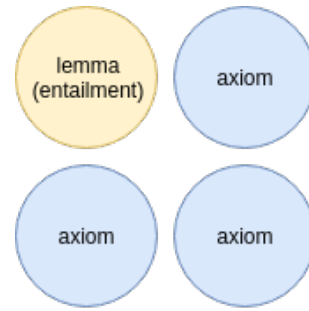


Figure 8: Justification β

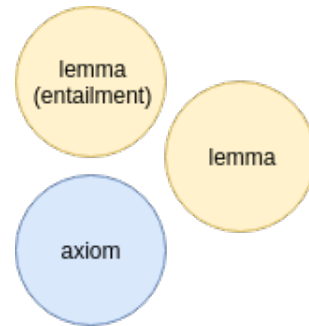


Figure 9: Justification γ

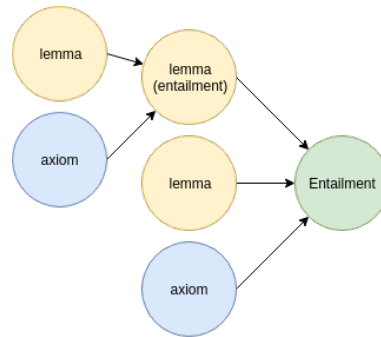


Figure 10: A proof tree Σ after inserting γ

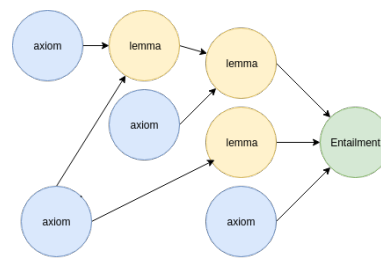


Figure 11: Final Proof Tree

Input	<p>Entailment: $Person \sqsubseteq Movie$ (Every person is a movie)</p> <p>Justification:</p> <ol style="list-style-type: none"> 1. $GoodMovie \equiv \forall hasRating.FourStarRating$ (A good movie is anything that has only four stars as ratings) 2. $Domain(hasRating, Movie)$ (Anything that has a rating is a movie) 3. $GoodMovie \sqsubseteq StarRatedMovie$ (Every good movie is a star-rated movie) 4. $StarRatedMovie \sqsubseteq Movie$ (Every star-rated movie is a movie)
Output	<p>Explanation:</p> <p>The statement “Every person is a movie” follows because the ontology implies that “Everything is a movie” (a).</p> <p>Statement (a) follows because:</p> <ul style="list-style-type: none"> - anything that has as rating something is a movie (from axiom 2), and - everything that has no rating at all is a movie (b). <p>Statement (b) follows because:</p> <ul style="list-style-type: none"> - everything that has no rating at all is a good movie (c), and - every good movie is a movie (d). <p>Statement (c) follows because axiom 1 in the justification actually means that “a good movie is anything that has no rating at all, or has only four stars as ratings”.</p> <p>Statement (d) follows because:</p> <ul style="list-style-type: none"> - every good movie is a star rated movie (from axiom 3), and - every star rated movie is a movie (from axiom 4).

Figure 12: Example results obtained when applying the framework.