UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Project Final Paper 2023

Title: Defeasible Entailment and Explanations

Author: Chipo Hamayobe

Project Abbreviation: DEE

Supervisor(s): Professor Thomas Meyer

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	5
Theoretical Analysis	*0*	*25*	5
Experiment Design and Execution	*0*	*20*	5
System Development and Implementation	*0*	*20*	20
Results, Findings and Conclusions	*10*	*20*	10
Aim Formulation and Background Work	*10*	*15*	15
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	
Total marks		**80**	

Defeasible Entailment and Explanations

Chipo Hamayobe
chipo@cs.uct.ac.za
Department of Computer Science
University of Cape Town
Cape Town, South Africa

ABSTRACT

In addition to enabling users to capture information about the world and engage in logical analysis, another crucial aspect provided by knowledge representation and reasoning tools is the inclusion of explanation capabilities. These representations serve a significant purpose by helping users understand entailments, extracting implied knowledge, and making it explicit through logical deductions. By acknowledging and appreciating the intricate nature of results produced by the KLM-style defeasible reasoning, this paper seeks to explore explanations by identifying the critical gap in the current discourse and, more importantly, the absence of comprehensive explanation facilities for these complex outcomes. We argue for an explanation framework based on justifications to clarify inferences from defeasible reasoning. Within this context, the algorithms discussed for Rational Closure and justification work effectively and are fine-tuned to efficiently generate explanations. Additionally, we provide a software system tool with both a graphical user interface and a command line interface that implements algorithms for defeasible entailment and explanations with outputs in user-friendly language. We conceive of this tool being employed as a debugging service aimed at addressing intricacies or issues within knowledge bases.

CCS CONCEPTS

• **Computing methodologies → Nonmonotonic, default reasoning and belief revision**; • **Theory of computation → Automated reasoning**.

KEYWORDS

artificial intelligence, knowledge representation and reasoning, defeasible reasoning, KLM framework, Rational Closure, defeasible justifications, defeasible explanations, defeasible debugging tool

1 INTRODUCTION

Knowledge representation and reasoning constitute a domain within the realm of Artificial Intelligence. It involves the modelling of information through formal logical frameworks, enabling the application of rule-based manipulations that pertain to specific modes of reasoning [3]. Within the diverse array of logical systems, varying levels of expressive power emerge. Classical logic, for instance, lacks the capacity to encapsulate the concept of *typicality*, posing a challenge when endeavouring to succinctly represent exceptional knowledge. This limitation becomes evident when contemplating the subsequent illustrative scenario. To demonstrate this understanding, let us consider the following example.

Example 1.1. Assume we are given the observable facts:

(1) Animals are wild
(2) Animals can hunt
(3) Pets are animals

Based on these assertions, we arrive at the deductions that *pets are wild* and *pets can hunt*. However, a paradox emerges if we introduce an additional statement,

(4) Pets are not wild

indicating that despite being able to hunt, pets are a unique category of animals that are not wild. This leads to the contradictory inference that pets are both wild and not simultaneously. As a result, our capacity to engage in logical reasoning concerning pets becomes compromised, rendering their existence untenable.

In order for this illustration to be effective, it is advisable to rephrase the first two initial facts in *Example 1.1* as:

(1) Animals typically are wild
(2) Animals typically can hunt

This wording encompasses a notion of ambiguity, enabling us to revise our inferences should we acquire information that contradicts them. Such a style of reasoning describes *defeasible reasoning*. This paper concentrates on a specific method for establishing defeasible reasoning which was proposed by Kraus, Lehmann and Magidor and appropriately recognised as the *KLM approach or framework* [12], which employs *propositional logic* as its foundation. Even though there are other formalisms for determining defeasible entailment within this framework and others, and we shall mention them in brief, we only focus on one known as *Rational Closure*, the most conservative within the *rational* family.

A vital element of the process of reasoning involves the ability to deduce fresh insights from existing information. Beyond merely deriving new knowledge, it proves highly advantageous to understand the rationale behind the derivation of specific information. Within formal logic, this understanding is facilitated through *explanations* [7]. Explanations hold significant importance as they are intrinsic to reasoning services that enhance the comprehension of knowledge bases. As a result, they assume a pivotal role in the practical utilisation of reasoning systems. In the realm of logical reasoning services, the primary means of offering explanations is via *justifications*. This paper is centred around this aspect, particularly concerning defeasible entailment justification. From *Example 1.1*, we inferred that *pets are wild* because the statements *"animals are wild"* and *"pets are animals"* provide a logically verifiable justification. This simple understanding is the basis of defeasible explanations explored in *Section 4*.

The primary aims of this paper are briefly described as follows:

- To explore and analyse the relevant theory and algorithms for KLM-style defeasible entailment and justification so as to present explanations in a user-friendly manner.
- To develop a software system tool with simple interfaces that implements the discussed algorithms so that it can be used as a debugging service for complex knowledge bases.
- To document and explain the inner workings of the developed software system tool so that other interested researchers could improve on it and/or modify it for other defeasible reasoning algorithms or usage scenarios.

Following this initial overview, *Section 2* establishes the essential background in classical logics necessary for this paper's context. *Section 3* provides an exploratory view of the KLM style of defeasible entailment, encompassing its distinctive attributes and the algorithms facilitating the computation of a variant of defeasible justification and the ensuing explanations discussed in *Section 4*. In *Section 5*, the requirements analysis, design, architecture and functionalities of the developed software system tool will be presented and explained. Subsequent sections highlight the related work, conclusions and the potential future works that could be undertaken based on this and other referenced literature as a basis.

It should be mentioned at this point that some of the literature discussed and the analysis of the software system tool may overlap with that presented by Morule [15] since the foundational basis for both forms of entailment and explanations is rooted in classical propositional logic.

2 BACKGROUND

2.1 Propositional Logic

Propositional logic finds its foundation in entities referred to as *propositional atoms*. These atoms embody fundamental statements, each of which can be attributed a truth value. Within our framework, we denote an atom that is perpetually `true` as $\top$, and one that is universally `false` is symbolised as $\bot$. The finite set of all proportional atoms is denoted as $\mathcal{P}$. By applying logical operators $(\neg, \vee, \wedge, \rightarrow, \leftrightarrow)$ to $\mathcal{P}$, we can iteratively construct more intricate statements, known as *propositional formulas* [1]. The *proportional language* $\mathcal{L}$ is the set of all such well-formed formulas. Lowercase Greek letters $\alpha, \beta, \gamma, \ldots$ are employed to represent formulas hence formally, $\gamma ::= \bot \mid \top \mid q \mid \neg\alpha \mid \alpha \vee \beta \mid \alpha \wedge \beta \mid \alpha \rightarrow \beta \mid \alpha \leftrightarrow \beta$. The operators in the final five combinations derive their semantics from an intuitive analogy with corresponding natural language expressions: *"not α", "α or β", "α and β", "if α then β"* and *"α if and only if β"*. In cases where statements take the form of $\alpha \rightarrow \beta$, α is denoted as the *antecedent*, while β is termed the *consequent*.

Meaning is attributed to propositional atoms through *interpretations* or *evaluations*. An evaluation functions by assigning a truth value to each atom, $\mathcal{P} \rightarrow \{T, F\}$, thereby presenting a distinct representation of the reality encapsulated by these atoms. If an atom p or a formula α is found to be `true` in a particular evaluation, we say that the evaluation *satisfies* p or α and *satisfaction* is denoted by $\Vdash$. α is therefore considered *unsatisfiable* if and only if $\neg\alpha = \top$. When a formula $\alpha \in \mathcal{L}$ aligns with the truth values and semantic rules of the operators within an interpretation $\mathcal{I}$, we classify $\mathcal{I}$ as a *model* of α, denoted $Mod(\alpha)$. We formally symbolically represent this as $\mathcal{I} \models \alpha$, meaning that α is *satisfied* by the interpretation $\mathcal{I}$, its model [11].

2.2 Entailment

A collection of finite propositional formulas is termed a *knowledge base*, $\mathcal{K} \subseteq \mathcal{L}$. An interpretation becomes a model of a knowledge base $\mathcal{K}$ if it functions as a model for every formula within $\mathcal{K}$, that is, for every model of $\mathcal{K}$, a formula $\alpha \in \mathcal{L}$ holds `true`. We assert that $\mathcal{K}$ *entails* a statement α, denoted as $\mathcal{K} \models \alpha$, when every model of $\mathcal{K}$ also serves as a model for α, that is, $Mod(\mathcal{K}) \subseteq Mod(\alpha)$ [11].

Definition 2.1. Let $\mathcal{F} \subseteq \mathcal{L}$ be a set of formulas and α a formula. $\mathcal{F}$ entails α, denoted $\mathcal{F} \models \alpha$, if and only if every model of $\mathcal{F}$ is a model of α [18].

This type of propositional implication operates within the framework of *classical reasoning*, a methodology for drawing conclusions based on provided information. Classical reasoning adheres to *monotonicity*, signifying that the introduction of new statements to $\mathcal{K}$ must align with and not contradict any prior inferences drawn from $\mathcal{K}$. This forms the foundational principle of a straightforward reasoning system [18], which we illustrate through the subsequent example.

Example 2.2. The assertions in *Example 1.1* involve the atoms *"animal" (a), "wild" (w), "hunt" (h)* and *"pet" (p)*, expressed as $\mathcal{P} \rightarrow \{a, w, h, p\}$, which can iteratively construct formulas, such as these in the knowledge base $\mathcal{K}$:

(1) Animals are wild $(a \rightarrow w)$
(2) Animals can hunt $(a \rightarrow h)$
(3) Pets are animals $(p \rightarrow a)$

In order to ascertain whether we can deduce the statements *"pets are wild ($p \rightarrow w$)"* or *"pets can hunt ($p \rightarrow h$)"*, we assess whether the knowledge base $\mathcal{K}$ entails either *pets are wild* or *pets can hunt*, denoted $\mathcal{K} \models (p \rightarrow w)$ and $\mathcal{K} \models (p \rightarrow h)$ respectively.

However, an issue arises when contradictory statements are introduced to $\mathcal{K}$, resulting in the absence of any models for $\mathcal{K}$. Consequently, any statement would be valid in all models of $\mathcal{K}$. This renders such a knowledge base devoid of meaning, as it entails all possible statements [18]. Hence, a framework for *non-monotonic reasoning* becomes necessary. In this regard, our preferred approach, detailed in this paper, is a form of defeasible reasoning known as the *KLM Approach*.

The task of verifying whether $\mathcal{K} \models \alpha$ can be simplified by assessing the *satisfiability* (the process of determining if there is a way to assign truth values to the constituent atoms such that the formula is `true`) of $\mathcal{K}$ when $\neg\alpha$ is included in it (expressed as $\mathcal{K} \cup \{\neg\alpha\}$). The determination of satisfiability can be performed using a solver for propositional logic satisfiability, *SAT Solver* in short [1].

2.3 Explanations

Within reasoning systems, *explanations* serve the purpose of clarifying the relevant statements within the knowledge base $\mathcal{K}$ that contribute to the logical implication between the said $\mathcal{K}$ and an entailed statement, a *query* [7]. Among the various methods for

[1]A *SAT Solver* is a software tool designed to ascertain whether an interpretation exists that satisfies a given propositional logic formula.

explanations in classical logics, *justifications* have been extensively studied. A justification $\mathcal{J}$, a subset of $\mathcal{K}$, for the logical implication of a formula β consists of a minimal subset of $\mathcal{K}$ that results in the implication of β. The collection of all justifications for the condition $\mathcal{K} \models \beta$ is symbolised as $\mathcal{J}(\mathcal{K}, \beta)$.

Definition 2.3. Let $\mathcal{K}$ be a knowledge base and β be a query such that $\mathcal{K} \models \beta$. The set of formulas $\mathcal{J}$ is a justification for $\mathcal{K} \models \beta$ if $\mathcal{J} \subseteq \mathcal{K}$, $\mathcal{J} \models \beta$ and for all $\mathcal{J}' \subseteq \mathcal{J}$, it holds that $\mathcal{J}' \not\models \beta$ [18].

It is important to note that a single query may have multiple justifications within $\mathcal{K}$. These justifications offer a straightforward and instinctive means of succinctly explaining why a logical implication is valid. They achieve this by pinpointing the specific formulas within $\mathcal{K}$ that underlie the given logical implication. To better illustrate this concept, consider the following example.

Example 2.4. From *Example 1.1*, does the classical knowledge base entail *pets are wild*, that is, $\mathcal{K} \models p \to w$?

$$\mathcal{K} = \{\textit{animals are wild, animals can hunt, pets are animals}\}$$
$$\text{or}$$
$$\mathcal{K} = \{a \to w, a \to h, p \to a\}$$

It can be established that, within context, because $Mod(\mathcal{K}) \subseteq Mod(p \to w)$, then $\mathcal{K} \models p \to w$. This conclusion is supported by the justification $\mathcal{J} = \{a \to w, p \to a\}$, as the set $\{a \to w, p \to a\} \models p \to w$, and the removal of any statement from this set, $\mathcal{J}$, results in the entailment becoming invalid.

Numerous algorithms have been created to calculate classical justifications [10]. These algorithms are categorised as either black-box or glass-box methods. For this purpose, we employ a black-box algorithm to determine all justifications for a classical $\mathcal{K}$. In these algorithms, the computation of justifications occurs independently of the underlying reasoning process. This characteristic grants them versatility across various logical frameworks [16], which makes them suitable for use with propositional logic, as we do in this paper.

3 DEFEASIBLE ENTAILMENT

The approach to non-monotonic reasoning explored in this paper is that of the *preferential approach*, which was first defined by Shoham [11]. Shoham defined the class of preferential logics, achieved by enriching regular semantics with a preference relation over all valuations. Then, for a formula α to be satisfiable, it must be satisfied by all the most preferred models of α. This set of most preferred models is defined as being all models of α that are minimal with respect to the preference relation [7]. Using the preferential semantics, notions of preferential entailment can be defined by specifying that $\alpha \mathrel{|\!\approx} \beta$, read as *"α preferentially entails β"*, if and only if all preferred models of α also satisfy β. A foundation for a corresponding proof-theoretic system for preferential reasoning was also defined by Gabbay [11]. Both the semantics and corresponding proof system were then extended by Kraus et al. to form the KLM Approach [12]. Rational Closure, an alternative syntactic definition of minimal ranked defeasible entailment, is the focus of this paper and is discussed in *Section 3.4.*

3.1 The KLM Approach

While various methodologies for defeasible reasoning exist, a prominently examined approach in the literature is the KLM Framework. This approach expands propositional logic by incorporating the notion of *defeasible implication*, $\mid\sim$, serving as a counterpart to the classical implication $\to$. Defeasible implications are represented as $\alpha \mid\sim \beta$, where $\alpha, \beta \in \mathcal{L}$, and this notation is interpreted as *"α typically implies β"*. Nested defeasible implication operations such as $(\alpha \mid\sim \beta) \mid\sim \gamma$ are not permitted.

A *defeasible knowledge base* is a finite collection of formulas comprising one or more defeasible implications. Defeasible entailment, $\mathrel{|\!\approx}$, is then established as a binary relationship that spans defeasible knowledge bases and defeasible implications. For instance, $\mathcal{K} \mathrel{|\!\approx} \alpha \mid\sim \beta$ signifies that within the scope of $\mathcal{K}$, *"$\mathcal{K}$ defeasibly entails that α typically implies β"*. Notably, while our supposition revolves around the inclusion of solely defeasible implications within defeasible knowledge bases, we possess the means to represent any classical formula α through the defeasible form $\neg\alpha \mid\sim \bot$ [8].

Lehmann and Magidor present a set of postulates that establish the foundation of *rational* defeasible entailment [14]. Each postulate serves as an assertion of an intuitively expected characteristic inherent to a reasonable defeasible entailment relationship, thus earning the moniker *"rational"*. In parallel to this axiomatic definition, rational entailment relationships possess a model-theoretic semantics, which, while beyond the scope of this paper and discussion, are sometimes precisely described by computational reasoning algorithms of practical complexity [8]. These reasoning algorithms constitute a pivotal focal point of our discussion and we emphasise Rational Closure, the most prominent variant of defeasible entailment within the KLM framework, denoted $\mathrel{|\!\approx}_{RC}$. We define a partial ordering, $\leq_{\mathcal{K}}$, over the ranked interpretations such that this ordering favours interpretations that are more *conservative* or *typical conservative* over those that are less typical.

Definition 3.1. If $\mathcal{K}$ is a knowledge base and $\mathcal{R}^{\mathcal{K}}$ the set of its ranked models, it holds for every $\mathcal{R}_0^{\mathcal{K}}, \mathcal{R}_1^{\mathcal{K}} \in \mathcal{R}^{\mathcal{K}}$ that $\mathcal{R}_0^{\mathcal{K}} \leq_{\mathcal{K}} \mathcal{R}_1^{\mathcal{K}} \iff$ for every $u \in \mathcal{U}$, $\mathcal{R}_0^{\mathcal{K}}(u) \leq \mathcal{R}_1^{\mathcal{K}}(u)$ [11].

All defeasible entailment discussions in the paper should be assumed to be $\mathrel{|\!\approx}_{RC}$, even when not explicitly stated or $\mathrel{|\!\approx}$ is used.

3.2 The KLM Properties

In contrast to classical entailment, the concept of defeasible entailment is not singular in nature. Instead, there are several distinct representations of defeasible entailment, such as *Rational Closure* [14], *Lexicographic Closure* [13] and *Relevant Closure* [6]. The KLM framework establishes a set of *rationality* principles, which have been asserted by Lehmann and Magidor as necessary criteria for any defeasible entailment approach [14]. These principles provide a way to differentiate between approaches that are considered suitable or unsuitable in the context of rational entailment.

When a defeasible entailment algorithm meets all the specified criteria, it is considered an appropriate variant of defeasible entailment and is referred to as *LM-rational.* The provided KLM properties for propositional logic are outlined as follows:

(1) `Reflexivity (Ref)`: $\mathcal{K} \mathrel{|\!\approx} \alpha \mid\sim \alpha$
(2) `Left Logical Equivalence (LLE)`: $\dfrac{\alpha \equiv \beta,\ \mathcal{K} \mathrel{|\!\approx} \alpha \mid\sim \gamma}{\mathcal{K} \mathrel{|\!\approx} \beta \mid\sim \gamma}$

(3) `Right Weakening (RW)`: $\dfrac{\mathcal{K} \models\!\approx \alpha \mid\!\sim \beta,\ \beta \models \gamma}{\mathcal{K} \models\!\approx \alpha \mid\!\sim \gamma}$

(4) `And`: $\dfrac{\mathcal{K} \models\!\approx \alpha \mid\!\sim \beta,\ \mathcal{K} \models\!\approx \alpha \mid\!\sim \gamma}{\mathcal{K} \models\!\approx \alpha \mid\!\sim \beta \wedge \gamma}$

(5) `Or`: $\dfrac{\mathcal{K} \models\!\approx \alpha \mid\!\sim \gamma,\ \mathcal{K} \models\!\approx \beta \mid\!\sim \gamma}{\mathcal{K} \models\!\approx \alpha \vee \beta \mid\!\sim \gamma}$

(6) `Cautious Monotonicity (CM)`: $\dfrac{\mathcal{K} \models\!\approx \alpha \mid\!\sim \beta,\ \mathcal{K} \models\!\approx \alpha \mid\!\sim \gamma}{\mathcal{K} \models\!\approx \alpha \wedge \beta \mid\!\sim \gamma}$

(7) `Rational Monotonicity (RM)`: $\dfrac{\mathcal{K} \models\!\approx \alpha \mid\!\sim \gamma,\ \mathcal{K} \not\models\!\approx \alpha \mid\!\sim \neg\beta}{\mathcal{K} \models\!\approx \alpha \wedge \beta \mid\!\sim \gamma}$

Each of these characteristics holds a relatively straightforward interpretation. Kaliski [11] offers an elaborate explanation of the intended significance of each of these properties. Let us contemplate the subsequent pair of defeasible implications that can be expressed using propositional logic as an example:

- *If Salah plays, then typically Liverpool FC wins ($s \mid\!\sim w$)*
- *If Mané plays, then typically Liverpool FC wins ($m \mid\!\sim w$)*

The concept of `Or` indicates that if a specific formula can be reasonably inferred from two or more distinct formulas, then it should also be a reasonable inference from either combination of those formulas using disjunction. It appears logical to deduce the following statement as it aligns precisely with the functioning of the `Or` property.

- *If Salah or Mané plays, then typically Liverpool FC wins ($s \vee m \mid\!\sim w$)*

3.3 Materialisation and Dematerialisation

The propositional formula $\alpha \rightarrow \beta$ corresponds to the *material counterpart* of a defeasible implication $\alpha \mid\!\sim \beta$. Alternatively, $\alpha \rightarrow \beta$ can be seen as the *materialisation* of $\alpha \mid\!\sim \beta$. We can deduce that the defeasible implication $\alpha \mid\!\sim \beta$ is the *dematerialisation* representation of the classical implication $\alpha \rightarrow \beta$ [11].

Definition 3.2. The materialisation of a defeasible knowledge base $\mathcal{K}$, denoted as $\overrightarrow{\mathcal{K}}$, signifies the conventional knowledge base:

$$\{\alpha \rightarrow \beta \mid \alpha \mid\!\sim \beta \in \mathcal{K}\}$$

Definition 3.3. The dematerialisation of a classical knowledge base $\mathcal{K}$, represented as $\underrightarrow{\mathcal{K}}$, is the knowledge base where each formula in $\mathcal{K}$, follows the structure $\alpha \rightarrow \beta$, where $\alpha \in \mathcal{L}, \beta \in \mathcal{L}$:

$$\{\alpha \mid\!\sim \beta \mid \alpha \rightarrow \beta \in \mathcal{K}\}$$

By employing the concept of knowledge base materialisation, we can establish a way to determine the uniqueness of a propositional formula concerning a knowledge base.

Definition 3.4. A propositional formula α, where $\alpha \in \mathcal{L}$, is deemed exceptional for a knowledge base $\mathcal{K}$ if $\mathcal{K} \models \neg\alpha$.

To illustrate this formulation, consider $\mathcal{K} = \{a \mid\!\sim w, p \mid\!\sim a, p \mid\!\sim \neg w\}$. In this case, p is more specific and is an exceptional element for $\mathcal{K}$, but a is not. This example demonstrates the inherent significance of exceptionality in a straightforward manner: We can only deduce implications of exceptional statements by dismissing the more general formulas present in $\mathcal{K}$.

The method for explaining defeasible entailments relies on the approach used for justifying classical entailments discussed in *Sections 2.2 and 2.3.* The classical entailment justification method takes as input, the materialised knowledge base $\mathcal{K}$. This means that the formulas found in the generated justifications are parts of the materialised knowledge base. In simpler terms, any justification, $\mathcal{J} \subseteq (\overrightarrow{\mathcal{R}_0} \cup \overrightarrow{\mathcal{R}_1} \cup \cdots \cup \overrightarrow{\mathcal{R}_\infty})$, that comes from the outcome of the classical justification process, can not be directly used for explaining defeasible matters unless the essential formulas are reinstated from their initial forms. This underlines the importance of the `Dematerialisation` algorithm (*Algorithm 1*). The algorithm goes through every justification $j \in \mathcal{J}$ and for every j, it examines each formula within it and if any formula, $\alpha \rightarrow \beta$, is not present in the initial knowledge base, it is modified to become the defeasible implication $\alpha \mid\!\sim \beta$ [18].

Algorithm 1: `Dematerialisation`

```
Input: Set of justifications $\mathcal{J}$ and a defeasible knowledge base $\mathcal{K}$
Output: Set of dematerialised justifications $\mathcal{J}$
1 for $j$ in $\mathcal{J}$ do
2     for $\alpha \rightarrow \beta$ in $j$ do
3         if $\alpha \rightarrow \beta \in \mathcal{K}$ then
4             Replace $\alpha \rightarrow \beta$ with $\alpha \mid\!\sim \beta$;
5         end
6     end
7 end
8 return $\mathcal{J}$;
```

3.4 Rational Closure

Rational Closure represents the most cautious approach to deriving conclusions from a defeasible knowledge base $\mathcal{K}$. This form of entailment draws limited inferences from the given $\mathcal{K}$. As outlined by Casini et al., this concept can be defined both algorithmically and semantically [6]. The semantic definition relies on structures called *ranked interpretations.* These interpretations essentially involve ordering all possible understandings in the set of all interpretations, $\mathcal{I}$, by their usualness. In this paper, we only consider the algorithmic definition.

Freund [9] introduced an algorithm to prioritise formulas necessary for Rational Closure. This distinct prioritisation is termed the *base rank* for formulas. Notably, the base rank for formulas within a knowledge base holds singularity. The rationale underlying the base rank is rooted in the notion that lower-ranked formulas are comparatively less extraordinary. Casini et al. subsequently formalised both the `BaseRank` (*Algorithm 2*) and `RationalClosure` (*Algorithm 3*) algorithms respectively [6].

The `BaseRank` procedure arranges all statements within $\overrightarrow{\mathcal{K}}$ based on their level of generality. This ranking results in statements that are always `true` (classical statements) being positioned at the lowest rank (the *infinite rank,* $\mathcal{R}_\infty$). The higher ranks contain materialisations of defeasible statements, which progressively increase in generality as one ascends the ranking order towards the top most, $\mathcal{R}_0$. We define a partial ordering, $\leq_{\mathcal{K}}$, over the ranked interpretations such that this ordering favours interpretations that are more *conservative* or *typical* over those that are less typical. It is worth noting that the minimal ranked entailment of a knowledge base $\mathcal{K}$ is essentially the same thing as the Rational Closure of $\mathcal{K}$.

Algorithm 2: BaseRank

```
Input: A knowledge base $\mathcal{K}$
Output: An ordered tuple $(\mathcal{R}_0, \dots, \mathcal{R}_{n-1}, \mathcal{R}_\infty, n)$
1  $i := 0$;
2  $E_0 := \overrightarrow{\mathcal{K}}$;
3  while $E_{i-1} \neq E_i$ do
4      $E_{i+1} := \{\alpha \to \beta \in E_i \mid E_i \models \neg\alpha\}$;
5      $\mathcal{R}_i := E_i \setminus E_{i+1}$;
6      $i := i + 1$;
7  end
8  $\mathcal{R}_\infty := E_{i-1}$;
9  if $E_{i-1} = \emptyset$ then
10     $n := i - 1$;
11 end
12 else
13     $n := i$;
14 end
15 return $(\mathcal{R}_0, \dots, \mathcal{R}_{n-1}, \mathcal{R}_\infty, n)$
```

Definition 3.5. If $\mathcal{K}$ is a knowledge base and $\mathcal{R}^{\mathcal{K}}$ the set of its ranked models, it holds for every $\mathcal{R}_0^{\mathcal{K}}, \mathcal{R}_1^{\mathcal{K}} \in \mathcal{R}^{\mathcal{K}}$ that $\mathcal{R}_0^{\mathcal{K}} \leq_{\mathcal{K}} \mathcal{R}_1^{\mathcal{K}} \iff$ for every $u \in \mathcal{U}$, $\mathcal{R}_0^{\mathcal{K}}(u) \leq \mathcal{R}_1^{\mathcal{K}}(u)$ [11].

Given a knowledge base $\mathcal{K}$, because the BaseRank algorithm assumes $\mathcal{K}$ contains only defeasible implications, Wang [18] examined the following two situations and made some observations:

(1) Suppose we have a classical implication β and $\beta \in \mathcal{K}$. As per the definition of defeasible implication within $\mathcal{K}$, we can substitute β with $\neg\beta \mid\!\sim \bot$. If we input $\mathcal{K}$ into the BaseRank algorithm, then $\neg\beta \to \bot \in E_0$, and for some level i, $\neg\beta \to \bot \in E_i$. Since the classical implication $\neg\beta \to \bot$ essentially implies β, line 4 of the algorithm places $\neg\beta \to \bot$ in E_{i+1}. This realisation leads us to the understanding that $\neg\beta \to \bot$ is consistently put into $\mathcal{R}_\infty$. As a result, we can conclude that the BaseRank algorithm always designates any classical implication β to the lowest rank, $\mathcal{R}_\infty$.
(2) Suppose we have a knowledge base $\mathcal{K}$, and in this context, let us consider the case where $\{\alpha \mid\!\sim \beta, \alpha \mid\!\sim \neg\beta\} \subseteq \mathcal{K}$. When we use $\mathcal{K}$ as the BaseRank algorithm input, it results in $\{\alpha \to \beta, \alpha \to \neg\beta\} \subseteq E_0$, and for a specific level i, $\{\alpha \to \beta, \alpha \to \neg\beta\} \subseteq E_i$. Because E_i implies $\neg\alpha$ at all times, line 4 of the algorithm consistently places $\alpha \to \beta$ and $\alpha \to \neg\beta$ into E_{i+1}. Based on this observation, we can draw the conclusion that if both $\alpha \mid\!\sim \beta$ and $\alpha \mid\!\sim \neg\beta$ are formulas in $\mathcal{K}$, the algorithm invariably assigns them to the lowest rank, $\mathcal{R}_\infty$.

To further expand on the literal understanding, let us consider a simple practical illustration that demonstrates the underlying idea of base ranking:

Example 3.6. Consider the following knowledge base $\mathcal{K}$:

(1) Animals typically are wild ($a \mid\!\sim w$)
(2) Pets typically are animals ($p \mid\!\sim a$)
(3) Pets typically are not wild ($p \mid\!\sim \neg w$)
(4) ExoticPets typically are animals ($e \mid\!\sim a$)
(5) ExoticPets typically are pets ($e \mid\!\sim p$)

The following steps help explain the statement ranking produced by the BaseRank algorithm.

(1) $\mathcal{K} = \{a \mid\!\sim w, p \mid\!\sim a, p \mid\!\sim \neg w, e \mid\!\sim a, e \mid\!\sim p\}$
(2) The materialisation of $\mathcal{K}$ is:
$$\overrightarrow{\mathcal{K}} = \{a \to w, p \to a, p \to \neg w, e \to a, e \to p\}$$
Therefore, $E_0 = \{a \to w, p \to a, p \to \neg w, e \to a, e \to p\}$.
(3) By utilizing line 4 of the algorithm, we derive $E_1 = \{\alpha \to \beta \in E_0 \mid E_0 \models \neg\alpha\}$. Only three interpretations fulfill E_0 : $\mathsf{a\overline{p}w\overline{e}}$, $\overline{\mathsf{apwe}}$ and $\overline{\mathsf{ap}}\mathsf{w}\overline{\mathsf{e}}$. All the three interpretations also fulfil $\overline{\mathsf{p}}$ and $\overline{\mathsf{e}}$. Consequently, $E_0 \models \neg p, E_0 \models \neg e$ and $E_1 = \{p \to a, p \to \neg w, e \to a, e \to p\}$. The subsequent step, line 5 of the algorithm, defines $\mathcal{R}_1 = \{a \to w\}$.
(4) During the second iteration of the while loop, E_2 is defined as $\{\alpha \to \beta \in E_1 \mid E_1 \models \neg\alpha\}$ and the interpretations satisfying E_1 are $\mathsf{ap\overline{w}e}$, $\mathsf{a\overline{p}\overline{w}\overline{e}}$, $\mathsf{\overline{a}p\overline{w}e}$, $\overline{\mathsf{apw}}\overline{\mathsf{e}}$, $\mathsf{a}\overline{\mathsf{pwe}}$ and $\overline{\mathsf{apwe}}$. From these interpretations, it becomes evident that $E_1 \not\approx \neg p$ and $E_1 \not\approx \neg e$. As a result, $E_2 = \emptyset$ and $\mathcal{R}_1 = \{p \to a, p \to \neg w, e \to a, e \to p\}$.
(5) The third iteration of the while loop renders $E_3 = \emptyset$, leading to termination due to $E_2 = E_3$. Consequently, $\mathcal{R}_\infty = \emptyset$ in line 10. The if-else construct spanning lines 8 through 14 designates $n = 3$.
(6) Conclusively, the algorithm yields the tuple: $(\{a \to w\}, \{p \to a, p \to \neg w, e \to a, e \to p\}, \emptyset, 3)$.
(7) From a visual standpoint, the rankings of $\overrightarrow{\mathcal{K}}$ can be observed in the following table.

$\mathcal{R}_0$	$a \to w$
$\mathcal{R}_1$	$p \to a, p\neg w, e \to a, e \to p$
$\mathcal{R}_\infty$	$\emptyset$

Figure 1: BaseRank output of *Example 3.6*

Algorithm 3: RationalClosure

```
Input: A knowledge base $\mathcal{K}$ and a defeasible implication $\alpha \mid\!\sim \beta$
Output: true, if $\mathcal{K} \approx \alpha \mid\!\sim \beta$, and false otherwise
1  $(\mathcal{R}_0, \dots, \mathcal{R}_{n-1}, \mathcal{R}_\infty, n)$ := BaseRank($\mathcal{K}$);
2  $i := 0$;
3  $\mathcal{R} := \bigcup_{i=0}^{j<n} \mathcal{R}_j$;
4  while $\mathcal{R}_\infty \cup \mathcal{R} \models \neg\alpha$ and $\mathcal{R} \neq \emptyset$ do
5      $\mathcal{R} := \mathcal{R} \setminus \mathcal{R}_i$;
6      $i := i + 1$;
7  end
8  return $\mathcal{R}_\infty \cup \mathcal{R} \models \alpha \to \beta$;
```

We can now proceed from the prior exemplification of base ranking to explain the rationale underpinning Rational Closure as guided by the RationalClosure algorithm. In this instance, we shall use the defeasible implication statement $e \mid\!\sim \neg w$ as our defeasible query; that is $\mathcal{K} \approx e \mid\!\sim \neg w$?

(1) The outcome of the base ranked instance on line 1 of the algorithm is employed. The algorithm's 3rd line defines the set $\mathcal{R} = \bigcup_{i=0}^{j<n} \mathcal{R}_j$. This operation is applied to the resultant

outcome of base ranking, yielding $\mathcal{R} = \{a \rightarrow w, p \rightarrow a, p \rightarrow \neg w, e \rightarrow a, e \rightarrow p\}$.

(2) The conditions presented on line 4 of the algorithm necessitate the evaluation of $\mathcal{R}_\infty \cup \mathcal{R} \models \neg e$. Since no model encompassing $\mathcal{R}_\infty \cup \mathcal{R}$ satisfies the condition d, it follows that $\mathcal{R}_\infty \cup \mathcal{R} \models \neg e$. Line 5 involves the removal of $\mathcal{R}_0$ from $\mathcal{R}$, leading to the modified set $\mathcal{R} = \{p \rightarrow a, p \rightarrow \neg w, e \rightarrow a, e \rightarrow p\}$.
(3) The interpretation denoted as $\mathsf{ap\overline{w}e}$ satisfies the set $\mathcal{R}$ as well as the condition e. Consequently, it can be inferred that $\mathcal{R}_\infty \cup \mathcal{R} \not\models \neg e$, which prompts the algorithm to exit the while loop. To conclude, it is imperative to ascertain the validity of $\mathcal{R}_\infty \cup \mathcal{R} \models e \rightarrow \neg w$. The interpretation $\mathsf{ap\overline{w}e}$ also satisfies the condition $e \rightarrow \neg w$. This affirms the truth of $\mathcal{R}_\infty \cup \mathcal{R} \models e \rightarrow \neg w$.
(4) As a result, the algorithm yields a `true` outcome, confirming the relationship $\mathcal{K} \approx e \rightarrow \neg w$. In simpler terms, the defeasible knowledge base $\mathcal{K}$ does entail the defeasible query, that *exotic pets typically are not wild.*

3.5 Alternatives to Rational Closure

3.5.1 Lexicographic closure. Based on the framework described by Lehmann and Magidor, Lehmann introduced the concept of Lexicographic Closure as an enhancement of Rational Closure, distinguished mainly by variations in the hierarchical ranking of statements. [13] This approach involves ranking statements within the same hierarchy, following the precedence set by the `BaseRank` algorithm, while simultaneously maintaining the initial ranking. When it comes to the extent of logical inferences drawn from a knowledge base, Lexicographic Closure demonstrates a lesser degree of conservatism compared to Rational Closure [6].

3.5.2 Relevant Closure. Originally introduced by Casini et al. [5] as a non-LM-rational alternative, the concept of relevant closure offers a nuanced perspective that seeks to strike a balance between the *prototypical* accuracy of Rational Closure and the inclusion of *presumptive* inferences rooted in the concept of *relevance* associated with each statement. This positions Relevant Closure as an intermediary point between Rational Closure and Lexicographic Closure along the spectrum of typical and inferred entailments. The approach is algorithmically defined and operates by pinpointing a subset of defeasible statements pertinent to a specific query. Subsequently, it applies a modified version of the defeasible entailment algorithm to the knowledge base, ensuring that only these pertinent statements become subject to removal [8].

4 DEFEASIBLE EXPLANATIONS

Horridge introduced the concept of justifications, which represent the smallest sets in the knowledge base $\mathcal{K}$ that entails a specific query. Horridge also introduced an algorithm to identify justifications for classical entailments [10]. However, when dealing with defeasible entailments, the concept of justification becomes more intricate. Based on *Description Logics*, Chama [7] provided a distinct definition for defeasible justification, which Everett et al. [8] termed *weak justifications.* How this notion relates to *strong justifications*, a more general notion of defeasible explanation given by Brewka and Ulbricht [4], was explored and extended to KLM by Everett et al. using a revised definition. As emphasised by Wang [18], Chama's definition of justification is applicable to the explanations we are exploring in this paper.

Definition 4.1. For a justification of a defeasible entailment, $\mathcal{K} \models \alpha$, the process involves selecting only the smallest sets within $\mathcal{K}$ that meet two conditions:

(1) The materialisation process of any minimal set must classically entail $\overrightarrow{\alpha}$.
(2) The minimal sets exclusively encompass the relevant and applicable axioms and these suitable and relevant axioms should only be ones that were taken into account when calculating Rational Closure entailment.

4.1 Defeasible Justification

Let us approach the discussion of defeasible justifications through an illustrative example.

Example 4.2. Given the defeasible knowledge base $\mathcal{K}$:

(1) Animals typically are wild ($a \mid\sim w$)
(2) Animals typically can hunt ($a \mid\sim h$)
(3) Pets are animals ($p \rightarrow a$)
(4) Pets typically are not wild ($p \mid\sim \neg w$)
(5) ExoticPets are pets ($e \rightarrow p$)
(6) ExoticPets typically are wild ($e \mid\sim w$)

The `BaseRank` algorithm produces the ordering of formulas shown in *Figure 2.*

$\mathcal{R}_0$	$a \mid\sim w, a \mid\sim h$
$\mathcal{R}_1$	$p \mid\sim \neg w$
$\mathcal{R}_2$	$e \mid\sim w$
$\mathcal{R}_\infty$	$p \rightarrow a, e \rightarrow p$

Figure 2: `BaseRank` algorithm output for $\mathcal{K}$ from *Example 4.2*

Using this formula ranking output, the `RationalClosure` algorithm determines that $\mathcal{K} \approx e \mid\sim w$. Additionally, the algorithm does not consider $\mathcal{R}_0 = \{a \mid\sim w, a \mid\sim h\}$ and $\mathcal{R}_1 = \{p \mid\sim \neg w\}$. Although one might initially consider both $\mathcal{J}_1 = \{e \mid\sim w\}$ and $\mathcal{J}_2 = \{a \mid\sim w, p \mid\sim a, e \rightarrow p\}$ as explanations supporting the claim. However, $\mathcal{J}_2$ contains $a \mid\sim w$, which was not factored into the Rational Closure calculation. Consequently, only $\mathcal{J}_1$ remains as the valid defeasible explanation since it only includes the suitable and relevant formula.

The `RationalClosure` algorithm discussed in *Section 3* merely gives back a `true` or `false` answer that tells us if the defeasible entailment, $\mathcal{K} \approx \alpha \mid\sim \beta$, is valid or not. Using the idea of defeasible justification, Chama [7] made adjustments to this algorithm for Description Logics to include a number that shows how many levels of axioms were ignored. Chama's approach to defeasible justification involves excluding axiom levels following the Rational Closure algorithm [18]. This change also involves using the `ComputeAllJustifications` algorithm (*Algorithm 7 in Appendix A.1*) process on the relevant and suitable axioms in the knowledge base. This results in generating explanations for defeasible arguments.

Based on Chama's reasoning and algorithms for justifications, Wang [18] extended these principles and produced similar algorithms for propositional logics, which we employ in this paper. The modified algorithms serve the same purpose as their original counterparts, *Algorithms 2* and *3*, but are optimised for efficiency in producing explanations.

Algorithm 4: `BaseRankForJustification`

```
   Input: A knowledge base $\mathcal{K}$
   Output: An ordered tuple $(\mathcal{R}_0, \ldots, \mathcal{R}_n, \mathcal{R}_\infty)$
1  $C := \{\alpha \rightarrow \beta \in \mathcal{K}\} \cup \{\alpha \rightarrow \beta, \alpha \rightarrow \neg\beta \in \mathcal{K} \mid \alpha \mathrel{|\!\sim} \beta$ and
   $\alpha \rightarrow \neg\beta \in \mathcal{K}\}$;
2  $E_0 := \mathcal{K} \setminus C$;
3  $i := 0$;
4  while $E_{i-1} \neq E_i$ do
5      $E_{i+1} := \{\alpha \mathrel{|\!\sim} \beta \in E_i \mid \overrightarrow{E_i} \cup C \models \neg\alpha\}$;
6      $\mathcal{R}_i := E_i \setminus E_{i+1}$;
7      $i := i + 1$;
8  end
9  $\mathcal{R}_\infty := C \cup E_{i-1}$;
10 if $E_{i-1} = \emptyset$ then
11     $n := i - 1$;
12 end
13 else
14     $n := i$;
15 end
16 return $(\mathcal{R}_0, \ldots, \mathcal{R}_n, \mathcal{R}_\infty)$
```

Wang enhanced the `BaseRank` algorithm even more by introducing rules that are applied to formulas that are consistently assigned to the infinity rank, $\mathcal{R}_\infty$ [18]. This modified version necessary for defeasible explanations is presented as `BaseRankForJustification` (*Algorithm 4*). To help understand the base ranking procedure narrated by the enhanced algorithm, let us consider the following knowledge base as input to the algorithm:

$$\mathcal{K} = \{p \rightarrow a, e \rightarrow p, a \mathrel{|\!\sim} w, a \mathrel{|\!\sim} h, p \mathrel{|\!\sim} \neg w, e \mathrel{|\!\sim} w\}$$

As we proceed through the algorithm, the variables are given these values:

- $C := \{p \rightarrow a, e \rightarrow p\}$
- $E_0 := \{a \mathrel{|\!\sim} w, a \mathrel{|\!\sim} h, p \mathrel{|\!\sim} \neg w, e \mathrel{|\!\sim} w\}$
- $E_1 := \{p \mathrel{|\!\sim} \neg fly, e \mathrel{|\!\sim} w\}$
- $\mathcal{R}_0 := \{a \mathrel{|\!\sim} w, a \mathrel{|\!\sim} h\}$
- $E_2 := \{e \mathrel{|\!\sim} w\}$
- $\mathcal{R}_1 := \{a \mathrel{|\!\sim} w\}$
- $E_3 := \emptyset$
- $\mathcal{R}_2 := \{a \mathrel{|\!\sim} w\}$
- $\mathcal{R}_\infty := \{p \rightarrow a, e \rightarrow p\}$

The base ranking output for $\mathcal{K}$ is shown visually in *Figure 3*.

$\mathcal{R}_0$	$a \mathrel{|\!\sim} w, a \mathrel{|\!\sim} h$
$\mathcal{R}_1$	$p \mathrel{|\!\sim} \neg w$
$\mathcal{R}_2$	$e \mathrel{|\!\sim} w$
$\mathcal{R}_\infty$	$p \rightarrow a, e \rightarrow p$

Figure 3: `BaseRankForJustification` algorithm output for $\mathcal{K}$

Much like how Chama [7] modified the Rational Closure algorithm to include an extra integer indicating the count of formula ranks that are ignored, Wang [18] made corresponding changes to the `RationalClosure` algorithm to align it with the necessary information for KLM style defeasible justification. This modified version of the algorithm is presented as `RationalClosureForJustification` (*Algorithm 5*).

Algorithm 5: `RationalClosureForJustification`

```
   Input: A knowledge base $\mathcal{K}$ and a defeasible implication $\alpha \mathrel{|\!\sim} \beta$
   Output: true, if $\mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \beta$, and false otherwise, rank $i$ and an
           ordered tuple of base ranked formulas $(\mathcal{R}_0, \ldots, \mathcal{R}_n, \mathcal{R}_\infty)$
1  $(\mathcal{R}_0, \ldots, \mathcal{R}_n, \mathcal{R}_\infty) :=$ BaseRankForJustification($\mathcal{K}$);
2  $i := 0$;
3  $\mathcal{R} := \bigcup_{i=0}^{j \leq n} \mathcal{R}_j$;
4  while $\overrightarrow{\mathcal{R}_\infty} \cup \overrightarrow{\mathcal{R}} \models \neg\alpha$ and $\mathcal{R} \neq \emptyset$ do
5      $\mathcal{R} := \mathcal{R} \setminus \mathcal{R}_i$;
6      $i := i + 1$;
7  end
8  return $\overrightarrow{\mathcal{R}_\infty} \cup \overrightarrow{\mathcal{R}} \models \alpha \rightarrow \beta, i, (\mathcal{R}_0, \ldots, \mathcal{R}_n, \mathcal{R}_\infty)$;
```

Given a defeasible knowledge base $\mathcal{K}$ and a defeasible query $\alpha \mathrel{|\!\sim} \beta$ as inputs, the `RationalClosureForJustification` algorithm returns the following results:

(1) A `true` or `false` value indicating whether $\mathcal{K} \mathrel{|\!\approx} \alpha \mathrel{|\!\sim} \beta$.
(2) An integer representing the count of ranks of formulas, if any, that are not considered when determining the logical implication.
(3) An ordered tuple $(\mathcal{R}_0, \ldots, \mathcal{R}_n, \mathcal{R}_\infty)$ which contains the smallest ranked formulas in $\mathcal{K}$.

To help grasp the concept of Rational Closure as depicted by the `RationalClosureForJustification` algorithm, we will explain it using the following example.

$$\mathcal{K} = \{p \rightarrow a, e \rightarrow p, a \mathrel{|\!\sim} w, a \mathrel{|\!\sim} h, p \mathrel{|\!\sim} \neg w, e \mathrel{|\!\sim} w\}$$

The base rank output from `BaseRankForJustification` sub-algorithm on line 1 is as follows:

- $\mathcal{R}_0 := \{a \mathrel{|\!\sim} w, a \mathrel{|\!\sim} h\}$
- $\mathcal{R}_1 := \{p \mathrel{|\!\sim} \neg w\}$
- $\mathcal{R}_2 := \{e \mathrel{|\!\sim} w\}$
- $\mathcal{R}_\infty := \{p \rightarrow a, e \rightarrow p\}$

We provide two distinct cases of defeasible implications as illustrative defeasible input queries to the algorithm:

(1) $\mathcal{K} \mathrel{|\!\approx} p \mathrel{|\!\sim} h$?
   - $\overrightarrow{\mathcal{R}} := \{e \rightarrow w, p \rightarrow \neg w, a \rightarrow w, a \rightarrow h\}$ and $\overrightarrow{\mathcal{R}_\infty} \cup \overrightarrow{\mathcal{R}} \models \neg p$.
   - It follows that $\mathcal{R}_0$ is removed from $\mathcal{R}$, then $\mathcal{R} := \{p \rightarrow \neg w, e \rightarrow w\}$.
   - Now $\overrightarrow{\mathcal{R}_\infty} \cup \overrightarrow{\mathcal{R}} \models \neg p$ does not hold.
   - Since $\overrightarrow{\mathcal{R}_\infty} \cup \overrightarrow{\mathcal{R}} \not\models \neg p \mathrel{|\!\sim} h$, therefore $\mathcal{K} \not\mathrel{|\!\approx} p \mathrel{|\!\sim} h$.
   - $i = 1$ is returned since it involves one cycle of eliminating the formulas with the lowest rank from $\mathcal{R}$.

(2) $\mathcal{K} \mathrel{|\!\approx} e \mathrel{|\!\sim} w$?
   - Going through the steps from lines 4 to 7, both $\mathcal{R}_0$ and $\mathcal{R}_1$ are taken away from $\mathcal{R}$ because $\overrightarrow{\mathcal{R}_\infty} \cup \overrightarrow{\mathcal{R}} \not\models \neg e$.

- $\overrightarrow{\mathcal{R}_\infty} \cup \overrightarrow{\mathcal{R}} := \{p \rightarrow a, e \rightarrow p, e \rightarrow fly\} \models e \mid\!\sim w$.
- As a result, the inference $\mathcal{K} \mathrel{|\!\approx} e \mid\!\sim w$ is valid.
- $i = 2$ is returned by the algorithm since two levels of formulas were removed.

Wang also introduced the KLMDefeasibleJustification algorithm (*Algorithm 6*), which consists of various smaller algorithms which come together to create the justification process. This algorithm employs the BaseRankForJustification and RationalClosureForJustification to calculate the defeasible implication. It also uses the ComputeAllJustifications algorithm. There is also a sub-algorithm responsible for identifying all explanations for conventional implication based on *Definations 2.1 and 2.3*. Lastly, the classical explanations are transformed into defeasible explanations using the dematerialisation algorithm (*Algorithm 1*).

The algorithm takes a knowledge base $\mathcal{K}$ and a defeasible implication $\alpha \mid\!\sim \beta$ as its inputs. When $\mathcal{K} \mathrel{|\!\approx} \alpha \mid\!\sim \beta$, the algorithm generates a collection of justification(s), $\mathcal{J}_i \in \mathcal{K}$, to support this implication such that every $\mathcal{J}_i \mathrel{|\!\approx} \alpha \mid\!\sim \beta$. However, if $\mathcal{K} \not\mathrel{|\!\approx} \alpha \mid\!\sim \beta$, the algorithm generates $\mathcal{T} \in \mathcal{K}$ such that $\mathcal{T} \not\mathrel{|\!\approx} \alpha \mid\!\sim \beta$.

Algorithm 6: KLMDefeasibleJustification

```
Input: Defeasible knowledge base $\mathcal{K}$ and defeasible query $\alpha \mid\!\sim \beta$
Output: Justification $\mathcal{J}$
$i := 0$;
$\mathcal{J} := \emptyset$;
entailment, $(\mathcal{R}_0, \mathcal{R}_1, \dots \mathcal{R}_\infty)$, rank :=
  RationalClosureForJustification($\mathcal{K}, \alpha \mid\!\sim \beta$);
if ¬ entailment then
    while $i <$ rank do
        $\mathcal{K} := \mathcal{K} \setminus \mathcal{K}_i$;
        $i := i + 1$;
    end
    return $\mathcal{K} \not\mathrel{|\!\approx} \alpha \mid\!\sim \beta$;
end
if rank == 0 then
    $\mathcal{J}$ := ComputeAllJustifications($\overrightarrow{\mathcal{K}}, \alpha \mid\!\sim \beta$);
    $\mathcal{J}$ := Dematerialisation($\mathcal{J}, \mathcal{K}$);
    return $\mathcal{J}$;
end
while $i <$ rank do
    $\mathcal{K} := \mathcal{K} \setminus \mathcal{K}_i$;
    $i := i + 1$;
end
$\mathcal{J}$ := ComputeAllJustifications($\overrightarrow{\mathcal{K}}, \alpha \mid\!\sim \beta$);
$\mathcal{J}$ := Dematerialisation($\mathcal{J}, \mathcal{K}$);
return $\mathcal{J}$;
```

The following two scenarios help explain the KLMDefeasibleJustification algorithm for $\mathcal{K}$ and entailment output from *Example 4.2*. Given a defeasible query as input, what is the justification set if the query is entailed by $\mathcal{K}$?

(1) $\mathcal{K} \mathrel{|\!\approx} p \mid\!\sim h$
- When examining the validity of defeasible entailment, the formulas in $\mathcal{R}_0$ were eliminated.
- As a result, the KLMDefeasibleJustification algorithm provides the subsequent explanation for the absence of entailment: $\{p \rightarrow a, e \rightarrow p, e \mid\!\sim w, p \mid\!\sim \neg w, a \mid\!\sim w, a \mid\!\sim h\} \not\mathrel{|\!\approx} p \mid\!\sim h$.
- Because $\mathcal{K} \not\mathrel{|\!\approx} p \mid\!\sim h, \mathcal{J} := \emptyset$

(2) $\mathcal{K} \mathrel{|\!\approx} e \mid\!\sim w$?
- When analyzing the defeasible query $e \mid\!\sim w$, it is found that $\mathcal{K} \mathrel{|\!\approx} e \mid\!\sim w$, and both $\mathcal{R}_0$ and $\mathcal{R}_1$ formulas were disregarded during the Rational Closure calculations.
- The loop from lines 16 to 19 in the KLMDefeasibleJustification process eliminates the formulas that were discarded from $\mathcal{K}$.
- As a result, $\mathcal{K} = \{p \rightarrow a, e \rightarrow p, e \mid\!\sim w\}$.
- Subsequently, the ComputeAllJustifications algorithm takes $\overrightarrow{\mathcal{K}}$ and $e \rightarrow w$ as inputs, which leads to the set of justifications $\mathcal{J} = \{e \rightarrow w\}$.
- Finally, the Dematerialisation algorithm transforms $\mathcal{J}$ into $\{e \mid\!\sim w\}$.
- This example highlights the contrast between classical and defeasible justifications. In the case of the classical implication $\overrightarrow{\mathcal{K}} \models e \rightarrow w$, there would have existed two explanations supporting the implication: $\{e \rightarrow w\}$ and $\{e \rightarrow p, p \rightarrow a, a \rightarrow w\}$. However, in the context of defeasible reasoning, the formula $a \mid\!\sim w$ is removed during the Rational Closure calculations resulting in only a single defeasible explanation remaining, $\{e \rightarrow w\}$.

5 SYSTEM DESIGN AND IMPLEMENTATION

In this section, we focus on the practical aspect of this paper. Besides presenting the theoretical algorithms for defeasible entailment and justification in *Sections 3* and *4*, we have turned these algorithms into a working software system tool. We have given this tool the name KLMDEETool, for *KLM Defeasible Entailment and Explanations Tool*. The source code is publicly available on *GitHub* [2] [3].

This section is structured as follows: In *Section 5.1*, we analyse the functional requirements and several design diagrams of the KLMDEETool. *Section 5.2* discusses the architectural fundamentals of the tool while *Section 5.3* outlines how it was implemented and the various external components used. The process of putting the mentioned algorithms into practice is also discussed in this section. We explain how the tool works and how to use it in *Section 5.4*. Lastly, the design of experimental scenarios and their execution for testing and assessment of the tool along with a summary of the feedback received is presented in *Section 5.5*.

5.1 Requirement Analysis and Design

In the process of defining what a software system needs to do (*functional requirements*) and how well it should do it (*non-functional requirements*), we gather information from various sources. This includes talking to users and stakeholders, watching how users perform tasks, studying procedure manuals and task lists, reviewing

[2] *GitHub* is a web-based platform used for collaborating on software projects by hosting, sharing, and managing them using version control systems.
[3] The source code is available at https://github.com/ChiefMonk/DEE. Compilation and execution instructions are included in a readme.rm file.

requests to improve the current system, and examining marketing materials and product definitions [19]. In our case, because our software needs to perform specific functions with a limited knowledge base size, we mainly focused on collecting functional requirements. To create the functional requirements for the KLMDEETool, we used feedback from our supervisors, analysed a similar tool developed by Wang [18], and studied the pseudo algorithms for entailment and justification. Since our tool primarily deals with these algorithms, our functional requirements involve the ability to handle logical arguments, manage exceptions, and explain its conclusions. In addition to functional requirements, we also considered non-functional aspects during the design phase. These include performance measures like speed and accuracy, user-friendliness in terms of interfaces, and reliability aspects such as how the system handles errors and maintains robustness. To aid in project management and better estimate delivery timelines that expose critical paths and resource allocation, we developed a *Program Evaluation and Review Technique (PERT) Chart* presented in *Appendix B.1.*

During the design phase, we concentrated on creating a detailed plan for how the tool will meet all the functional requirements and some non-functional ones. This included how the pseudo algorithms for defeasible reasoning will be translated into an object-oriented language, creating data structures to store arguments and counterarguments for knowledge bases, and planning for user interfaces that clearly present explanations. The output of this phase was the development of *UML diagrams*, *class diagrams* and *use-cases diagram* used to represent the system's architecture and interactions among components and external users. These are presented and explained in *Appendix B.*

Throughout these phases, it was important to continuously engage with the stakeholders for feedback and validation, ensuring that the tool effectively solves the intended problems in a user-friendly manner. After analysing the problem and available requirements, we decided to develop two applications that utilise the same codebase and interfaces: a *Desktop Application* and a *Console Application.*

5.2 System Architecture

The KLMDEETool is developed in Java, an object-oriented programming language, and is structured around a well-defined architecture that adheres to the *Multi-tier Architecture* pattern of software engineering [17]. It consists of several interconnected components that work together to facilitate the processing and analysis of defeasible reasoning scenarios. The architecture is designed to ensure efficiency, modularity, and ease of use as shown in *Figure 4.*

At its core, the system encompasses the *UI Manager* responsible for providing a unified user interface for all the functionality KLMDEETool offers. This *user interface* tier or component offers both a *command line interface (CLI)* and a *graphical user interface (GUI)* for user interaction. The CLI allows users to specify the inputs as parameters on the command line while the GUI provides a user-friendly interface for input and result visualisation. Both of these interfaces consume and provide the same functionality but only differ in presentation and the verbosity of the debug and information outputs. The user interface manager interacts with various modules and services for different functionalities, such as

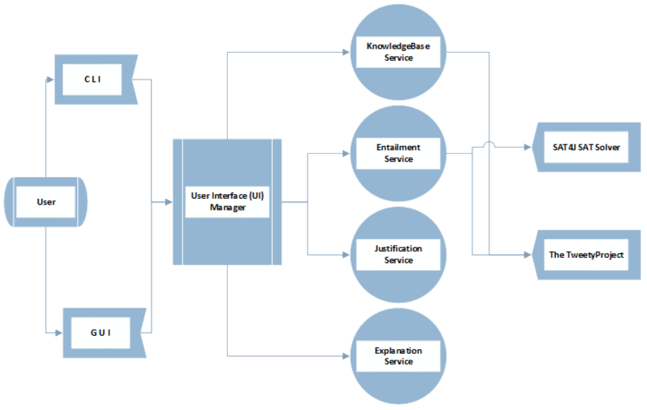
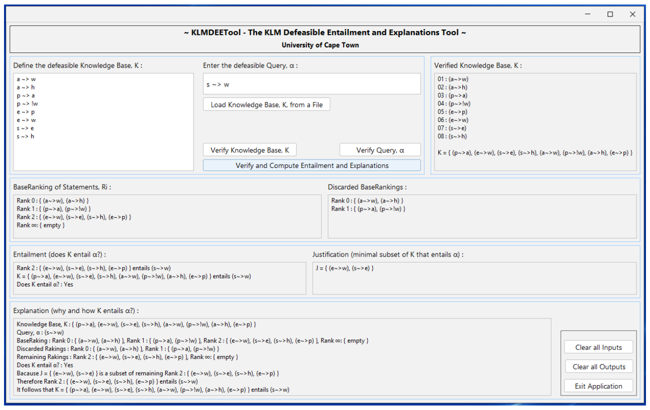


Figure 4: The Multi-tier Architecture of the KLMDEETool

the *entailment service*, *justification service*, *explanation service* and the *knowledgebase service*, responsible for storing and managing the defeasible knowledge base related data. These services in turn communicate with 3rd party libraries and tools to effect the desired action and result.

The entailment and justification service layers employ algorithms derived from the KLM approach discussed in *Sections 3* and *4*. It processes the input knowledge base and queries to determine if entailments hold true. In cases where entailments are established, the justification service generates justifications for these conclusions based on the algorithms for justification discussed in *Section 4*. The explanation service module is responsible for providing step-by-step explanations for the identified entailments. It utilises the generated justifications to guide users through the reasoning process, outlining the reasons behind each conclusion and showcasing the specific rules or statements that contribute to the whole result.


~ KLMDEETool - The KLM Defeasible Entailment and Explanations Tool ~
University of Cape Town
Load Knowledge Base, K, from a File
Verify Knowledge Base, K
Verify Query, α
Verify and Compute Entailment and Explanations
Clear all Inputs
Clear all Outputs
Exit Application

Define the defeasible Knowledge Base, K :

```
a ~> w
a ~> h
p ~> a
p ~> !w
e ~> p
e ~> w
s ~> e
s ~> h
```

Enter the defeasible Query, α :

```
s ~> w
```

Verified Knowledge Base, K :

```
01 : (a~>w)
02 : (a~>h)
03 : (p~>a)
04 : (p~>!w)
05 : (e~>p)
06 : (e~>w)
07 : (s~>e)
08 : (s~>h)

K = { (p~>a), (e~>w), (s~>e), (s~>h), (a~>w), (p~>!w), (a~>h), (e~>p) }
```

BaseRanking of Statements, Ri :

```
Rank 0 : { (a~>w), (a~>h) }
Rank 1 : { (p~>a), (p~>!w) }
Rank 2 : { (e~>w), (s~>e), (s~>h), (e~>p) }
Rank ∞: { empty }
```

Discarded BaseRankings :

```
Rank 0 : { (a~>w), (a~>h) }
Rank 1 : { (p~>a), (p~>!w) }
```

Entailment (does K entail α?) :

```
Rank 2 : { (e~>w), (s~>e), (s~>h), (e~>p) } entails (s~>w)
K = { (p~>a), (e~>w), (s~>e), (s~>h), (a~>w), (p~>!w), (a~>h), (e~>p) } entails (s~>w)
Does K entail α? : Yes
```

Justification (minimal subset of K that entails α) :

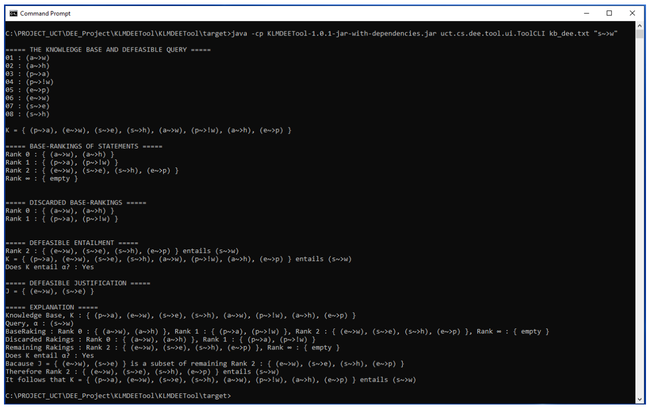
```
J = { (e~>w), (s~>e) }
```

Explanation (why and how K entails α?) :

```
Knowledge Base, K : { (p~>a), (e~>w), (s~>e), (s~>h), (a~>w), (p~>!w), (a~>h), (e~>p) }
Query, α : (s~>w)
BaseRaking : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>a), (p~>!w) }, Rank 2 : { (e~>w), (s~>e), (s~>h), (e~>p) }, Rank ∞: { empty }
Discarded Rakings : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>a), (p~>!w) }
Remaining Rakings : Rank 2 : { (e~>w), (s~>e), (s~>h), (e~>p) }, Rank ∞: { empty }
Does K entail α? : Yes
Bacause J = { (e~>w), (s~>e) } is a subset of remaining Rank 2 : { (e~>w), (s~>e), (s~>h), (e~>p) }
Therefore Rank 2 : { (e~>w), (s~>e), (s~>h), (e~>p) } entails (s~>w)
It follows that K = { (p~>a), (e~>w), (s~>e), (s~>h), (a~>w), (p~>!w), (a~>h), (e~>p) } entails (s~>w)
```

Figure 5: The KLMDEETool GUI with an active example output

Overall, the architecture of this software system blends the principles of defeasible reasoning within the KLM framework with a modular and user-friendly design, enabling users to explore and understand complex defeasible scenarios and their explanations effectively. The KLMDEETool offers a GUI that facilitates user engagement with the software. We opted for the *Java Swing package*



```
C:\PROJECT_UCT\DEE_Project\KLMDEETool\KLMDEETool\target>java -cp KLMDEETool-1.0.1-jar-with-dependencies.jar uct.cs.dee.tool.ui.ToolCLI kb_dee.txt "s~>w"

===== THE KNOWLEDGE BASE AND DEFEASIBLE QUERY =====
01 : (a~>w)
02 : (a~>h)
03 : (p~>a)
04 : (p~>!w)
05 : (e~>p)
06 : (e~>w)
07 : (s~>e)
08 : (s~>h)

K = { (p~>a), (e~>w), (s~>e), (s~>h), (a~>w), (p~>!w), (a~>h), (e~>p) }

===== BASE-RANKINGS OF STATEMENTS =====
Rank 0 : { (a~>w), (a~>h) }
Rank 1 : { (p~>a), (p~>!w) }
Rank 2 : { (e~>w), (s~>e), (s~>h), (e~>p) }
Rank ∞ : { empty }


===== DISCARDED BASE-RANKINGS =====
Rank 0 : { (a~>w), (a~>h) }
Rank 1 : { (p~>a), (p~>!w) }


===== DEFEASIBLE ENTAILMENT =====
Rank 2 : { (e~>w), (s~>e), (s~>h), (e~>p) } entails (s~>w)
K = { (p~>a), (e~>w), (s~>e), (s~>h), (a~>w), (p~>!w), (a~>h), (e~>p) } entails (s~>w)
Does K entail α? : Yes

===== DEFEASIBLE JUSTIFICATION =====
J = { (e~>w), (s~>e) }

===== EXPLANATION =====
Knowledge Base, K : { (p~>a), (e~>w), (s~>e), (s~>h), (a~>w), (p~>!w), (a~>h), (e~>p) }
Query, α : (s~>w)
BaseRaking : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>a), (p~>!w) }, Rank 2 : { (e~>w), (s~>e), (s~>h), (e~>p) }, Rank ∞ : { empty }
Discarded Rakings : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>a), (p~>!w) }
Remaining Rakings : Rank 2 : { (e~>w), (s~>e), (s~>h), (e~>p) }, Rank ∞ : { empty }
Does K entail α? : Yes
Bacause J = { (e~>w), (s~>e) } is a subset of remaining Rank 2 : { (e~>w), (s~>e), (s~>h), (e~>p) }
Therefore Rank 2 : { (e~>w), (s~>e), (s~>h), (e~>p) } entails (s~>w)
It follows that K = { (p~>a), (e~>w), (s~>e), (s~>h), (a~>w), (p~>!w), (a~>h), (e~>p) } entails (s~>w)

C:\PROJECT_UCT\DEE_Project\KLMDEETool\KLMDEETool\target>
```

Figure 6: The `KLMDEETool` CLI with an active example output

to develop the GUI due to its straightforward nature and suitability for our tool's requirements. *Figures 5* and *6* visually portray the GUI and CLI interfaces of the `KLMDEETool` with an example showing a valid entailment, justification and explanation.

5.3 System Implementation

To aid in processing and solving complex logical expressions and queries efficiently, the `KLMDEETool` incorporates and relies on two external libraries or packages: the *TweetyProject* [4] and the *SAT4J SAT Solver* [5].

The *TweetyProject* offers a software framework that encompasses models and functions applicable to most forms of logic, including propositional logic. `KLMDEETool` implementation extends the propositional logic models from the *TweetyProject* to construct the necessary models and functions specific to the KLM Framework. However, due to constraints and constants defined by the *TweetyProject*, the conventional symbols commonly used in literature to represent KLM Framework operations cannot be employed. Instead, the propositional logic operators are replaced by the *TweetyProject*'s equivalents as shown in *Table 7*. Additionally, we developed a parser capable of reading strings like α ~> β (for $\alpha \mid\!\sim \beta$) and generating a defeasible implication instance, enabling the tool to carry out operations such as materialisation. For the resulting classical justification that the `KLMDefeasibleJustification` algorithm materialises, a tool is first required to compute classical entailment based on $\mathcal{K}$ and the query. In this context, we utilise the *SAT4J SAT solver* [2] to perform the necessary classical entailment computations. The execution of the base rank, Rational Closure defeasible entailment, justification and dematerialisation processes adheres to *Algorithms 1, 4, 5 and 6* respectively. To determine a single classical justification, we closely follow Horridge's algorithms as shown in *Appendix A.2*.

5.4 Functionality and Usage

In orchestrating the program's operations, the UI Manager dictates the actions executed in response to user interactions. It undertakes

[4]The *TweetyProject* consists of diverse Java libraries that embody strategies for knowledge representation formalisms, encompassing classical logics, conditional logics, and probabilistic logics.
[5]The *SAT4J SAT Solver* is a Java library designed to solve problems related to boolean satisfaction and optimisation.

Connective Name	Symbol	*TweetyProject* equivalent
negation	$\neg$	!
disjunction	$\vee$	||
conjunction	$\wedge$	&&
implication	$\rightarrow$	=>
bi-implication	$\leftrightarrow$	<=>
defeasible implication	$\mid\!\sim$	~>

Figure 7: Logical symbol replacements with equivalents

a range of actions, each contingent upon specific user inputs. As *Figure 5* describes, the Desktop Application (GUI) functions as follows:

(1) The knowledge base $\mathcal{K}$ is defined by either:
   - entering it manually, a statement per line, in the provided text area.
   - clicking the *"Load Knowledge Base, $\mathcal{K}$, from a File"* button that opens a window for the user to choose a file with predefined statements. The contents of the file are then displayed in the manual entry text area.

(2) The user can optionally choose to pre-verify the knowledge base by clicking the *"Verify Knowledge Base"* button.
(3) Once the knowledge base is verified, it is shown in the correct format in the area provided.
(4) The user then inputs text that represents the defeasible query α.
(5) The user can optionally choose to pre-verify the defeasible query by clicking the *"Verify Query"* button.
(6) Upon clicking the *"Verify and Compute Defeasible Entailment and Explanations"* button, the program then:
   - Uses the `BaseRankForJustification` algorithm to base rank the statements and displays the results in the appropriate text area.
   - Uses the `RationalClosureForJustification` algorithm to determine if $\mathcal{K}$ entails the query or not.
   - Uses the `KLMDefeasibleJustification` algorithm to calculate the defeasible justifications and explain why and how the entailment holds.

(7) The resulting defeasible base ranking, discarded ranks, entailment, justification and explanations are presented in the output text areas provided.
(8) To conclude the program, the user can exit by clicking the *"Exit Application"* button.

Figure 6 shows the functionality of the Console Application (CLI) application described as follows:

(1) The path to the file containing the knowledge base $\mathcal{K}$ is passed on the command line as the *first parameter*.
(2) The defeasible query α is passed as a string on the command line as the *second parameter*.
(3) When the user presses *Enter*, both $\mathcal{K}$ and α are verified and any errors present are output to the console.
(4) The same methods and functions used by the GUI are employed also for the CLI-based tool.

(5) The resulting defeasible base ranking, discarded ranks, entailment, justification and explanations are output to the console.

The `KLMDEETool` receives two input parameters: a defeasible knowledge base and a defeasible query. As illustrated in *Figures 5* and *6*, an exemplar input for the knowledge base is provided, represented by $\mathcal{K} = \{a \mid\sim w, a \mid\sim h, p \mid\sim a, p \mid\sim \neg w, e \mid\sim p, e \mid\sim w, s \mid\sim e, s \mid\sim h\}$. For the query, such as $s \mid\sim w$, a user enters it into the designated text field on the interface. Within the output text areas, the tool presents a list of base-ranked statements, a sequence of discarded formula ranks from the knowledge base (if any), along with the formulas that remain subsequent to each elimination. The tool then furnishes credible defeasible justifications concerning the specified defeasible entailment $\mathcal{K} \approx s \mid\sim w$. Lastly, the `KLMDEETool` outputs a chronological explanation of how and why the entailment was arrived at by combining all the various outputs and commentary in a single text area.

5.5 Testing, Evaluation and Execution

In the realm of this paper, the rigorous evaluation of the `KLMDEETool` is paramount. *Unit tests* played a pivotal role by meticulously scrutinising individual components and algorithms to ascertain their accuracy and correctness. These tests ensured that the foundational building blocks of the `KLMDEETool`, responsible for capturing complex inference patterns, operate flawlessly. To practically achieve this, we used the *JUnit* [6] Java framework and the developed tests are included in the source code.

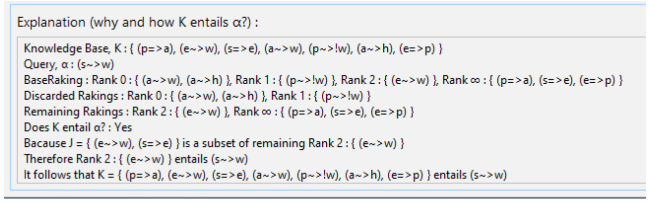
Functional testing extended this scrutiny to assess the tool's compliance with specified requirements and its ability to effectively perform the desired logical operations. In parallel, *usability tests* delved into the user experience, ensuring that the tool's interface, interaction flows, and presentation align with user expectations and promote efficient utilisation. Furthermore, *performance tests* gauged the capacity of the `KLMDEETool` to handle intricate reasoning tasks within feasible timeframes, guaranteeing its responsiveness and scalability even when confronted with extensive knowledge bases and complex entailment scenarios.

To make testing easier in terms of logical understanding and performance, we limited the number of statements in the input knowledge base to a maximum of 10. All these tests were performed by us. To evaluate the functional efficiency of the `KLMDEETool`, we developed and executed multiple tests such as those depicted in *Figure 5*. We examined scenarios involving inputs from a defeasible knowledge base $\mathcal{K}$, consisting of statements such as the following. The desktop application (GUI) output of a full explanation for $\mathcal{K} \approx s \mid\sim w$ is shown in *Figure 8*.

(1) Animals typically are wild ($a \mid\sim w$)
(2) Animals typically hunt ($a \mid\sim h$)
(3) Pets are animals ($p \rightarrow a$)
(4) Pets typically are not wild ($p \mid\sim \neg w$)
(5) ExoticPets are pets ($e \rightarrow p$)
(6) ExoticPets typically are wild ($e \mid\sim w$)
(7) Simba is an ExoticPet ($s \rightarrow e$)

or

$$\mathcal{K} = \{a \mid\sim w, a \mid\sim h, p \rightarrow a, p \mid\sim \neg w, e \rightarrow p, e \mid\sim w, s \rightarrow e\}$$



```
Explanation (why and how K entails α?) :
Knowledge Base, K : { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) }
Query, α : (s~>w)
BaseRaking : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }, Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Discarded Rakings : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }
Remaining Rakings : Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Does K entail α? : Yes
Bacause J = { (e~>w), (s=>e) } is a subset of remaining Rank 2 : { (e~>w) }
Therefore Rank 2 : { (e~>w) } entails (s~>w)
It follows that K = { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) } entails (s~>w)
```

Figure 8: GUI explanation output for $\mathcal{K} \approx s \mid\sim h$?

As the figures in *Appendix C* show, the tool produced accurate defeasible entailment and justification results for each test scenario executed, aligning well with the theoretical illustrations presented in *Sections 3* and *4*. This feedback coupled, with a manual determination of a justification set and the verified concurrency from our supervisor, was enough to determine and justify the correct operation of the `KLMDEETool`.

6 DISCUSSION AND RELATED WORK

Kraus, Lehmann, and Magidor [12] introduced the KLM approach to defeasible reasoning through the proposition of a structured formal framework. This framework aimed to tackle the complexities brought about by non-monotonic reasoning and exceptions inherent within formal logic systems. Within their work, Lehmann and Magidor [14] not only explained the concept of Rational Closure but also meticulously outlined the postulates governing rational defeasible entailment. Additionally, they provided an algorithm for determining the Rational Closure entailment.

In a unifying endeavour, Kaliski [11] compiled existing literature, orchestrating a contemporary synthesis of KLM-style defeasible reasoning into a thesis. This comprehensive compilation encompasses a unified assortment of theoretical contributions within the KLM framework, serving as a valuable and consolidated point of reference for researchers.

Horridge [10] embarked on a thorough exploration into classical justification, meticulously delving into its computational aspects. This exploration yielded algorithms for enumerating classical justifications, further supplemented by an in-depth efficiency analysis of these algorithms.

Everett et al. [8] expanded on the ideas of weak justification, which were initially applied only to Rational Closure. This was done by extending these ideas to include Relevant and Lexicographic Closure and introducing algorithms to list these justifications. Additionally, an assessment of a modified definition of strong justification was done, taking into account specific challenges, and putting forth an algorithm to identify such justifications in the context of Rational Closure.

Chama's [7] contribution revolved around introducing a concept defined as defeasible justification for Description Logics, accompanied by a tailored justification algorithm specifically designed for Rational Closure defeasible entailment. Drawing from Horridge's algorithms, Chama's approach aligns with the reasoning process for Rational Closure, but with an emphasis on classical justification rather than classical entailment.

[6] *JUnit* is a widely adopted open-source framework that provides a structured and organised approach to writing and running unit tests in Java applications.

In addition to providing the necessary theory on defeasible justification within the KLM framework and extending Chama's algorithms to propositional logic, Wang [18] took a pragmatic approach by creating a software tool complete with a graphical user interface. This tool effectively implements Rational Closure justification algorithms enabling the exploration of a knowledge base against specific queries. Despite its simplicity, the program correctly generates a set of justifications for the defeasible entailment based on simple knowledge bases. In a validation effort, representative examples were employed to evaluate the algorithm's performance, confirming its alignment with intuitive expectations.

7 CONCLUSIONS

This paper begins with an overview of knowledge representation and reasoning, introducing how propositional logic is utilised for expressing and representing such reasoning. It then distinguishes between classical reasoning and defeasible reasoning using practical examples. The challenges of comprehending conclusions drawn from defeasible reasoning are highlighted, and the role of justifications in explaining defeasible entailments and aiding user understanding is discussed. The paper addresses this need by presenting algorithms developed by Chama [7] and extended by Wang [18] that compute justifications for the KLM-style Rational Closure variant of defeasible entailment.

The primary contribution of this paper is a software system tool implementing the proposed defeasible entailment and justification algorithms. Following the multi-tier architecture pattern, the software provides both a CLI and a GUI for user interaction. Users can input a defeasible knowledge base and a defeasible query string, and the tool displays the resulting base ranking, discard rankings, defeasible entailment, justification set, and the corresponding explanations in a user-friendly manner. To validate its effectiveness, the tool underwent a thorough testing encompassing a range of representative test cases. Therefore, the `KLMDEETool` can be used as a debugging service for knowledge bases. The source code of the tool is publicly available on *GitHub*.

8 FUTURE WORKS

The algorithms and principles explored in this paper concerning explanations for Rational Closure entailment are rooted in the work and propositions done by Chama [7] and further extended by Wang [18]. Future research endeavours could potentially leverage this outcome to apply it to other conceptions of defeasible entailment within the KLM framework or even across broader contexts of defeasible entailment. The optimised algorithms for justification could be extended to other entailment formalisms such as Lexicographic Closure and Relevant Closure.

The `KLMDEETool` serves as a rudimentary solution for simple defeasible justification suitable for straightforward knowledge bases. Enhancements to the tool's scalability and efficiency are viable improvements. A comprehensive assessment of the tool's performance would provide insight into its limitations. Such assessments should consider factors such as the size of the knowledge base, the intricacies associated with identifying justifications, and various adjustable parameters of the tool. The `KLMDEETool` can be extended further to other defeasible reasoning formalisms such as Lexicographic Closure and Relevant Closure. The logical operations can also be modified and tested with bigger knowledge bases and complex queries. *Natural Language Processing (NLP)* techniques and tools could also be employed to provide more robust explanations, perhaps in a user's chosen language.

ACKNOWLEDGEMENTS

I want to thank my supervisor, *Professor Tommie Meyer*, for his unwavering patience, guidance and invaluable insights. I also thank *Mr Gary Stewart* for the valuable feedback and progress valuation. I am also grateful to my colleague, *Orefile Morule*, for the numerous discussions about knowledge, representation, reasoning and life. For endless prayers and encouragement, I dedicate this work to my loving family, especially my mother *Alice* and *Auntie Lillian*. May the good Lord bless them all, abundantly.

REFERENCES

[1] Mordechai Ben-Ari. 2012. Propositional logic: Formulas, models, tableaux. *Mathematical Logic for Computer Science* (2012), 7–47.
[2] Berre, Daniel Le. 2008. SAT4J SAT Solver. https://www.sat4j.org. Accessed: 2023-08-17.
[3] Ronald J Brachman and Hector J Levesque. 2004. Actions. In *Knowledge Representation and Reasoning*. Elsevier, 285–303.
[4] Gerhard Brewka and Markus Ulbricht. 2019. Strong explanations for nonmonotonic reasoning. *Description Logic, Theory Combination, and All That: Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday* (2019), 135–146.
[5] Giovanni Casini, Thomas Meyer, Kodylan Moodley, and Riku Nortjé. 2014. Relevant closure: A new form of defeasible reasoning for description logics. In *Logics in Artificial Intelligence: 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings 14*. Springer, 92–106.
[6] Giovanni Casini, Thomas Meyer, and Ivan Varzinczak. 2019. Taking Defeasible Entailment Beyond Rational Closure. In *Logics in Artificial Intelligence*. Springer International Publishing, Cham, 182–197.
[7] Victoria Chama. 2020. *Explanation for defeasible entailment*. Master's thesis. Faculty of Science, University of Cape Town, Rondebosch, Cape Town, 7700.
[8] Lloyd Everett, Emily Morris, and Thomas Meyer. 2021. Explanation for KLM-Style Defeasible Reasoning. In *Southern African Conference for Artificial Intelligence Research*. Springer, South Africa, 192–207.
[9] Michael Freund. 1998. Preferential reasoning in the perspective of Poole default logic. *Artificial Intelligence* 98, 1 (1998), 209–235. https://doi.org/10.1016/S0004-3702(97)00053-2
[10] Matthew Horridge. 2011. *Justification based explanation in ontologies*. Ph. D. Dissertation. University of Manchester, UK.
[11] Adam Kaliski. 2020. *An Overview of KLM-Style Defeasible Entailment*. Master's thesis. Faculty of Science, University of Cape Town, Rondebosch, Cape Town, 7700.
[12] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. 1990. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence* 44, 1 (1990), 167–207. https://doi.org/10.1016/0004-3702(90)90101-5
[13] Daniel Lehmann. 1999. Another perspective on Default Reasoning. *Annals of Mathematics and Artificial Intelligence* 15 (11 1999). https://doi.org/10.1007/BF01535841
[14] Daniel Lehmann and Menachem Magidor. 1992. What does a conditional knowledge base entail? *Artificial Intelligence* 55, 1 (1992), 1–60. https://doi.org/10.1016/0004-3702(92)90041-U
[15] Orefile Morule. 2023. *Justifications for Classical Entailment*. Honour's thesis. Faculty of Science, University of Cape Town, Rondebosch, Cape Town, 7700.
[16] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. 2005. Debugging OWL ontologies. In *Proceedings of the 14th international conference on World Wide Web*. 633–640.
[17] Savitch Walter and Mock Kenrick. 2016. *Absolute Java* (6th ed.). Pearson Education, Boston.
[18] Steve Wang. 2022. *Defeasible Justification for the KLM Framework*. Master's thesis. Faculty of Science, University of Cape Town, Rondebosch, Cape Town, 7700.
[19] Jeffrey L Whitten and Lonnie D Bentley. 2007. *Systems Analysis and Design Methods* (7th ed.). McGraw-Hill/Irwin, New York, USA.

A ALGORITHMS

A.1 Compute All Justifications Algorithm

To compute all justifications, Horridge suggests a method that utilises the HS-Tree approach to discover all classical justifications for a Description Logics entailment. This technique, initially introduced in theory by Reiter, was designed to identify the smallest set that overlaps with a given collection of conflicting sets within a defined universe. Additionally, the HS-Tree technique can be employed to dynamically identify these conflicting sets [18].

Algorithm 7: `ComputeAllJustifications`

```
Input: Knowledge base $\mathcal{K}$ and entailment $\alpha$
Output: Set of Justifications $\mathcal{J}$
$S_{working} := \mathcal{K}$;
$X_{explored} := \emptyset$;
$X_{result} := \emptyset$;
$\mathcal{J}_{root}$ := ComputeSingleJustification($S_{working}, \alpha$);
$X_{result} := X_{result} \cup \{\mathcal{J}_{root}\}$;
$v_{root}$ := GetFreshNode($\mathcal{J}_{root}$);
Enqueue($v_{root}, Q$);
SetRoot($T_{hst}, v_{root}$);
while $Q \neq \emptyset$ do
    $v_{head}$ = Dequeue($Q$);
    $j_{head}$ = GetLabel($v_{head}$);
    for $\beta \in j_{head}$ do
        $S_{path}$ = GetPathToRootLabelSet($v_{head}, T_{hst}$) $\cup \{\beta\}$;
        if $S_{path} \notin X_{explored}$ then
            $X_{explored} = X_{explored} \cup \{S_{path}\}$;
            $\mathcal{J}'$ =
             ComputeNonIntersectingJustification($S_{path}, X_{result}$);

            if $\mathcal{J}' == \emptyset$ then
                $S_{working} = S_{working} \setminus \{S_{path}\}$;
                $\mathcal{J}'$ =
                 ComputeSingleJustification($S_{working}, \alpha$);
                $S_{working} = S_{working} \cup \{S_{path}\}$;
            end
            $v_{fresh}$ = GetFreshNode($\mathcal{J}'$);
            $e$ = GetFreshEdge(($v_{fresh}, v_{head}$), $\beta$);
            $T_{hst} = T_{hst} \cup \{e\}$;
            if $\mathcal{J}' \neq \emptyset$ then
                $X_{result} = X_{result} \cup \{\mathcal{J}'\}$;
                Enqueue($v_{fresh}, Q$);
            end
        end
    end
end
return $X_{result}$;
```

A.2 Compute Single Justification Algorithm

This algorithm is designed to calculate a single justification for an entailment in Propositional Logic. It algorithm is built upon the sub-algorithms `ExpandFormulas` and `ContractFormulas` respectively. Initially, the algorithm identifies a subset S from the knowledge base $\mathcal{K}$ that leads to the query's entailment using the `ExpandFormulas` algorithm. Subsequently, this set S is reduced using the `ContractFormulas` algorithm, aiming to find the smallest subset of S that still entails the query [18].

Algorithm 8: `ComputeSingleJustification`

```
Input: Knowledge base $\mathcal{K}$ and entailment $\alpha$
Output: Justifications $\mathcal{J}$
if $\alpha \in \mathcal{K}$ then
    return $\alpha$;
end
$S$ := ExpandFormulas($\mathcal{K}, \alpha$);
if $S == \emptyset$ then
    return $\emptyset$;
end
$\mathcal{J}$ := ContractFormulas($S, \alpha$);
return $\mathcal{J}$
```

B KLMDEETOOL DESIGN DIAGRAMS

B.1 PERT Chart

A PERT (Program Evaluation and Review Technique) Chart, is a project management tool used in software development and other industries to visualize and plan the tasks and dependencies within a project. It consists of nodes (representing project tasks) connected by arrows (representing task dependencies). The chart helps project managers and teams estimate the project timeline, identify critical paths, allocate resources efficiently, and manage the project's progress effectively. It is particularly useful for complex projects with many interrelated tasks [19], which are applicable in this case. The PERT Chart for the `KLMDEETool` is shown in *Figure 9*.

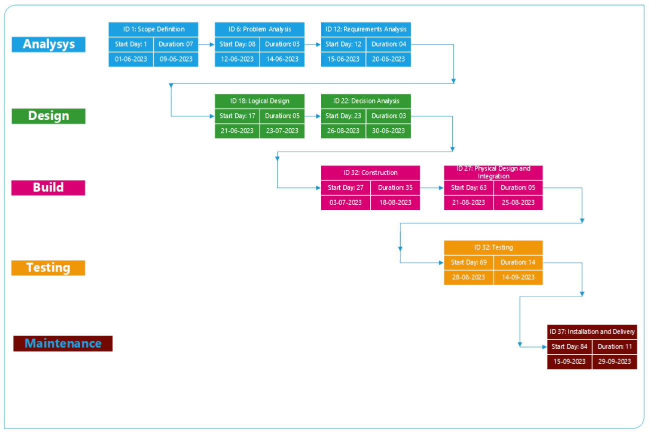


Figure 9: The `KLMDEETool` PERT Chart

Software development typically involves several phases, which can vary depending on the specific development methodology being used. Here are the five phases we employed for developing the `KLMDEETool` as described by Whitten and Bentley [19]:

(1) *Planning and Analysis* - The team conducted a thorough examination of the project's requirements.
(2) *Design* - Create a blueprint for the software's architecture and user interface.
(3) *Build or Implementation* - Write the actual code for the software based on the design specifications.

(4) *Testing* - Software undergoes rigorous testing to identify and fix bugs, ensure functionality meets requirements, and assess performance and security.

(5) *Maintenance* - The software is deployed to production environments for end-users to access and utilize. Deployment may involve configuration, setup, and ongoing maintenance

B.2 Use-Cases Diagram

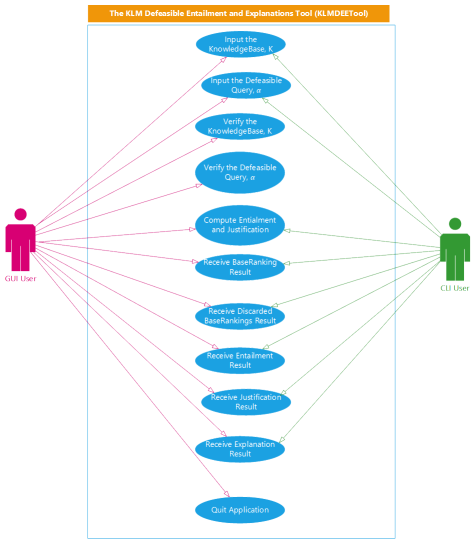

The KLM Defeasible Entailment and Explanations Tool (KLMDEETool)
Input the KnowledgeBase, K
Input the Defeasible Query, α
Verify the KnowledgeBase, K
Verify the Defeasible Query, α
Compute Entialment and Justification
Receive BaseRanking Result
Receive Discarded BaseRankings Result
Receive Entailment Result
Receive Justification Result
Receive Explanation Result
Quit Application
GUI User
CLI User

Figure 10: The `KLMDEETool` Use-Cases Diagram

B.3 Class Diagram

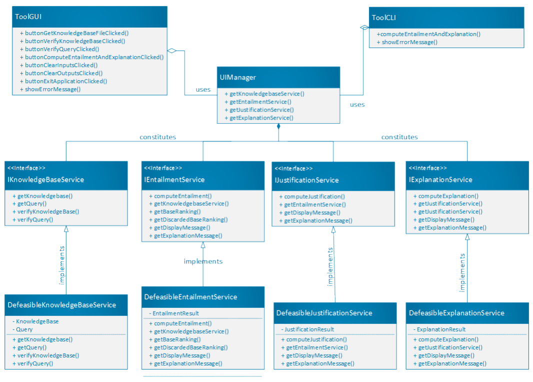


Figure 11: The `KLMDEETool` Class Diagram

B.4 UML Diagram

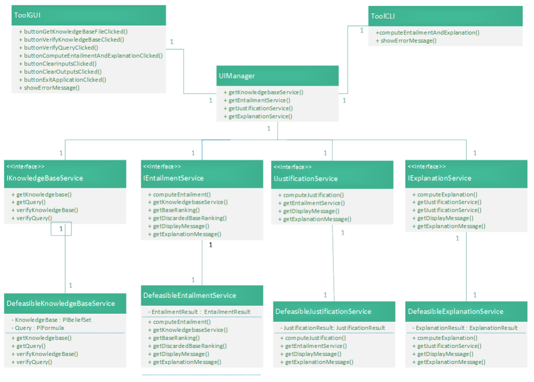

ToolGUI
+ buttonGetKnowledgeBaseFileClicked()
+ buttonVerifyKnowledgeBaseClicked()
+ buttonVerifyQueryClicked()
+ buttonComputeEntailmentAndExplanationClicked()
+ buttonClearInputsClicked()
+ buttonClearOutputsClicked()
+ buttonExitApplicationClicked()
+ showErrorMessage()
ToolCLI
+computeEntailmentAndExplanation()
+ showErrorMessage()
UIManager
+ getKnowledgebaseService()
+ getEntailmentService()
+ getJustificationService()
+ getExplanationService()
<<Interface>>
IKnowledgeBaseService
+ getKnowledgebase()
+ getQuery()
+ verifyKnowledgeBase()
+ verifyQuery()
<<Interface>>
IEntailmentService
+ computeEntailment()
+ getKnowledgebaseService()
+ getBaseRanking()
+ getDiscardedBaseRanking()
+ getDisplayMessage()
+ getExplanationMessage()
<<Interface>>
IJustificationService
+ computeJustification()
+ getEntailmentService()
+ getDisplayMessage()
+ getExplanationMessage()
<<Interface>>
IExplanationService
+ computeExplanation()
+ getJustificationService()
+ getJustificationService()
+ getDisplayMessage()
+ getExplanationMessage()
DefeasibleKnowledgeBaseService
- KnowledgeBase : PlBeliefSet
- Query : PlFormula
+ getKnowledgebase()
+ getQuery()
+ verifyKnowledgeBase()
+ verifyQuery()
DefeasibleEntailmentService
- EntailmentResult : EntailmentResult
+ computeEntailment()
+ getKnowledgebaseService()
+ getBaseRanking()
+ getDiscardedBaseRanking()
+ getDisplayMessage()
+ getExplanationMessage()
DefeasibleJustificationService
- JustificationResult: JustificationResult
+ computeJustification()
+ getEntailmentService()
+ getDisplayMessage()
+ getExplanationMessage()
DefeasibleExplanationService
- ExplanationResult : ExplanationResult
+ computeExplanation()
+ getJustificationService()
+ getDisplayMessage()
+ getExplanationMessage()

Figure 12: The `KLMDEETool` UML Diagram

C KLMDEETOOL TESTING AND EXECUTION

For the following a defeasible knowledge base $\mathcal{K}$, we tested the `KLMDEETool` with different defeasible queries.

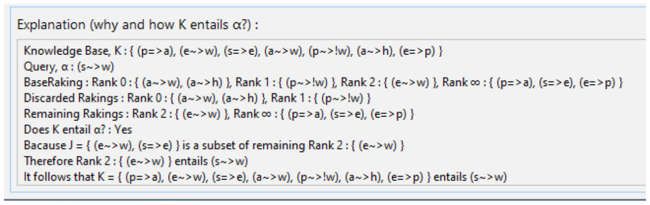
(1) Animals typically are wild ($a \mid\sim w$)
(2) Animals typically hunt ($a \mid\sim h$)
(3) Pets are animals ($p \rightarrow a$)
(4) Pets typically are not wild ($p \mid\sim \neg w$)
(5) ExoticPets are pets ($e \rightarrow p$)
(6) ExoticPets typically are wild ($e \mid\sim w$)
(7) Simba is an ExoticPet ($s \rightarrow e$)

or

$$\mathcal{K} = \{a \mid\sim w, a \mid\sim h, p \rightarrow a, p \mid\sim \neg w, e \rightarrow p, e \mid\sim w, s \rightarrow e\}$$

For the different input query scenarios executed below, for emphasis, we only show the explanation output part for both the Desktop (GUI) and Console (CLI) applications. The explanation output includes all the information shown in other sections, such as the knowledge base $\mathcal{K}$, defeasible query α, base ranked statements, discarded base ranks, entailment and justification.

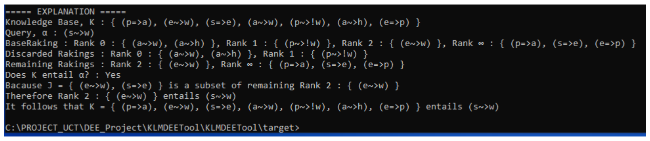
C.1 $\mathcal{K} \mid\approx$ ***Simba*** $\mid\sim$ ***Wild*** ($\mathcal{K} \mid\approx s \mid\sim w$)?



Explanation (why and how K entails α?) :

```
Knowledge Base, K : { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) }
Query, α : (s~>w)
BaseRaking : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }, Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Discarded Rakings : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }
Remaining Rakings : Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Does K entail α? : Yes
Bacause J = { (e~>w), (s=>e) } is a subset of remaining Rank 2 : { (e~>w) }
Therefore Rank 2 : { (e~>w) } entails (s~>w)
It follows that K = { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) } entails (s~>w)
```

Figure 13: GUI explanation output for $\mathcal{K} \mid\approx s \mid\sim w$?

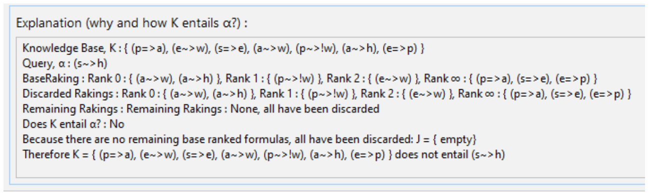


```
===== EXPLANATION =====
Knowledge Base, K : { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) }
Query, α : (s~>w)
BaseRaking : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }, Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Discarded Rakings : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }
Remaining Rakings : Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Does K entail α? : Yes
Bacause J = { (e~>w), (s=>e) } is a subset of remaining Rank 2 : { (e~>w) }
Therefore Rank 2 : { (e~>w) } entails (s~>w)
It follows that K = { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) } entails (s~>w)

C:\PROJECT_UCT\DEE_Project\KLMDEETool\KLMDEETool\target>
```

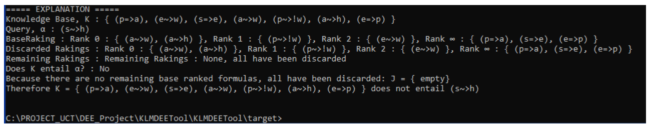
Figure 14: CLI explanation output for $\mathcal{K} \mid\approx s \mid\sim w$?

C.2 $\mathcal{K} \approx$ ***Simba*** $\mid\!\sim$ ***Hunt*** ($\mathcal{K} \approx s \mid\!\sim h$)?



```
Explanation (why and how K entails α?) :
Knowledge Base, K : { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) }
Query, α : (s~>h)
BaseRaking : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }, Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Discarded Rakings : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }, Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Remaining Rakings : Remaining Rakings : None, all have been discarded
Does K entail α? : No
Because there are no remaining base ranked formulas, all have been discarded: J = { empty}
Therefore K = { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) } does not entail (s~>h)
```

Figure 15: GUI explanation output for $\mathcal{K} \approx s \mid\!\sim h$?

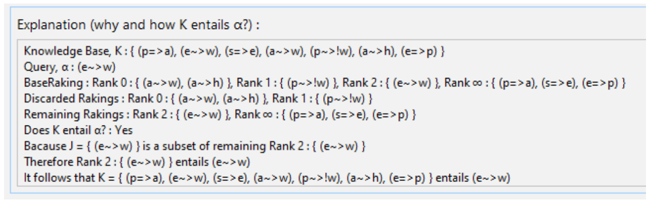


```
===== EXPLANATION =====
Knowledge Base, K : { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) }
Query, α : (s~>h)
BaseRaking : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }, Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Discarded Rakings : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }, Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Remaining Rakings : Remaining Rakings : None, all have been discarded
Does K entail α? : No
Because there are no remaining base ranked formulas, all have been discarded: J = { empty}
Therefore K = { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) } does not entail (s~>h)


C:\PROJECT_UCT\DEE_Project\KLMDEETool\KLMDEETool\target>
```

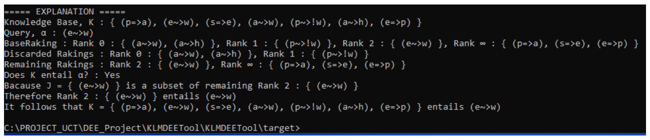
Figure 16: CLI explanation output output for $\mathcal{K} \approx s \mid\!\sim h$?

C.3 $\mathcal{K} \approx$ ***ExoticPet*** $\mid\!\sim$ ***Wild*** ($\mathcal{K} \approx e \mid\!\sim w$)?



```
Explanation (why and how K entails α?) :
Knowledge Base, K : { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) }
Query, α : (e~>w)
BaseRaking : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }, Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Discarded Rakings : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }
Remaining Rakings : Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Does K entail α? : Yes
Bacause J = { (e~>w) } is a subset of remaining Rank 2 : { (e~>w) }
Therefore Rank 2 : { (e~>w) } entails (e~>w)
It follows that K = { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) } entails (e~>w)
```

Figure 17: GUI explanation output for $\mathcal{K} \approx e \mid\!\sim w$?



```
===== EXPLANATION =====
Knowledge Base, K : { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) }
Query, α : (e~>w)
BaseRaking : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }, Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Discarded Rakings : Rank 0 : { (a~>w), (a~>h) }, Rank 1 : { (p~>!w) }
Remaining Rakings : Rank 2 : { (e~>w) }, Rank ∞ : { (p=>a), (s=>e), (e=>p) }
Does K entail α? : Yes
Bacause J = { (e~>w) } is a subset of remaining Rank 2 : { (e~>w) }
Therefore Rank 2 : { (e~>w) } entails (e~>w)
It follows that K = { (p=>a), (e~>w), (s=>e), (a~>w), (p~>!w), (a~>h), (e=>p) } entails (e~>w)

C:\PROJECT_UCT\DEE_Project\KLMDEETool\KLMDEETool\target>
```

Figure 18: CLI explanation output for $\mathcal{K} \approx e \mid\!\sim w$?