Defeasible Justification for the KLM Framework

By

Steve Wang

A thesis presented in accordance with the requirements for the degree of Masters of Science in the Faculty of Science

Declaration

I, Steve Wang, do hereby declare that this Masters thesis titled "Defeasaible Justification for the KLM Framework" is my own work and has not been submitted for any other degree award to any other University. I understand that failure to attribute material which is obtained from another source may be considered plagiarism.

Signed:

Date:

Acknowledgements

I want to thank the University of Cape Town (UCT) and the Computer Science department for giving me the opportunity and the resources to support this Master's dissertation. This dissertation is only possible with guidance from my supervisor Prof. Tommie Meyer and my co-supervisor, Prof. Deshen Moodley, both from the University of Cape Town. Lastly, I thank my family and friends who helped in achieving my goals and aided me in many aspects to complete this dissertation.

Contents

1 Introduction **2**

2 Background **5**
2.1 Overview 5
2.2 Propositional Logic 6
2.2.1 Syntax 6
2.2.2 Semantics 7
2.3 Justification for Classical Entailment 10
2.4 The KLM Framework 11
2.5 Rational Closure 11

3 Justification for Defeasible Entailment using the KLM Framework **16**
3.1 Overview 16
3.2 Justification for Defeasible Entailment in Description Logic . 19
3.3 Rational Closure Algorithm 20
3.3.1 Base Ranking for Justification 21
3.3.2 Rational Closure for Justification 23
3.4 Classical Justification Algorithm 25
3.4.1 Expand Formulas 26
3.4.2 Contract Formulas 28
3.4.3 Compute Single Justification 31
3.4.4 Compute All Justifications 32
3.5 Dematerialisation Algorithm 41
3.6 Justification Algorithm for Defeasible Entailment using the KLM Framework 41

4 Implementation of Defeasible Justification Algorithm 44
4.1 Overview 44
4.2 Input and Output Parameters 45
4.3 Software Architecture 45
4.4 External Packages 49
4.5 Algorithm Implementation 50
4.6 Testing and Evaluation 50

5 Conclusion and Future Work 53
5.1 Summary and Contribution 53
5.2 Future Work 54

Abstract

Knowledge Representation (KR) and Reasoning are essential aspects of Artificial Intelligence (AI) as they allow AI systems to conduct logical reasoning. Most classical logics, such as Propositional Logic (PL), are monotonic, which means that adding new knowledge to a knowledge base cannot cause the retraction of a previously drawn conclusion. These classical logics cannot easily handle exceptions to typical scenarios. Defeasible reasoning is a type of non-monotonic reasoning, which allows the notion of "defeasible implication". The Kraus, Lehmann, and Magidor (KLM) Framework is an extension of PL that can perform defeasible reasoning. The results of defeasible reasoning using the KLM Framework are often challenging to understand. Therefore, one needs a framework to justify conclusions drawn from defeasible reasoning.

We propose a theoretical framework for defeasible justification using the KLM Framework and a software tool that implements the framework. The theoretical framework is based on an existing theoretical framework for Description Logic (DL) which we translate to PL. The defeasible justification algorithm uses the statement ranking required by the KLM-style form of defeasible entailment, known as rational closure. Classical justifications are computed based on materialised formulas (classical counterparts of defeasible formulas). The resulting classical justifications are converted to defeasible justifications based on the input knowledge base.

We provide a software tool with a graphical user interface (GUI) that implements the algorithm. Given a defeasible knowledge base and a query, such that the knowledge base defeasibly entails the query, the program produces a set of justifications for the defeasible entailment. We use a set of representative examples to evaluate the defeasible justification algorithm and argue that its results conform to intuition. The same examples are used to confirm the correctness of the algorithm implementation.

Chapter 1

Introduction

Knowledge Representation (KR) is a field in Artificial Intelligence (AI) that is dedicated to representing knowledge systematically using symbols and operations [31]. The literature suggests various systems used to represent knowledge, and most such systems are based on logic [35, 8, 4]. Some logics are more complex than others and can represent more complex knowledge. For example, Description Logic (DL) [4] is more expressive compared to Propositional Logic (PL) [8]. The work in this dissertation is presented in PL.

Reasoning with logic allows AI systems to produce logical deductions. Various reasoning systems can be categorised based on their characteristics and attributes [34, 15, 27]. In this dissertation, we are interested in a specific property of reasoning systems known as *monotonicity*. Monotonic reasoning, also referred to as classical reasoning, has the following property: adding new knowledge cannot cause the retraction of previously drawn deductions. This characteristic of monotonic reasoning restricts the reasoning system's ability to handle exceptions to typical scenarios. Non-monotonic reasoning [29] overcomes such a restriction. In this dissertation, we look at a specific type of non-monotonic reasoning known as *defeasible reasoning* [32]. The Kraus, Lehmann, and Magidor (KLM) Framework [25] is an extension of PL equipped to perform defeasible reasoning.

The conclusions produced from reasoning with knowledge by logical deductions are known as entailments. The notion of entailment in PL is well established [8], and there are software tools that can compute entailment [13, 12]. Similarly, defeasible entailment is also well defined [25, 26]. Tools that compute defeasible entailment are proposed in recent literature [28, 30].

The conclusions produced by the defeasible reasoning tools are often difficult to understand due to the volume and complexity of the statements. Given a large set of statements, it is difficult for users to extract the exact statements used by the reasoning tools to make conclusions. Complex statements, such as nested statements, make it difficult for users to understand how the reasoning tools reach their conclusions. In this dissertation, we aim to provide an algorithm and a tool which extract the minimal subsets of the statements that contributes to the conclusion. Horridge refers to such minimal subsets as *justifications*. [21] Justifications serve as explanations for entailment by extracting the exact statements in the knowledge set used to reach conclusions. Chama formalised an algorithm that computes justifications for defeasible entailment using DL in her dissertation [11]. Currently, no defeasible justification algorithm for PL is presented in the literature, and no tool has been built to compute defeasible justification for PL. Furthermore, PL reasoners are more efficient compared to DL reasoners and as a result, they can handle a larger set of statements. For this reason, we justify the need for a defeasible justification algorithm for PL despite the existing justification algorithms for DL.

We present an example demonstrating the intuition behind defeasible entailment and defeasible justification. Consider the following set of statements:

- Penguins are birds.
- Robins are birds.
- Special penguins are penguins.
- Birds typically can fly.
- Birds typically have wings.
- Penguins typically cannot fly.
- Special penguins typically fly.

Using the KLM-style form of defeasible reasoning known as *rational closure*, the above statements defeasibly entail special penguins typically can fly. Section 3.3 demonstrates the entailment computation and suggests that the statements that birds typically fly, that birds typically have wings, and

that penguins typically cannot fly, are disregarded in the process of the computation.

Intuitively, the knowledge that special penguins typically fly should constitute a justification of that same statement (that special penguins typically fly). Based on the intuition of classical justification, one might also consider the knowledge that special penguins are penguins, that penguins are birds, and that birds typically fly to constitute a justification for the conclusion that special penguins typically fly. However, the latter set contains statements disregarded in the entailment calculation. Therefore, the former is the only valid justification for the defeasible entailment.

This dissertation presents an algorithm that computes justifications for defeasible entailment in PL. Furthermore, we present a software implementation of the algorithm. We published a paper based on this dissertation at the Southern African Conference of Artificial Intelligence 2022 (SACAIR) [37].

The dissertation is structured as follows:

Chapter 2 presents the prior background knowledge for this dissertation. The chapter presents concepts and notions that form the building blocks to formalising a defeasible justification algorithm. Topics covered in Chapter 2 includes propositional logic, justification for classical reasoning, the KLM Framework and rational closure.

Chapter 3 presents the defeasible justification algorithm for PL. The algorithm consists of several sub-algorithms. Each section in the chapter presents a sub-algorithm with demonstration examples.

Chapter 4 discusses the implementation of the algorithm presented in Chapter 3. The discussion of the implementation includes the following aspects: software architecture, external packages used, input and output parameters, algorithm implementation and, the evaluation of the implementation.

Lastly, Chapter 5 concludes the dissertation by summarising this dissertation's contributions and presenting some ideas for future work.

Chapter 2

Background

2.1 Overview

Propositional Logic (PL) [8] is the fundamental logic on which this dissertation is based. The notions of interpretation, entailment, and justification are well-established in the literature. Reasoning using PL is monotonic, meaning one cannot retract a previously drawn conclusion. In this dissertation, we also refer to such monotonic reasoning as classical reasoning.

Defeasible reasoning is an implementation of non-monotonic reasoning. The Kraus, Lehmann, and Magidor (KLM) Framework [25] is an extension of PL equipped to perform defeasible reasoning. The notion of base rank and rational closure is required to perform defeasible reasoning using the KLM Framework. Horridge defines the notion of justification for Description Logic (DL) reasoning [21]. This dissertation focuses on providing justifications for defeasible reasoning using the KLM Framework.

The concepts mentioned above and notions form the building blocks for constructing a defeasible justification algorithm. Figure 2.1 presents a visual diagram of the relations between these building blocks.

This chapter introduces PL by presenting the syntax and semantics required to represent knowledge systematically in section 2.2. The notion of classical entailment in PL leads us to present a method to justify classical entailment in section 2.3. Section 2.4 introduces the KLM Framework. Section 2.5 presents the notion of defeasible entailment and algorithms that

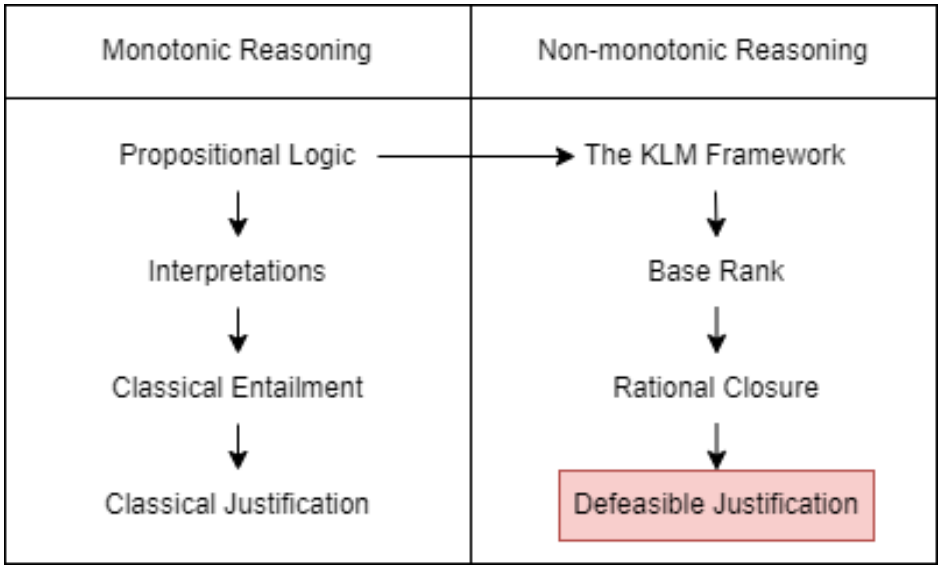

Figure 2.1: Foundational Concepts and Notions of Defeasible Justification

compute defeasible entailment.

2.2 Propositional Logic

PL is a formalisation of knowledge that forms the basis of reasoning in computer science [8]. PL performs classical reasoning, meaning one cannot retract a previously drawn conclusion. For example, consider the statements "penguins are birds", "birds can fly", and "penguins cannot fly". Classical reasoning deduces that there are no penguins. In this instance if we include the statement "there are penguins", it will contradict the deduction.

In this section, we present the syntax and semantics of PL as a foundational building block for the rest of the dissertation.

2.2.1 Syntax

Formulas in PL are constructed using the following connectives: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, and a finite set of propositional atoms. The connectives are binary operations except for $\neg$, which is a unary operator. *Propositional atoms* are

variables that can be interpreted to truth values. The set of all propositional atoms is denoted as $\mathcal{P}$. The $\top$ constant in PL denotes a tautology that is always interpreted as true, and the $\bot$ constant is always interpreted as false. An example of a formula in PL where α, β, δ and γ are propositional atoms is $(\alpha \wedge \beta) \rightarrow (\delta \vee \neg\gamma)$. It can be read as "alpha and beta implies delta or not gamma". Furthermore, the set of all formulas can be denoted as $\mathcal{L}$, and formulas in PL can be defined inductively with the following definition.

Definition 2.1 (Propositional Formulas)**.** *$\mathcal{L}$ can be inductively defined with the following rules. Let $\mathcal{P}$ be a finite set of propositional atoms then*

- *For every propositional atom $p \in \mathcal{P}$, $p \in \mathcal{L}$.*
- *For a formula $\alpha \in \mathcal{L}$, $\neg\alpha \in \mathcal{L}$.*
- *For formulas $\alpha, \beta \in \mathcal{L}$, $\alpha \wedge \beta, \alpha \vee \beta, \alpha \rightarrow \beta, \alpha \leftrightarrow \beta \in \mathcal{L}$.*
- *Only expressions given in the above three points are formulas in $\mathcal{L}$.*

2.2.2 Semantics

An *interpretation* maps propositional atoms to either *true* or *false*. This notion can be further extended to all propositional formulas using satisfiability. Formally, interpretations and satisfiability are defined as follows:

Definition 2.2 (Interpretation)**.** *An interpretation is a total function $\mathcal{I} : \mathcal{P} \mapsto \{T, F\}$ that assigns one of the truth values T or F to every $p \in \mathcal{P}$.*

Definition 2.3 (Satisfiability)**.** *An interpretation $\mathcal{I}$ satisfies some $p \in \mathcal{P}$ if $\mathcal{I}(p) = T$ and it is denoted by $\mathcal{I} \Vdash p$. An interpretation does not satisfy some $p \in \mathcal{P}$ if $\mathcal{I}(p) = F$ and it is denoted by $\mathcal{I} \nVdash p$. Satisfiability can be extended to any formula in $\mathcal{L}$ with the following criteria: For any $\alpha, \beta \in \mathcal{L}$*

- *$\mathcal{I} \Vdash \neg\alpha$ if and only if $\mathcal{I} \nVdash \alpha$*
- *$\mathcal{I} \Vdash \alpha \wedge \beta$ if and only if $\mathcal{I} \Vdash \alpha$ and $\mathcal{I} \Vdash \beta$*
- *$\mathcal{I} \Vdash \alpha \vee \beta$ if and only if either $\mathcal{I} \Vdash \alpha$ or $\mathcal{I} \Vdash \beta$ or both*
- *$\mathcal{I} \Vdash \alpha \rightarrow \beta$ if and only if $\mathcal{I} \nVdash \alpha$ or $\mathcal{I} \Vdash \beta$ or both*
- *$\mathcal{I} \Vdash \alpha \leftrightarrow \beta$ if and only if $\mathcal{I} \Vdash \alpha \rightarrow \beta$ and $\mathcal{I} \Vdash \beta \rightarrow \alpha$*

- $\mathcal{I} \Vdash \top$
- $\mathcal{I} \nVdash \bot$

An interpretation that satisfies a formula is also known as a model of that formula.

Definition 2.4 (Model)**.** *For any $\alpha \in \mathcal{L}$ and an interpretation $\mathcal{I}$, if $\mathcal{I} \Vdash \alpha$, then $\mathcal{I}$ is a model of α.*

A formula is satisfiable if it has a model. A formula is a tautology if all interpretations satisfy it. A formula α is unsatisfiable if there are no interpretations $\mathcal{I}$ such that $\mathcal{I} \Vdash \alpha$. A typical example of an unsatisfiable formula is $\alpha \wedge \neg\alpha$, for any propositional formula α.

The following is a formal definition of Tarskian's notion of logical consequence, also known as entailment.

Definition 2.5 (Classical Entailment)**.** *Let $U \subseteq \mathcal{L}$ be a set of formulas and α a formula. U entails α, denoted $U \vDash \alpha$, if and only if every model of U is a model of α.*

Entailment can also be defined in terms of *interpretations* (definition 2.2) as follows: Let $U \subseteq \mathcal{L}$ be a set of formulas and α a formula, $U \vDash \alpha$, if and only if for every interpretation $\mathcal{I}$ such that $\mathcal{I} \Vdash U$, it follows that $\mathcal{I} \Vdash \alpha$ also holds.

The notion of two formulas being logically equivalent can be understood as these two formulas entail each other or share the same models.

Definition 2.6 (Logical Equivalence)**.** *Let $\alpha_1, \alpha_2 \in \mathcal{L}$. If for every interpretation I such that $I \Vdash \alpha_1$, it also holds that $I \Vdash \alpha_2$ and for every interpretation I such that $I \Vdash \alpha_2$, it also holds that $I \Vdash \alpha_1$, then α_1 is* logically equivalent *to α_2. The notation $\alpha_1 \equiv \alpha_2$ denotes logical equivalence.*

The following set of logically equivalent formulas is often used to simplify or restructure formulas.

- $\neg(\neg\alpha) \equiv \alpha$
- $\alpha \to \beta \equiv \neg\alpha \vee \beta$
- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$

A set of finite propositional formulas is also known as a knowledge base.

Definition 2.7 (Knowledge Base)**.** *A knowledge base, denoted $\mathcal{K}$, is a finite set of propositional formulas.*

The notion of satisfiability, entailment, and logical equivalence can be extended to a knowledge base. These are defined as follows:

Definition 2.8 (Satisfiability for a Knowledge Base)**.** *A knowledge base $\mathcal{K} = \{\alpha_1, ...\alpha_n\}$ is satisfiable if and only if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \Vdash \alpha_i$ for all i where $1 \leq i \leq n$. The satisfying interpretation is a model of $\mathcal{K}$. $\mathcal{K}$ is unsatisfiable if and only if, for every interpretation, $\mathcal{I}$, there exists i such that $\mathcal{I} \nVdash \alpha_i$.*

In this dissertation, an interpretation $\mathcal{I}$ that satisfies a knowledge base $\mathcal{K}$ is denoted by $\mathcal{I} \Vdash \mathcal{K}$. It follows from definition 2.5 and definition 2.7 that $\mathcal{K} \vDash \alpha$ if and only if every model of $\mathcal{K}$ is a model of α.

Definition 2.9 (Logical Equivalence for a Knowledge Base)**.** *Given two knowledge bases, $\mathcal{K}_1, \mathcal{K}_2 \in \mathcal{L}$. $\mathcal{K}_1$ is logically equivalent to $\mathcal{K}_2$, denoted $\mathcal{K}_1 \equiv \mathcal{K}_2$, if and only if for every interpretation $\mathcal{I}$ such that $\mathcal{I} \Vdash \mathcal{K}_1$ then $\mathcal{I} \Vdash \mathcal{K}_2$ and for every interpretation $\mathcal{I}$ such that $\mathcal{I} \Vdash \mathcal{K}_2$ then $\mathcal{I} \Vdash \mathcal{K}_1$.*

For a knowledge base $\mathcal{K}$ and a propositional formula α, $\mathcal{K} \vDash \alpha$ if and only if $\mathcal{K} \cup \{\neg\alpha\}$ is unsatisfiable [6]. For example, consider the following knowledge base $\mathcal{K} = \{robins \rightarrow birds, birds \rightarrow fly\}$ and the query $robins \rightarrow fly$. The entailment $\mathcal{K} \vDash robins \rightarrow fly$ holds if and only if the set $\{robins \rightarrow birds, birds \rightarrow fly, \neg(robins \rightarrow fly)\}$ is unsatisfiable. Notice that $\neg(robins \rightarrow fly) \equiv robins \wedge \neg fly$. Therefore, the entailment $\mathcal{K} \vDash robins \rightarrow fly$ holds because the set $\{robins \rightarrow birds, birds \rightarrow fly, robins \wedge \neg fly\}$ is unsatisfiable.

The above example demonstrates that an entailment problem of checking if α is entailed by $\mathcal{K}$ can be restructured into a satisfiability problem of checking the satisfiability of $\mathcal{K} \cup \{\neg\alpha\}$.

Currently, there exist entailment checking tools such as SAT Solvers [17, 14].

2.3 Justification for Classical Entailment

Conclusions drawn from entailment-checking tools are often difficult to understand. There are several approaches to computing explanations for entailments [5, 21]. Baader et al. proposed an axiom pinpointing algorithm as the extension of the standard tableau-based reasoning algorithms for computing consequences from DL [5]. Horridge proposed the notion of *justification* to explain the deductions of entailments [21]. The intuition behind a *justification* given a knowledge base $\mathcal{K}$ and a query α such that $\mathcal{K} \models \alpha$ is the following: a *justification* $\mathcal{J} \subseteq \mathcal{K}$ is a subset of $\mathcal{K}$ such that $\mathcal{J} \models \alpha$ and there is no proper subset $\mathcal{J}' \subset \mathcal{J}$ such that $\mathcal{J}' \models \alpha$. Horridge defines *justification* in Description Logic (DL), but in the context of this dissertation, *justification* is defined in PL as follows:

Definition 2.10 (Classical Justification)**.** *Let $\mathcal{K}$ be a knowledge base and a query α such that $\mathcal{K} \models \alpha$. The set of formulas $\mathcal{J}$ is a justification for $\mathcal{K} \models \alpha$ if $\mathcal{J} \subseteq \mathcal{K}$, $\mathcal{J} \models \alpha$ and for all $\mathcal{J}' \subset \mathcal{J}$, it holds that $\mathcal{J}' \not\models \alpha$.*

Consider the knowledge base $\mathcal{K} = \{robins \rightarrow birds, birds \rightarrow fly, penguins \rightarrow birds, birds \rightarrow wings\}$, and the query $robins \rightarrow fly$. The justification for the entailment $\mathcal{K} \models robins \rightarrow fly$ is the set $\mathcal{J} = \{robins \rightarrow birds, birds \rightarrow fly\}$. The set $\mathcal{J}$ is a justification because $\mathcal{J} \models robins \rightarrow fly$, and if any formula is removed for $\mathcal{J}$, then $\mathcal{J}$ no longer entails the query.

In this dissertation, we further define justification for a query α that is not entailed by a knowledge base $\mathcal{K}$. The justification for a non-entailment is simply the set that does not entail the query. In classical reasoning, the justification for $\mathcal{K} \not\models \alpha$ is the knowledge base itself.

Definition 2.11 (Justification for Non-entailment)**.** *Let $\mathcal{K}$ be a knowledge base and a query α such that $\mathcal{K} \not\models \alpha$. The justification for $\mathcal{K} \not\models \alpha$ is $\mathcal{K}$.*

Consider the knowledge base $\mathcal{K} = \{robins \rightarrow birds, birds \rightarrow fly, penguins \rightarrow \neg fly\}$ and the query $penguins \rightarrow birds$. $\mathcal{K} \not\models penguins \rightarrow birds$, and the justification is simply $\mathcal{K}$.

Horridge proposed algorithms to compute justification for entailments in DL [21]. We translate Horridge's algorithm to PL, presented in section 3.6 of this dissertation.

2.4 The KLM Framework

Consider the following knowledge base $\mathcal{K} = \{birds \rightarrow fly, penguins \rightarrow birds, penguins \rightarrow \neg fly\}$. Classical reasoning concludes $\mathcal{K} \models \neg penguins$, which means there are no penguins. However, we know that penguins are birds that cannot fly. Penguins are an exception to the typical scenario where birds can fly.

Non-monotonic reasoning overcomes such a shortfall of monotonic reasoning. In this dissertation, we will look at a specific type of non-monotonic reasoning known as defeasible reasoning. The KLM Framework provides a preferential approach to defeasible reasoning [25].

The KLM Framework extends PL with the additional binary operation of *defeasible implication*, denoted by $\mid\sim$. The formula $\alpha \mid\sim \beta$ is read as "alpha typically implies beta". Defeasible implication operations cannot be nested. For example, one cannot have the formula $(\alpha \mid\sim \beta) \mid\sim \gamma$.

In this dissertation, we define a finite set of formulas containing defeasible implications as a *defeasible knowledge base*.

The *material counterpart* of a defeasible implication $\alpha \mid\sim \beta$ is the propositional formula $\alpha \rightarrow \beta$. Alternatively, $\alpha \rightarrow \beta$ is the *materialisation* of $\alpha \mid\sim \beta$. Given a defeasible knowledge base $\mathcal{K}$, $\overrightarrow{\mathcal{K}}$ is the *materialisation* of $\mathcal{K}$ if all defeasible implications $\alpha \mid\sim \beta \in \mathcal{K}$ are replaced with its material counterpart $\alpha \rightarrow \beta$.

2.5 Rational Closure

A defeasible knowledge base is a finite set of defeasible implications. All classical implications can be replace with corresponding defeasilbe implications [10]. For example, the classical implication $\alpha \rightarrow \beta$ can be expressed as the defeasible implication $\neg(\alpha \rightarrow \beta) \mid\sim \bot$. Therefore a knowledge base that contains classical implications can be converted into a knowledge base that contains defeasible implications only. In this section, we assume knowledge bases are defeasible unless specified otherwise.

Kraus, Lehmann, and Magidor presented the notion of *preferential models* [25]. Lehmann and Magidor defined *preferential entailment* based on preferential models, denoted $\mid\approx_p$ [26]. Furthermore, they defined the notion

of exceptionality using preferential entailment. Preferential entailment is not good enough to perform defeasible reasoning because it is monotonic [26]. For example, given the statements robins are birds and birds typically have wings, preferential entailment cannot entail the statement robins typically have wings. Therefore, Lehmann and Magidor defined rational closure, denoted $\mid\!\!\approx_{RC}$, as a form of defeasible reasoning [26]. Corollary 6 of [26] shows that exceptional formulas in the knowledge base can be identified using classical entailment instead of preferential entailment. Preferential entailment forms the core of computing rational closure. Freund presented an algorithm that computes rational closure, denoted by $\vdash_{RC}$ [16]. The algorithm $\vdash_{RC}$ takes a finite defeasible knowledge base $\mathcal{K}$ and a defeasible implication $\alpha \mid\sim \beta$ as inputs. The result of $\vdash_{RC}$ is **true** if and only if $\mathcal{K} \mid\!\!\approx_{RC} \alpha \mid\sim \beta$. The algorithm iteratively identifies a set of exceptional formulas. More specifically, $E_{i+1} = \{\alpha \mid\sim \beta \in E_i | E_i \mid\!\!\approx_p \neg\alpha\}$ is a set of exceptional formulas in E_i. Freund uses classical entailment to identify the set of exceptional formulas as $E_{i+1} = \{\alpha \mid\sim \beta \in E_i | \overrightarrow{E_i} \vDash \neg\alpha\}$ [16]. For simplicity, in the remainder of this dissertation, we use the symbol $\mid\!\!\approx$ to denote defeasible entailment.

Freund presented an algorithm that ranks formulas required by *rational closure* [16]. We refer to this specific ranking of formulas as **base rank**. The **base rank** of the formulas in a knowledge base is unique [19]. The intuition behind **base rank** is that the lower-ranked formulas are less exceptional.

Casini et al. formalised the **base rank** and **rational closure** algorithms into pseudocode, as shown below in algorithm 1 and algorithm 2, respectively [10].

Algorithm 1 computes the base rank of formulas in a given knowledge $\mathcal{K}$ in an iterative approach. In each iteration, the algorithm collects the exceptional formulas from E_i into E_{i+1} and assigns the remaining formulas in E_i to rank R_i. The algorithm continues to do so until there are no more exceptional formulas.

As shown in algorithm 2, given the base ranking $(\mathcal{R}_0, \mathcal{R}_1, ..., \mathcal{R}_\infty)$ of the formulas in a knowledge base $\mathcal{K}$ and a query $\alpha \mid\sim \beta$, *rational closure* finds the largest subset $\mathcal{S} \subseteq (\mathcal{R}_0 \cup \mathcal{R}_1 \cup ... \cup \mathcal{R}_\infty)$ such $\mathcal{S} \vDash \alpha$ by iteratively removing lower ranked formulas. The defeasible entailment $\mathcal{K} \mid\!\!\approx \alpha \mid\sim \beta$ holds if $\mathcal{S} \vDash \alpha \rightarrow \beta$.

To demonstrate the base ranking algorithm (algorithm 1), we will use some notations to denote the different interpretations of the knowledge base. For any propositional atom α, α denotes the interpretation where α is true and, $\overline{\alpha}$ denotes the interpretation where α is false. For example, $\overline{\alpha}\beta\overline{\gamma}$ denotes

the interpretation where α and γ are false and β is true.

Algorithm 1 BaseRank

1: Input: A knowledge base $\mathcal{K}$
2: Output: An ordered tuple $(R_0, ..., R_{n-1}, R_\infty, n)$
3: i := 0;
4: $E_0 := \overrightarrow{\mathcal{K}}$;
5: **while** $E_{i-1} \neq E_i$ **do**
6: $\quad E_{i+1} := \{\alpha \rightarrow \beta \in E_i \mid E_i \models \neg\alpha\}$;
7: $\quad R_i := E_i \setminus E_{i+1}$;
8: $\quad i := i + 1$;
9: **end while**
10: $R_\infty := E_{i-1}$;
11: **if** $E_{i-1} == \varnothing$ **then**
12: $\quad n := i - 1$;
13: **else**
14: $\quad n := i$;
15: **end if**
16: **return** $(R_0, ..., R_{n-1}, R_\infty, n)$

The following example from Kaliski's dissertation demonstrates the intuition behind the base ranking [22].

1. Consider the following knowledge base $\mathcal{K} := \{boat \mid\sim floats, leaky \mid\sim boat, leaky \mid\sim \neg floats, FlyingDutchman \mid\sim boat, FlyingDutchman \mid\sim leaky\}$. For simplicity, we represent the propositional atoms using letters as follows: b for $boat$, f for $floats$, l for $leaky$ and d for $FlyingDutchman$.

2. The material counterpart of the knowledge base $\mathcal{K} := \{b \mid\sim f, l \mid\sim b, l \mid\sim \neg f, d \mid\sim b, d \mid\sim l\}$ is $\overrightarrow{\mathcal{K}} = \{b \rightarrow f, l \rightarrow b, l \rightarrow \neg f, d \rightarrow b, d \rightarrow l\}$. Therefore, $E_0 = \{b \rightarrow f, l \rightarrow b, l \rightarrow \neg f, d \rightarrow b, d \rightarrow l\}$.

3. Applying line 6 of the algorithm, we have $E_1 := \{\alpha \rightarrow \beta \in E_0 \mid E_0 \models \neg\alpha\}$. Only three interpretations satisfy E_0: $b\bar{l}f\bar{d}$, $\overline{blfd}$ and $\overline{bl}f\bar{d}$. All three interpretations also satisfy $\bar{l}$ and $\bar{d}$. Therefore, $E_0 \models \neg l$, $E_0 \models \neg d$ and $E_1 := \{l \rightarrow b, l \rightarrow \neg f, d \rightarrow b, d \rightarrow l\}$. Line 7 of the algorithm defines $R_1 := \{b \rightarrow f\}$.

4. The second iteration of the while loop defines $E_2 \coloneqq \{\alpha \rightarrow \beta \in E_1 \mid E_1 \models \neg\alpha\}$.The following interpretations satisfy E_1: $b\overline{l}\overline{f}d$, $b\overline{l}f\overline{d}$, $bl\overline{f}\overline{d}$, $\overline{bl}f\overline{\mathrm{d}}$, $bl\overline{fd}$ and $\overline{bfl}d$. From these interpretations, we see that $E_1 \not\models \neg l$ and $E_1 \not\models \neg d$. Hence, $E_2 \coloneqq \varnothing$ and $R_1 \coloneqq \{l \rightarrow b, l \rightarrow \neg f, d \rightarrow b, d \rightarrow l\}$.

5. It is easy to see that the third iteration of the while loop results in $E_3 \coloneqq \varnothing$. The loop terminates because $E_2 = E_3$. Line 10 defines $R_\infty \coloneqq \varnothing$. The if-else statement from lines 10 to 15 assigns $n = 3$.

6. Finally, the algorithm returns the tuple: $(\{b \rightarrow f\}, \{l \rightarrow b, l \rightarrow \neg f, d \rightarrow b, d \rightarrow l\}, \varnothing, 3)$. More visually, one can view the rankings of $\vec{\mathcal{K}}$ in the following table.

R_∞	$\varnothing$
R_1	$l \rightarrow b, l \rightarrow \neg f, d \rightarrow b, d \rightarrow l$
R_0	$b \rightarrow f$

Algorithm 2 RationalClosure

```
1: Input: A knowledge base 𝒦, and a defeasible implication α |~ β
2: Output: true if 𝒦 |≈ α |~ β, and false otherwise
3: (R_0, ..., R_{n-1}, R_∞, n) := BaseRank(𝒦);
4: i := 0
5: R := ⋃_{i=0}^{j<n} R_j;
6: while R_∞ ∪ R ⊨ ¬α and R ≠ ∅ do
7:     R := R ∖ R_i;
8:     i := i + 1;
9: end while
10: return R_∞ ∪ R ⊨ α → β;
```

We continue from the above base rank example to illustrate the intuition behind rational closure. Additionally, we use the defeasible implication $d \mathrel{|\!\sim} \neg f$ as the query.

1. We use the output of the base ranked example in line 3 of algorithm 2. Line 5 of the algorithm defines $R \coloneqq \bigcup_{i=0}^{j<n} R_j$. Apply this to the result of base ranking. We have $R \coloneqq \{b \rightarrow f, l \rightarrow b, l \rightarrow \neg f, d \rightarrow b, d \rightarrow l\}$.

2. The conditions on line 6 of the algorithm require us to determine $R_\infty \cup R \models \neg d$. As no model of $R_\infty \cup R$ satisfies d, we have $R_\infty \cup R \models \neg d$. Line 7 removes R_0 from R. Therefore, $R = \{l \to b, l \to \neg f, d \to b, d \to l\}$.

3. The interpretation $bl\overline{f}d$ satisfies R and d. Therefore, $R_\infty \cup R \not\models \neg d$ and the algorithm exit the while loop. Lastly, we must determine if $R_\infty \cup R \models d \to \neg f$. The interpretation $bl\overline{f}d$ also satisfies $d \to \neg f$. Hence, $R_\infty \cup R \models d \to \neg f$ is true. The algorithm returns true, and indeed $\mathcal{K} \mid\!\approx d \mid\!\sim \neg f$. In English, the knowledge base $\mathcal{K}$ defeasible entails that the Flying Dutchman typically does not float.

Chapter 3

Justification for Defeasible Entailment using the KLM Framework

3.1 Overview

In this chapter, we present the theoretical work of this dissertation. More specifically, we propose a defeasible justification for the KLM Framework, which we refer to as **KLMDefeasibleJustification**. We first look at the definition of defeasible justification. Then we investigate a defeasible justification algorithm for DL proposed by Chama [11]. With that, we translate the defeasible algorithm to PL. We then present an example illustrating the intuition of defeasible justification for PL.

In subsection 2.3, we presented Horridge's definition of justifications which are the minimal sets in the knowledge base that entails the query. Horridge presented an algorithm that identifies the justification for a classical entailment. However, justification for defeasible entailment is a layer more complex.

Chama defined defeasible justification for DL as follows: for a justification of a defeasible entailment, $\mathcal{K} \mid\!\sim \alpha$, we are only selecting the minimal sets in $\mathcal{K}$ such that (1) the materialisation of the minimal set classically entails $\overrightarrow{\alpha}$ and (2) the minimal sets only contain the appropriate and applicable axioms. Such appropriate and applicable axioms are those axioms that were not disregarded when computing rational closure [11].

Observing the transition from classical reasoning to defeasible reasoning presents another approach to understanding defeasible justification. Initially, in the classical case, we have justifications (minimal sets) that support a classical entailment. When we move from classical reasoning to defeasible reasoning, we select the justifications that only contain appropriate and applicate axioms. Of course, certain axioms in the defeasible justifications need to be transformed to the defeasible version based on the defeasible knowledge base.

Chama presented a defeasible justification algorithm for DL [11]. Based on the definition of defeasible justification, Chama modified the rational closure algorithm for DL to return an integer that indicates the number of ranks of axioms disregarded. Chama's defeasible justification algorithm omits ranks of axioms according to the rational closure algorithm. The defeasible justification algorithm for DL also uses the ComputeAllJustifications algorithm as a sub-procedure. Horridge proposed the ComputeAllJustifications algorithm to identify all classical justifications for an entailment in DL [21]. Applying the ComputeAllJustifications algorithm to the appropriate and applicable axioms in the knowledge base produces defeasible justifications.

We take a similar approach to construct a defeasible justification algorithm for PL. Firstly, we modify the RationalClosure algorithm for PL (algorithm 2) to return an additional integer value that indicates the number of ranks of formulas disregarded. Secondly, we apply the PL version of ComputeAllJustification to the appropriate and applicable formulas. Since ComputeAllJustification produces classical justification, we transform the classical justifications into defeasible justifications as the last procedure.

To illustrate the intuition of the defeasible justification algorithm for PL, we present the following example: Consider the classical knowledge base $\mathcal{K}$ with the following PL formulas, $\mathcal{K} = \{penguins \rightarrow birds, robins \rightarrow birds, specialpenguins \rightarrow penguins, birds \rightarrow fly, birds \rightarrow wings, special-penguins \rightarrow fly\}$ with the query $specialpenguins \rightarrow fly$. There are two possible classical justifications for $\mathcal{K} \models specialpenguins \rightarrow fly$: $\mathcal{J}_1 = \{special-penguins \rightarrow fly\}$ and $\mathcal{J}_2 = \{SpecialPenguins \rightarrow penguins, penguins \rightarrow birds, birds \rightarrow fly\}$. Notice specialpenguins are unsatisfiable because classical reasoning deduces specialpenguins cannot fly (based on specialpenguins are penguins and penguins cannot fly), but the knowledge base also states that specialpenguins can fly.

In the defeasible case, consider the defeasible knowledge base $\mathcal{K} = \{penguins \rightarrow birds, robins \rightarrow birds, specialpenguins \rightarrow penguins, birds \mathrel{|\!\sim} fly, birds \mathrel{|\!\sim} wings, penguins \mathrel{|\!\sim} \neg fly, specialpenguins \mathrel{|\!\sim} fly\}$ with the query $specialpenguins \mathrel{|\!\sim} fly$. According to the BaseRank algorithm (algorithm 1), we should get the following ranking of formulas:

R_∞	$penguins \rightarrow birds, robins \rightarrow birds, specialpenguins \rightarrow penguins$
R_2	$specialpenguins \mathrel{|\!\sim} fly$
R_1	$penguins \mathrel{|\!\sim} \neg fly$
R_0	$birds \mathrel{|\!\sim} fly, birds \mathrel{|\!\sim} wings$

Provided the above formula ranking, the RationalClosure algorithm (algorithm 2) concludes that $\mathcal{K} \models specialpenguins \mathrel{|\!\sim} fly$. Furthermore, the RationalClosure algorithm disregards formulas in $\mathcal{R}_0$ and $\mathcal{R}_1$.

One would think both $\mathcal{J}_1 = \{specialpenguins \mathrel{|\!\sim} fly\}$ and $\mathcal{J}_2 = \{special-penguins \rightarrow penguins, penguins \mathrel{|\!\sim} birds, birds \mathrel{|\!\sim} fly\}$ are justifications supporting the entailment. However, $\mathcal{J}_2$ includes $birds \mathrel{|\!\sim} fly$, which was disregarded in the rational closure computation. Therefore, only $\mathcal{J}_1$ is the valid defeasible justification as it only contains the appropriate and applicable formula.

For simplicity, in the remaining chapters of this dissertation, we refer to defeasible justification as *justification* and assume all knowledge bases are defeasible unless specified otherwise.

The **KLMDefeasibleJustification** algorithm is composed of multiple sub-algorithms. Figure 3.1 shows the composition of the justification algorithm. It first computes the defeasible entailment with **RationalClosure**, which is a slightly adjusted version of algorithm 2. This is followed by a sub-algorithm that **finds all justifications** (definition 2.10) for classical entailment (definition 2.5). Lastly, the classical justifications are converted to defeasible justifications using the **dematerialisation** algorithm.

This chapter is structured as follows. We first look at the composition of Chama's defeasible justification algorithm in section 3.2. Then we present the sub-algorithms **RationalClosure**, **FindAllClassicalJustifications**, and **Dematerialisation** in sections 3.3, 3.4. and 3.5, respectively. Lastly, we present the justification algorithm and some representative examples in section 3.6.

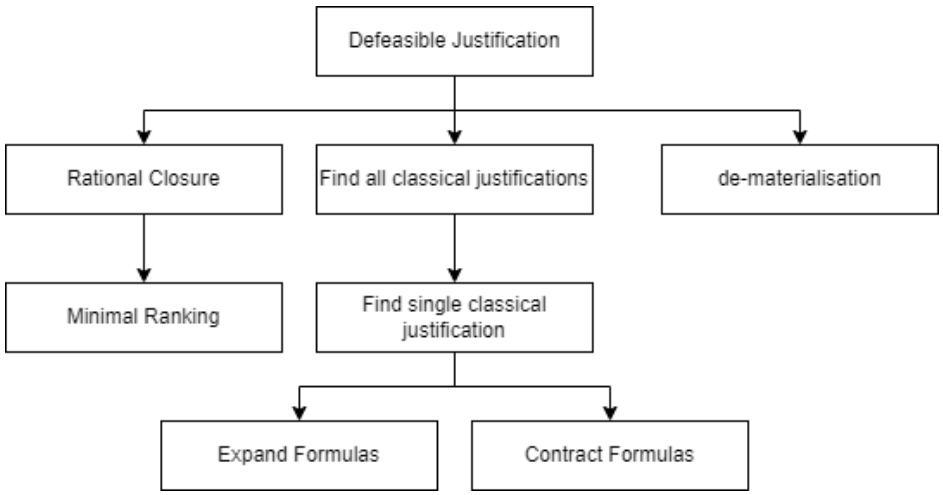

Figure 3.1: Defeasible Justification Algorithm Composition

3.2 Justification for Defeasible Entailment in Description Logic

Chama's defeasible justification algorithm uses two sub-algorithms, namely **RationalClosureForJustification** and **ComputeAllJustification**. The functionality of each sub-algorithm is as follows:

- **RationalClosureForJustification**($\mathcal{K}, \alpha \mid\sim \beta$) computes the defeasible entailment given a knowledge base $\mathcal{K}$ and a query $\alpha \mid\sim \beta$. The sub-algorithm returns the boolean value *true* if $\mathcal{K} \mid\approx \alpha \mid\sim \beta$ and *false*, otherwise. Additionally, it returns an integer value, *rank*, which indicates the number of rank of formulas disregarded when computing the entailment.

- **ComputingAllJustifications**($\mathcal{K}, \alpha \rightarrow \beta$) computes all justification(s) for the classical entailment $\mathcal{K} \models \alpha \rightarrow \beta$. This sub-algorithm returns a set of classical justifications. Each justification is a minimal subset of the knowledge base that classically entails the query.

Algorithm 3 is a simplified version of the justification algorithm for DL presented by Chama [11]. The simplification removed the explicit details that only apply to DL, which is irrelevant to this dissertation. For clarification, Chama uses $\mathcal{D}_i$ to denote the formula(s) ranked i by base ranking.

As shown in lines 10 - 12 in algorithm 3, if there are rank(s) of formulas disregarded when computing the rational closure, then the same rank(s) of formulas are also disregarded when computing the justification. We follow the same approach to formulate a defeasible justification algorithm using the KLM Framework. This is further discussed in the rest of this chapter.

Algorithm 3 ComputeDefeasibleJustification($\mathcal{K}, \alpha$)

1: Input: Knowledge base $\mathcal{K}$ and query α
2: Output: Justification $\mathcal{J}$
3: $i \leftarrow 0$
4: $\mathcal{J}_{result} \leftarrow \varnothing$
5: rank $\leftarrow$ **RationalClosureForJustification**($\mathcal{K}, \alpha$)
6: **if** rank $= 0$ **then**
7: $\quad \mathcal{J}_{result} \leftarrow$ **ComputeAllJustifications**($\mathcal{K}, \alpha$)
8: $\quad$ **return** $\mathcal{J}_{result}$
9: **end if**
10: **while** $i < rank$ **do**
11: $\quad \mathcal{K}_{new} \leftarrow \mathcal{K}_{new} \setminus \mathcal{D}_i$
12: $\quad i \leftarrow i + i$
13: **end while**
14: $\mathcal{J}_{result} \leftarrow$ **ComputeAllJustification**($\mathcal{K}_{new}, \alpha$)
15: **return** $\mathcal{J}_{result}$

3.3 Rational Closure Algorithm

As mentioned in section 2.5, Freund presented a set of systematic procedures which ranks formulas as required by rational closure [16]. Casini et al. formalised Freund's procedures into programming-style algorithms **BaseRank** (algorithm 1) and **RationalClosure** (algorithm 2) [10]. However, the **RationalClosure** algorithm only returns a boolean value indicating whether the defeasible entailment $\mathcal{K} \mid\!\approx \alpha \mid\!\sim \beta$ holds or not. The **KLMDefeasibleJustification** algorithm requires **RationalClosure** to return additional values to compute justifications. We refer to the adjusted **RationalClosure** algorithm as **RationalClosureForJustification**. Given a knowledge base $\mathcal{K}$ and a query $\alpha \mid\!\sim \beta$ as inputs, the **RationalClosureForJustification** algorithm needs to return the following values:

- **Defeasible entailment indicator** is a boolean value that is *true* if $\mathcal{K} \mid\!\approx \alpha \mid\!\sim \beta$ and *false*, otherwise.

- **Rank indicator** is an integer value which refers to the number of rank(s) of formula(s) that is disregarded in computing the defeasible entailment.

- **Ranked formulas** is an ordered tuple $(\mathcal{R}_0, ..., \mathcal{R}_n, \mathcal{R}_\infty)$ which is the minimal ranked formulas in $\mathcal{K}$.

The **BaseRank** algorithm can be further improved by applying rules to formulas that are always assigned to the infinity rank. The following subsections present the **BaseRankForJustification** and **RationalClosureForJustification** algorithms, respectively.

3.3.1 Base Ranking for Justification

The **BaseRank** algorithm (algorithm 1) assumes the input knowledge base contains only defeasible implications.

Given a knowledge base $\mathcal{K}$, consider the following two scenarios and observations on each scenario:

- Let α be a classical implication and $\alpha \in \mathcal{K}$. It follows from the theorem **??** that we can replace α with $\neg\alpha \mid\!\sim \bot$ in $\mathcal{K}$. Consider $\mathcal{K}$ as input to algorithm 1 then $\neg\alpha \rightarrow \bot \in E_0$ and for some i, $\neg\alpha \rightarrow \bot \in E_i$. Since the classical entailment $\neg\alpha \rightarrow \bot \models \alpha$ always holds, line 6 of the **BaseRank** algorithm puts $\neg\alpha \rightarrow \bot$ in E_{i+1}. As a result of this observation, we can deduce that $\neg\alpha \rightarrow \bot$ is always assigned to $\mathcal{R}_\infty$. Therefore, we conclude that **BaseRank** assigns any classical implication α to $\mathcal{R}_\infty$.

- Let $\{\alpha \mid\!\sim \beta, \alpha \mid\!\sim \neg\beta\} \subseteq \mathcal{K}$ and consider $\mathcal{K}$ as input to algorithm 1. Then $\{\alpha \rightarrow \beta, \alpha \rightarrow \neg\beta\} \subseteq E_0$ and for some i, $\{\alpha \rightarrow \beta, \alpha \rightarrow \neg\beta\} \subseteq E_i$. Since, $E_i \models \neg\alpha$ always holds, line 6 of **BaseRank** always adds $\alpha \rightarrow \beta$ and $\alpha \rightarrow \neg\beta$ to E_{i+1}. From this observation, we can conclude that if both $\alpha \mid\!\sim \beta$ and $\alpha \mid\!\sim \neg\beta$ are present in $\mathcal{K}$, then **BaseRank** algorithm always assigns these formulas to $\mathcal{R}_\infty$.

Based on the above observations and conclusions, we can construct a **BaseRankForJustification** algorithm that is more computationally efficient than **BaseRank**.

Algorithm 4 BaseRankForJustification

Input: Knowledge base $\mathcal{K}$
Output: An ordered tuple $\{\mathcal{R}_0, ..., \mathcal{R}_\infty\}$
$\mathcal{C} := \{\alpha \to \beta \in \mathcal{K}\} \cup \{\alpha \to \beta, \alpha \to \neg\beta \in \mathcal{K} | \alpha \nvdash \beta \text{ and } \alpha \nvdash \neg\beta \in \mathcal{K}\}$
$E_0 := \mathcal{K} \setminus \mathcal{C}$
$i := 0$
while $E_{i-1} \neq E_i$ **do**
    $E_{i+1} := \{\alpha \nvdash \beta \in E_i | \overrightarrow{E_i} \cup \mathcal{C} \vDash \neg\alpha\}$
    $\mathcal{R}_i := E_i \setminus E_{i+1}$
    $i = i + 1$
end while
$\mathcal{R}_\infty := \mathcal{C} \cup E_{i-1}$
if $E_{i-1} = \varnothing$ **then**
    $n = i - 1$
else
    $n = i$
end if
Return $(\mathcal{R}_0, ..., \mathcal{R}_n, \mathcal{R}_\infty)$

We present an example to illustrate the intuition behind algorithm 4 and base ranking. Consider the following knowledge base as the input to algorithm 4: $\mathcal{K} = \{penguins \to birds, robins \to birds, specialpenguins \to penguins, birds \nvdash fly, birds \nvdash wings, penguins \nvdash \neg fly, specialpenguins \nvdash fly\}$. The variables are assigned the following values as we run through the algorithm:

- $\mathcal{C} := \{penguins \to birds, robins \to birds, specialpenguins \to penguins\}$
- $E_0 := \{birds \nvdash fly, birds \nvdash wings, penguins \nvdash \neg fly, specialpenguins \nvdash fly\}$
- $E_1 := \{penguins \nvdash \neg fly, specialpenguins \nvdash fly\}$
- $\mathcal{R}_0 := \{birds \nvdash fly, birds \nvdash wings\}$
- $E_2 := \{specialpenguins \nvdash fly\}$
- $\mathcal{R}_1 := \{birds \nvdash fly\}$

Table 3.1: Base Rank of formulas in $\mathcal{K}$

Rank	Formulas
∞	$penguins \rightarrow birds, robins \rightarrow birds, specialpenguins \rightarrow penguins$
R_2	$specialpenguins \mid\sim fly$
R_1	$penguins \mid\sim \neg fly$
R_0	$birds \mid\sim fly, birds \mid\sim wings$

- $E_3 \coloneqq \varnothing$
- $\mathcal{R}_2 \coloneqq \{birds \mid\sim fly\}$
- $\mathcal{R}_\infty \coloneqq \{penguins \rightarrow birds, robins \rightarrow birds, specialpenguins \rightarrow penguins\}$

The base ranking of $\mathcal{K}$ can be visualised as shown in table 3.1.

3.3.2 Rational Closure for Justification

Similar to Chama's modification to the rational closure algorithm to return an additional integer which indicates the number of ranks of formulas disregarded. We adjust the RationalClosure algorithm (algorithm 2) to meet the required information for **KLMDefeasibleJustification**. We refer to the adjusted algorithm as **RationalClosureForJustification**.

Algorithm 5 RationalClosureForJustification

Input: A knowledge base $\mathcal{K}$, and a defeasible implication $\alpha \mid\sim \beta$
Output: **true** if $\mathcal{K} \mid\approx \alpha \mid\sim \beta$, and **false** otherwise, rank i and the tuple of base ranked formulas $(R_0, ..., R_\infty)$
$(R_0, ..., R_n, R_\infty) :=$ BaseRankForJustification($\mathcal{K}$);
$i := 0$
$R := \bigcup_{i=0}^{j \leq n} R_j$
while $\overrightarrow{R_\infty} \cup \overrightarrow{R} \models \neg\alpha$ and $R \neq \varnothing$ **do**
    $R := R \setminus R_i$
    $i := i + 1$
end while
return $\overrightarrow{R_\infty} \cup \overrightarrow{R} \models \alpha \rightarrow \beta$, i, $(R_0, ..., R_n, R_\infty)$

We illustrate the intuition behind algorithm 5 and rational closure by presenting representative examples. Consider the example knowledge base $\mathcal{K}$ and base ranking of $\mathcal{K}$ from section 3.3.1. We have $(\{birds \mathrel{|\!\sim} fly, birds \mathrel{|\!\sim} wings\}, \{penguins \mathrel{|\!\sim} \neg fly\}, \{specialpenguins \mathrel{|\!\sim} fly\}, \{penguins \rightarrow birds, robins \rightarrow birds, specialpenguins \rightarrow penguins\})$ as output from **BaseRankForJustification** sub-algorithm on line 3 of algorithm 5.

We present the three different defeasible implications as input queries to algorithm 5:

1. $robins \mathrel{|\!\sim} wings$
   - $\overrightarrow{R} = \{specialpenguins \rightarrow fly, penguins \rightarrow \neg fly, birds \rightarrow fly, birds \rightarrow wings\}$ and $\overrightarrow{R_\infty} \cup \overrightarrow{R} \models \neg robins$ does not hold. Therefore, no formulas are discarded.
   - Since, $\overrightarrow{R_\infty} \cup \overrightarrow{R} \models robins \rightarrow wings$, we have $\mathcal{K} \mid\!\approx robins \mathrel{|\!\sim} wings$.
   - The algorithm returns $i = 0$ because no formulas were discarded.
2. $penguins \mathrel{|\!\sim} wings$
   - $\overrightarrow{R} = \{specialpenguins \rightarrow fly, penguins \rightarrow \neg fly, birds \rightarrow fly, birds \rightarrow wings\}$ and $\overrightarrow{R_\infty} \cup \overrightarrow{R} \models \neg penguins$. Therefore, R_0 is removed from R and $R = \{penguins \mathrel{|\!\sim} \neg fly, specialpenguins \mathrel{|\!\sim} fly\}$.
   - Now, $\overrightarrow{R_\infty} \cup \overrightarrow{R} \models \neg penguins$ does not hold.
   - $\overrightarrow{R_\infty} \cup \overrightarrow{R} \not\models penguins \rightarrow wings$ and therefore $\mathcal{K} \not\mid\!\approx penguins \mathrel{|\!\sim} wings$.
   - The algorithm returns $i = 1$ because there is one iteration of removing the bottom-ranked formulas from $\mathcal{R}$.
3. $specialpenguins \mathrel{|\!\sim} fly$
   - Iterating over lines 6 to 9 of algorithm 5, both R_0 and R_1 is removed from R such that the condition $R_\infty \cup \overrightarrow{R} \models \neg\alpha$ does not hold.
   - $R_\infty \cup \overrightarrow{R} = \{penguins \rightarrow birds, robins \rightarrow birds, specialpenguins \rightarrow penguins, specialpenguins \rightarrow fly\} \models specialpenguins \rightarrow fly$.
   - Hence, the defeasible entailment $\mathcal{K} \mid\!\approx specialpenguins \mathrel{|\!\sim} fly$ holds.
   - Since two ranks for formulas were discarded, the algorithm returns $i = 2$.

3.4 Classical Justification Algorithm

Currently, there is a range of existing tools that compute classical entailment explanations for PL. These tools vary in their algorithm, implementation, and approach to the entailment explanation. In this dissertation, we take the approach proposed by Horridge[21]. Horridge's proposal was presented in DL; we translate Horridge's algorithms to PL.

Consider the classical version of the example knowledge base in section 3.3.2, $\overrightarrow{\mathcal{K}} = \{penguins \rightarrow birds, robins \rightarrow birds, specialpenguins \rightarrow penguins, birds \rightarrow fly, birds \rightarrow wings, penguins \rightarrow \neg fly, specialpenguins \rightarrow fly\}$. Intuitively, each of the following queries should have the corresponding justifications:

1. $\overrightarrow{\mathcal{K}} \models robins \rightarrow wings$ is justified by $\{robins \rightarrow birds, brids \rightarrow wings\}$.

2. $\overrightarrow{\mathcal{K}} \not\models penguins \rightarrow wings$. Therefore justification $\mathcal{J} = \varnothing$.

3. $\overrightarrow{\mathcal{K}} \models specialpenguins \rightarrow fly$ has two set of justifications: $\{specialpenguins \rightarrow fly\}$ and $\{specialpenguins \rightarrow penguins, penguins \rightarrow birds, birds \rightarrow fly\}$.

Figure 3.2 shows the composition of the classical entailment algorithm proposed by Horridge. In computing a single justification for an entailment, Horridge uses the technique of expanding a subset of axioms until the set entails the query. To ensure the resulting subset is the minimal set for entailment, Horridge proposed to retract axioms from the subset without breaking the entailment. Lastly, Horridge uses the hitting set tree (HS-Tree) technique and computing a single justification to identify all possible justifications for the entailment.

In the following subsections, we build up the classical entailment explanation algorithm for PL by looking at each of the sub-algorithms. Firstly, we define the PL equivalent of the ExpandAxioms and ContractAxioms algorithms. These sub-algorithms are then used in computing a single justification in PL. Lastly, we can construct an algorithm using the HS-Tree technique that identifies all justifications for the entailment. The algorithms are almost identical compared to DL algorithms defined by Horridge. This is because the techniques for finding all justifications are logic independent.

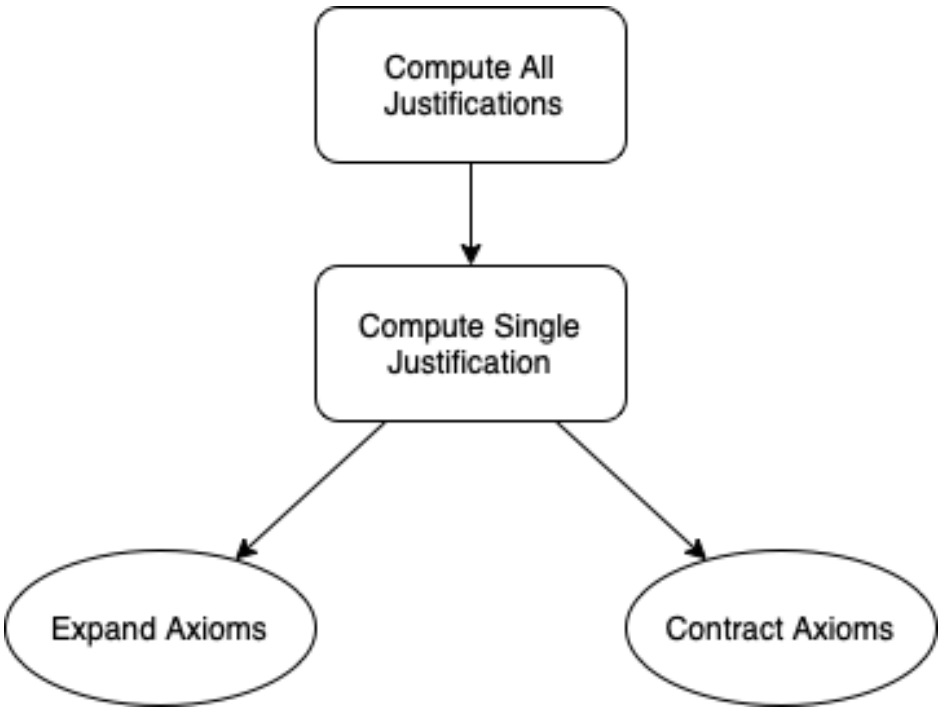

Figure 3.2: Algorithm Topology of Classical Entailment Explanation

3.4.1 Expand Formulas

Horridge defines *signature* in DL [21]. In this dissertation, we define *signature* in PL as follows:

Definition 3.1 (Signature)**.** *A formula's signature is the set of propositional atoms occurring in the formula.*

Horridge's strategy in identifying a subset of the knowledge base that entails the query is to start with an initial set, $S \subseteq \mathcal{K}$, and add formulas to the set until it entails the query. A challenge with this approach is to identify the most sensible initial set S.

A reasonable approach to identifying the initial set S is to put all the formulas related to the query α in S. A good approximation is the collection of formulas where their signature intersects with the query's signature.

For example, consider the knowledge base $\mathcal{K} := \{\alpha, \alpha \wedge \beta, \gamma, \delta \vee \neg\gamma, \neg\beta\}$, and the query $\alpha \rightarrow \gamma$. We define the set Σ as the query's signature. Therefore, we

have $\Sigma := \{\alpha, \gamma\}$. The initial set S is a subset of $\mathcal{K}$ where each formula in S has a signature that intersects with Σ. In this example, $S := \{\alpha, \alpha \wedge \beta, \gamma, \delta \vee \neg\gamma\}$.

In this dissertation, we define *FindRelatedFormulas(*$\Sigma, \mathcal{K}$) as the function that identifies the formulas in $\mathcal{K}$ with a signature that intersects with Σ.

Algorithm 6 ExpandFormulas

```
1: Input: Knowledge base 𝒦 and query α → β
2: Output: Set 𝒮
3: if 𝒦 ⊭ α → β then
4:     return ∅
5: else
6:     S := ∅
7:     S′ := ∅
8:     Σ := signature(α → β)
9:     while S′ ≠ S do
10:        S′ = S
11:        S = S ∪ FindRelatedFormulas(Σ, 𝒦)
12:        if S ⊨ α → β then
13:            return S
14:        end if
15:        Σ = signature(S)
16:    end while
17: end if
18: return S
```

ExpandFormulas (algorithm 6) is the PL equivalent of **ExpandAxioms**. Line 3 of **ExpandFormulas** checks if the input knowledge base $\mathcal{K}$ entails the query $\alpha \rightarrow \beta$. If the entailment does not hold, there is no explanation, and the result is an empty set. For cases where $\mathcal{K} \models \alpha \rightarrow \beta$, the algorithm initialised the sets S and S' to empty sets and Σ to the signature of $\alpha \rightarrow \beta$.

In essence, the algorithm sensibly expands the set S until it entails the query. The algorithm terminates when the set S stops expanding. There are two primary operations we need to account for:

1. Keep track of the growth of the set S: The set S' is set S in the previous iteration of the while loop. Therefore, S did not expand if $S' == S$ and the algorithm exits the while loop.

2. Appropriately expand S: In each iteration, the set S expands based on the signature of the query $\alpha \to \beta$ and its related formulas.

Notice that the **ExpandFormulas** algorithm is guaranteed to terminate because $\mathcal{K}$ is defined as finite. Therefore, set S cannot expand infinitely.

Expand Formulas Example

For example, consider the knowledge base $\mathcal{K} \coloneqq \{\alpha \to \beta, \beta \to \gamma, \gamma \to \delta, \delta \to \sigma, \omega \vee \neg\zeta, \zeta \wedge \gamma\}$ and the query $\alpha \to \sigma$.

1. The algorithm initialises Σ to the signature of the query, which is the set $\{\alpha, \sigma\}$. In the first iteration of the while loop, line 11 updates S to $\{\alpha \to \beta, \delta \to \sigma\}$. It is clear that $S \not\models \alpha \to \sigma$.

2. In the second iteration, S expands further to $\{\alpha \to \beta, \beta \to \gamma, \delta \to \sigma\}$ and it still does not yet entail the query. A further iteration expands the set S to $\{\alpha \to \beta, \beta \to \gamma, \gamma \to \delta, \delta \to \sigma, \zeta \wedge \gamma\}$ and now $S \models \alpha \to \sigma$. Finally, the algorithm returns S and terminates.

Notice that S in the example is not the minimal set that justifies the entailment. The set $\{\alpha \to \beta, \beta \to \gamma, \gamma \to \delta, \delta \to \sigma\}$ is a subset of S, and it also entails the query. In the next subsection, we define an algorithm that contracts the set S without breaking the entailment.

3.4.2 Contract Formulas

Given a knowledge base $\mathcal{K}$ and a subset $S \subseteq \mathcal{K}$ such that $S \models \alpha \to \beta$, the algorithm **ContractFormulas** returns a subset $S' \subseteq S$ such that $S' \models \alpha \to \beta$.

There are a few approaches to reducing the set S; Horridge mentions the simple approaches of removing the formulas one by one and the sliding window technique [21]. However, it is evident that divide and conquer is the more efficient technique for an extensive knowledge base. Algorithm 7 is the recursive implementation of the divide and conquer approach to contract a set of formulas S that entails the query $\alpha \to \beta$.

Algorithm 7 constructs a binary tree where each node splits the formulas into two subsets, S_L and S_R. In essence, the **ContractFormulas** algorithm discards branches that contain formulas which do not contribute to the entailment of the query.

Algorithm 7 ContractFormulas

1: Input: A set S and entailment α, such that $S \models \alpha$
2: Output: Set $S' \subseteq S$ such that $S' \models \alpha$
3: return *ContractFormulasRecursive*$(\varnothing, \mathcal{K}, \alpha)$
4: *ContractFormulasRecusive*$(S_{support}, S_{whole}, \alpha)$
5: **if** $|S_{whole}| == 1$ **then**
6: return S_{whole}
7: **end if**
8: $S_L, S_R :=$ *Splite*(S_{whole})
9: **if** $S_{support} \cup S_L \models \alpha$ **then**
10: return *ContractFormulasRecursive*$(S_{support}, S_L, \alpha)$
11: **end if**
12: **if** $S_{support} \cup S_R \models \alpha$ **then**
13: return *ContractFormulasRecursive*$(S_{support}, S_R, \alpha)$
14: **end if**
15: $S'_L :=$ *ContractFormulasRecursive*$(S_{support} \cup S_R, S_L, \alpha)$
16: $S'_R :=$ *ContractFormulasRecursive*$(S_{support} \cup S'_L, S_R, \alpha)$
17: return $S'_L \cup S'_R$

Notice that the set $S \subseteq \mathcal{K}$ is a finite set of formulas and cannot be divided and conquered infinitely. Therefore, **ContractFormulas** is guaranteed to terminate.

Contract Formulas Example

Consider the knowledge base $\mathcal{K} := \{\alpha \to \beta, \zeta \wedge \gamma, \gamma \to \delta, \delta \to \sigma, \omega \vee \neg\zeta, \beta \to \gamma\}$ and the query $\alpha \to \gamma$.

1. In the first recursion, we have $S_{support} = \varnothing$ and $S_{whole} = \mathcal{K}$. By splitting S_{whole} we get $S_L = \{\alpha \to \beta, \zeta \wedge \gamma, \gamma \to \delta\}$ and $S_R = \{\delta \to \sigma, \omega \vee \neg\zeta, \beta \to \gamma\}$. It is clear that neither $S_{support} \cup S_L \models \alpha \to \gamma$ nor $S_{support} \cup S_R \models \alpha \to \gamma$. Therefore, we execute the next recursion with the updated parameters to calculate S'_L as shown in line 15.

2. The next recursion initiates with the following parameters: $S_{support} = \{\delta \to \sigma, \omega \vee \neg\zeta, \beta \to \gamma\}$, $S_{whole} = \{\alpha \to \beta, \zeta \wedge \gamma, \gamma \to \delta\}$ and the query $\alpha \to \gamma$. Splitting S_{whole}, we get $S_L = \{\alpha \to \beta, \zeta \wedge \gamma\}$ and $S_R = \{\gamma \to \delta\}$. It is evident that $S_{support} \cup S_L \models \alpha \to \gamma$. Therefore the algorithm returns the

value of the next recursion using the following parameters $S_{support}, S_L$ and $\alpha \to \gamma$ respectively.

3. The following recursion initiates with the parameters: $S_{support} = \{\delta \to \sigma, \omega \vee \neg\zeta, \beta \to \gamma\}$, $S_{whole} = \{\alpha \to \beta, \zeta \wedge \gamma\}$ with the query $\alpha \to \gamma$. From S_{whole} we get $S_L = \{\alpha \to \beta\}$ and $S_R = \{\zeta \wedge \gamma\}$. Similar to the last recursion, we have $S_{support} \cup S_L \vDash \alpha \to \gamma$ and it follows that this recursion terminates with the return value *ContractFormulaRecursive*($S_{support}, S_L, \alpha \to \gamma$). It is easy to see that the return value is $\alpha \to \beta$ because $S_L = \{\alpha \to \beta\}$. This value propagates up to the first recursion and we have $S'_L = \alpha \to \beta$.

4. The calculated value of S'_L allows use to calculate S'_R as show in line 16 of the algorithm. As indicated in step 1 in the first recursion, the algorithm had the following values for each variable:

   - $S_{support} = \varnothing$
   - $S_{whole} = \{\alpha \to \beta, \zeta \wedge \gamma, \gamma \to \delta, \delta \to \sigma, \omega \vee \neg\zeta, \beta \to \gamma\}$
   - $\alpha \to \gamma$
   - $S_L = \{\alpha \to \beta, \zeta \wedge \gamma, \gamma \to \delta\}$
   - $S_R = \{\delta \to \sigma, \omega \vee \neg\zeta, \beta \to \gamma\}$
   - $S'_L = \{\alpha \to \beta\}$

5. According to line 16, S'_R is calculated with *ContractFormulasRecursive*($S_{support} \cup S'_L, S_R, \alpha \to \gamma$). By applying the variables mentioned above, we have $S'_R =$ *ContractFormulasRecursive*($\varnothing \cup \alpha \to \beta, \{\delta \to \sigma, \omega \vee \neg\zeta, \beta \to \gamma\}, \alpha \to \beta$).

6. In the first recursion of calculating S'_R, the algorithm splits S_{whole} into the following: $S_L = \{\delta \to \sigma, \omega \vee \neg\zeta\}$ and $S_R = \{\beta \to \gamma\}$ with $S_{support} = \{\alpha \to \beta\}$. It is clear that $S_{support} \cup S_R \vDash \alpha \to \gamma$ and the algorithm terminates with the final set $S' = \{\alpha \to \beta, \beta \to \gamma\}$

Figure 3.3 presents a visual illustration of the above example. The formulas in the red leaf nodes are ignored because these formulas do not contribute to the entailment of $\alpha \to \gamma$. The remaining formulas in the green leaf nodes form the smallest subset of formulas that entails $\alpha \to \gamma$.

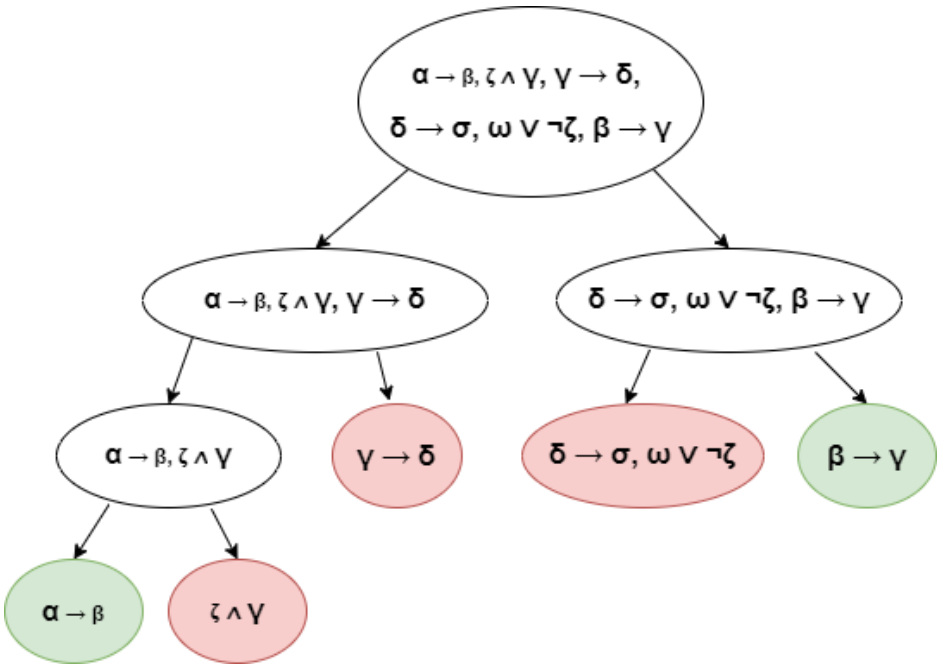

Figure 3.3: Visual illustration of classical entailment example

3.4.3 Compute Single Justification

An algorithm that computes a single justification for an entailment in PL is constructed using **ExpandFormulas** and **ContractFormulas** sub-algorithms. The algorithm first finds a subset $S \subseteq \mathcal{K}$ that entails the query using **ExpandFormulas**. It then contracts the set S using **ContractFormulas** to find the smallest subset of S that entails the query.

Algorithm 8 finds a single justification for an entailment $\mathcal{K} \models \alpha$. Figure 3.4 is a visual illustration of the intuition behind the algorithm of finding a single justification for an entailment.

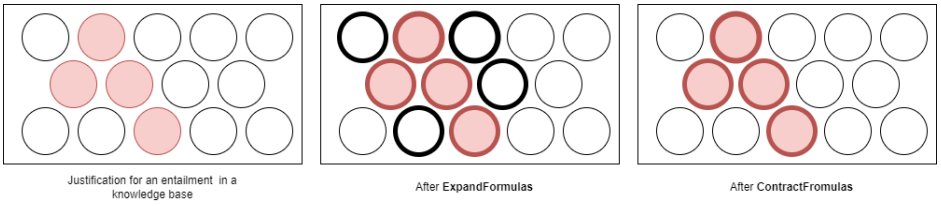

Figure 3.4: Visual illustration of finding a single justification

Algorithm 8 ComputeSingleJustification

```
1: Input: Knowledge base 𝒦 and entailment α
2: Output: Justification 𝒥
3: if α ∈ 𝒦 then
4:     return α
5: end if
6: S := ExpandFormulas(𝒦, α)
7: if S == ∅ then
8:     return ∅
9: end if
10: 𝒥 := ContractFormulas(S, α)
11: return 𝒥
```

3.4.4 Compute All Justifications

Horridge proposes an algorithm that uses the HS-Tree technique to find all classical justifications of a DL entailment[21]. The HS-Tree technique was originally applicable in the *Theory of Diagnosis* in which Reiter presented the technique to identify the minimal hitting set given a set of conflicting sets in a universe [33]. The HS-Tree technique can also be used to dynamically identify the set of conflicting sets.

A knowledge base and a set of justifications map to the notion of the universe and a set of conflicting sets in the field of diagnosis, respectively. Kalyanpur et al. proposed using the HS-Tree algorithm to find all justifications for an entailment [23].

Many works of literature have demonstrated a problem with Reiter's HS-Tree algorithm [20, 39]. The third pruning rule of the HS-Tree algorithm may disregard some valid branches of the HS-Tree. The rule states the following:

- If nodes n and n' have been labelled by sets S and S' of C, respectively, and if S' is a proper subset of S, then for each $\alpha \in S \setminus S'$ mark as redundant the edge from node n labelled by α. A redundant edge, together with the sub-tree beneath it, may be removed from the HS-Tree while preserving the property that the resulting pruned HS-Tree will yield all minimal hitting sets for C.

In both figure 3.5 and figure 3.6, the branch labelled "a", along with its sub-tree is disregarded by the pruning rule.

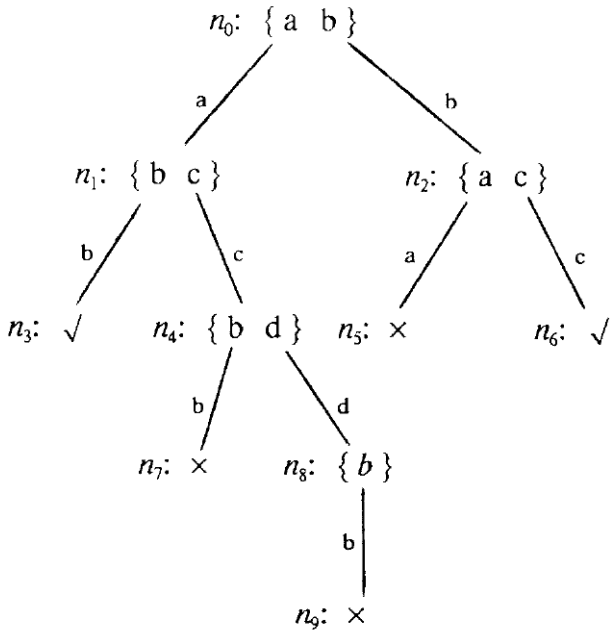

Figure 3.5: HS-Tree demonstrating the problem with the pruning rule

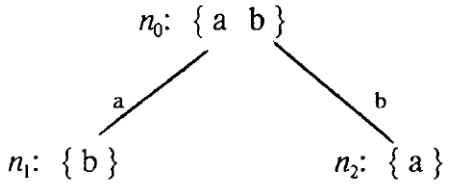

Figure 3.6: HS-Tree demonstrating the problem with the pruning rule

As a result, valid justifications (labels on the nodes) may be disregarded. Notice that this problem occurs when a child node has a label that is a proper subset of the label in one of its ancestor nodes. By the definition of justification, this scenario would never happen. For example, given an ancestor node labelled with justification J and a child node labelled with justification J' where $J' \subset J$, both J and J' cannot be justifications for the same entailment.

Hence, the problems identified in Reiter's HS-Tree algorithm are not applicable when identifying all justifications for an entailment.

This dissertation presents a similar approach to compute all justifications for classical PL entailment. We propose the same algorithm as Horridge's algorithm, as it is logic independent.

To demonstrate algorithm 9, we present the following example: Consider the knowledge base $\mathcal{K} = \{\alpha \rightarrow \beta, \beta \rightarrow \gamma, \alpha \rightarrow \gamma, \beta \rightarrow \delta, \delta \rightarrow \gamma, \alpha \rightarrow \delta\}$ with the query $\alpha \rightarrow \gamma$.

1. Line 3 - 5 initialises the following variables: $S_{working} = \mathcal{K}$, $X_{explored} = \varnothing$ and $X_{result} = \varnothing$. Line 6 computes a single justification for the entailment by calling the function *ComputeSingleJustifiaction*$(\mathcal{K}, \alpha \rightarrow \gamma)$. The result is simply $\alpha \rightarrow \gamma$ because $\alpha \rightarrow \gamma \in \mathcal{K}$. Therefore, we have $J_{root} = \alpha \rightarrow \gamma$ and $X_{result} = \{\alpha \rightarrow \gamma\}$. Furthermore, line 9 adds v_{root} to the queue Q. Currently, the tree only has the root node labelled with $\alpha \rightarrow \gamma$.

2. Line 11 - 33 executes as long as there is some node in the queue Q. In the first iteration we have the $v_{head} = v_{root}$ and $j_{head} = \{\alpha \rightarrow \gamma\}$. Since there is only one element in j_{head}, the for loop on lines 14 - 32 only executes once. From the next few lines of the algorithm, we have the following variables:

   - $S_{path} = \varnothing \cup \{\alpha \rightarrow \gamma\}$
   - $X_{explored} = \varnothing \cup \{\alpha \rightarrow \gamma\}$

Algorithm 9 ComputeAllJustification

Input: Knowledge Base $\mathcal{K}$ and entailment α
Output: Set of Justifications $\mathcal{J}$
$S_{working} := \mathcal{K}$
$X_{explored} := \varnothing$
$X_{result} := \varnothing$
$\mathcal{J}_{root} :=$ *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 $Q \neq \varnothing$ **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**
            $X_{explored} = X_{explored} \cup \{S_{path}\}$
            $J' =$*GetNonIntersectingJustification*(S_{path}, X_{result})
            if $J' == \varnothing$ **then**
                $S_{working} = S_{working} \setminus \{S_{path}\}$
                $J' =$ *ComputeSingJustification*$(S_{working}, \alpha)$
                $S_{working} = S_{working} \cup \{S_{path}\}$
            end if
            $v_{fresh} =$*GetFreshNode*(J')
            $e =$ *GetFreshEdge*$(\langle v_{fresh}, v_{head} \rangle, \beta)$
            $T_{hst} = T_{hst} \cup \{e\}$
            if $J' \neq \varnothing$ **then**
                $X_{result} = X_{result} \cup \{J'\}$
                Enqueue(v_{fresh}, Q)
            end if
        end if
    end for
end while
return X_{result}

Figure 3.7: HS-Tree at step 2.

- J' =*GetNonIntersectingJustifications*($S_{path}, X_{explored}$).
  The *GetNonIntersectingJustifications*($S_{path}, X_{explored}$) function removes all the formulas in S_{path} and $X_{explored}$ from $\mathcal{K}$ and computes a justification in the remaining formulas for the entailment. In this case, the remaining formulas are $\{\alpha \rightarrow \beta, \beta \rightarrow \gamma, \beta \rightarrow \delta, \delta \rightarrow \gamma, \alpha \rightarrow \delta\}$. It is easy to see the subset $\{\alpha \rightarrow \beta, \beta \rightarrow \gamma\}$ is a justification for $\alpha \rightarrow \gamma$. Therefore, $J' = \{\alpha \rightarrow \beta, \beta \rightarrow \gamma\}$.

Lines 24 - 26 construct an HS-Tree as shown in figure 3.7.

Lastly, the algorithm adds J' to $X_{results}$ and adds the new node to the queue Q.

3. The next iteration of the while loop is based on the last node added to Q where the label of the node $j_{head} = \{\alpha \rightarrow \beta, \beta \rightarrow \gamma\}$. For the case where the algorithm looks at $\alpha \rightarrow \beta$, we have the following variables:

   - $S_{path} = \{\alpha \rightarrow \beta, \alpha \rightarrow \gamma\}$
   - $X_{explored} = \{\{\alpha \rightarrow \gamma\}, \{\alpha \rightarrow \beta, \alpha \rightarrow \gamma\}\}$
   - We can see that $\mathcal{K}$ with S_{path} and $X_{explored}$ removed is the set $\{\beta \rightarrow \gamma, \beta \rightarrow \delta, \delta \rightarrow \gamma, \alpha \rightarrow \delta\}$. The subset $\{\alpha \rightarrow \delta, \delta \rightarrow \sigma\}$ is a non-intersecting justification of the entailment $\alpha \rightarrow \sigma$. Therefore, $J' = \{\alpha \rightarrow \delta, \delta \rightarrow \gamma\}$.

   In this iteration, lines 24 - 26 add a new node to the tree as shown in figure 3.8.

   Lastly, the algorithm adds J' to X_{result} and adds the new node to Q.

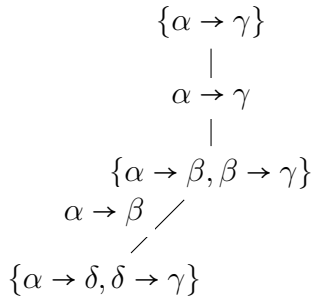

Figure 3.8: HS-Tree at step 3.

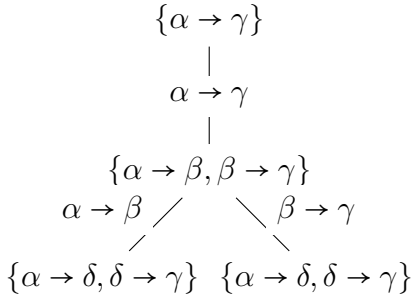

Figure 3.9: HT-Tree at step 4.

4. For the case where the algorithm looks at $\beta \to \gamma$ for the node with the label $\{\alpha \to \beta, \beta \to \gamma\}$, we have the following variables:

   - S_{path} = $\{\alpha \to \gamma, \beta \to \gamma\}$
   - $X_{explored}$ = $\{\{\alpha \to \gamma\}, \{\alpha \to \beta, \alpha \to \gamma\}, \{\alpha \to \gamma, \beta \to \gamma\}\}$
   - Similar to the previous iteration, we have J' = $\{\alpha \to \delta, \delta \to \gamma\}$.

   Figure 3.9 show the HS-Tree as a result of this iteration.

   Note that although the new node has the same label as the previous node, the two nodes are still considered distinct nodes. The algorithm then adds the node to Q and adds J' to X_{result}

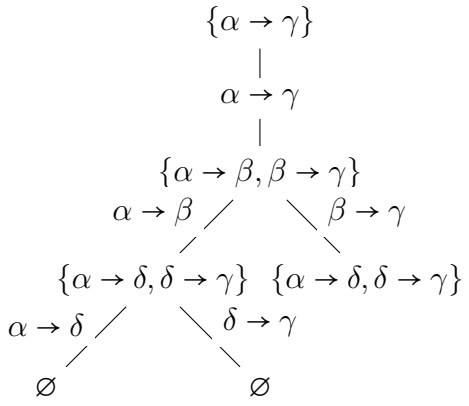

Figure 3.10: HS-Tree at step5.

5. The algorithm then looks at the left node with the label $\{\alpha \to \delta, \delta \to \gamma\}$. In the case where we consider $\alpha \to \delta$, the variables update as follows:

   - $S_{path} = \{\alpha \to \gamma, \alpha \to \gamma, \alpha \to \delta\}$
   - $X_{explored} = \{\{\alpha \to \gamma\}, \{\alpha \to \beta, \alpha \to \gamma\}, \{\alpha \to \gamma, \beta \to \gamma\}, \{\alpha \to \gamma, \alpha \to \gamma, \alpha \to \delta\}\}$
   - There is no more non-intersecting justification for the entailment. Line 20 updates $S_{working}$ to $S_{working} \setminus \{S_{path}\}$ and this evaluates to the set $\{\beta \to \gamma, \beta \to \delta, \delta \to \gamma\}$. No subset of $S_{working}$ entails $\alpha \to \gamma$. Therefore, $J' = \varnothing$.

   With the same procedures, one can also conclude that $J' = \varnothing$ when the algorithm takes the $\delta \to \gamma$ path from the node.

   The resulting HS-Tree are shown in figure 3.10.

6. Now the algorithm considers the right node with the label $\{\alpha \to \delta, \delta \to \gamma\}$. Once again, when the algorithm considers $\delta \to \gamma$, no justifications can be found. However, when $\alpha \to \delta$ is considered, the following variables are updated:

   - $S_{path} = \{\alpha \to \gamma, \beta \to \gamma, \alpha \to \delta\}$
   - $X_{explored} = \{\{\alpha \to \gamma\}, \{\alpha \to \beta, \alpha \to \gamma\}, \{\alpha \to \gamma, \beta \to \gamma\}, \{\alpha \to \gamma, \alpha \to \gamma, \alpha \to \delta\}, \{\alpha \to \gamma, \alpha \to \gamma, \delta \to \gamma\}, \{\alpha \to \gamma, \beta \to \gamma, \delta \to \gamma\}, \{\alpha \to \gamma, \beta \to \gamma, \alpha \to \delta\}\}$

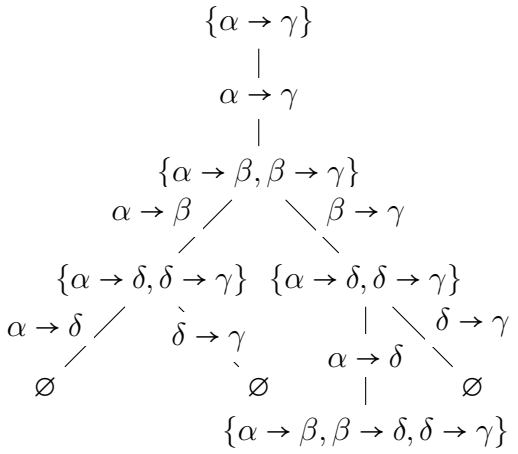

Figure 3.11: HS-Tree at step 6.

- It is evident that there are no more non-intersecting justifications. Line 20 of the algorithm updates $S_{working}$ to $\{\alpha \to \beta, \beta \to \delta, \delta \to \gamma\}$. Since, $S_{working} \vDash \alpha \to \gamma$, we have $J' = \{\alpha \to \beta, \beta \to \delta, \delta \to \gamma\}$.

The resulting HS-Tree from this iteration is shown in figure 3.11.

The algorithm adds the new node to Q and adds $\{\alpha \to \beta, \beta \to \delta, \delta \to \gamma\}$ to X_{result}.

7. Looking at the last node in Q and following the same procedure, one can easily conclude that there is no other justification for the entailment. The algorithm terminates with the HS-Tree presented in figure 3.12.

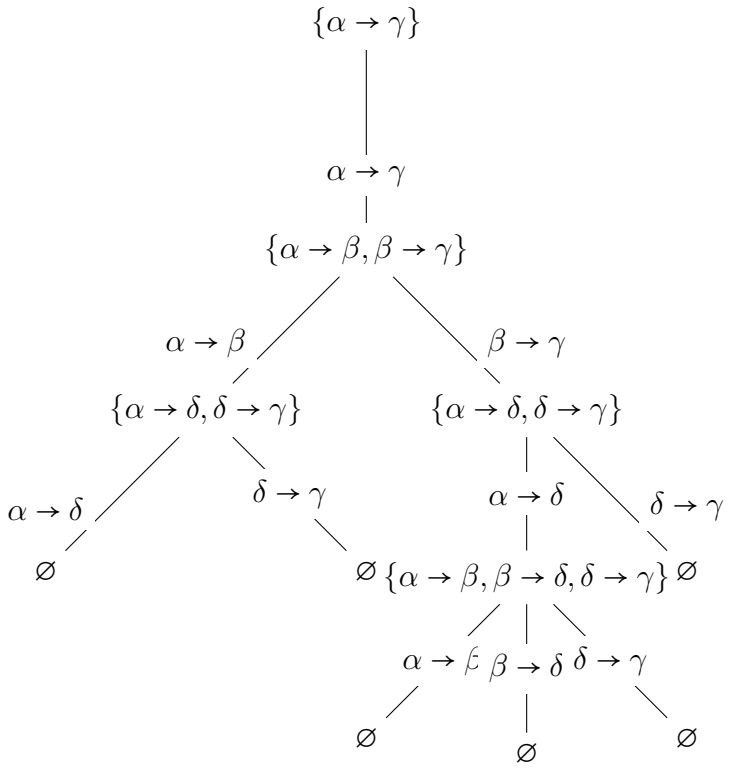

Figure 3.12: Final HS-Tree

3.5 Dematerialisation Algorithm

In the previous subsection, we have shown that the defeasible entailment justification algorithm uses the classical entailment justification algorithm. The classical entailment justification algorithm takes the materialised knowledge base $\mathcal{K}$ as input. Consequently, formulas in the resulting justifications are subsets of the materialised knowledge base. In other words, for any justification j in the result of the classical justification algorithm, $j \subseteq (\overrightarrow{R_0} \cup \overrightarrow{R_1} \cup ... \cup \overrightarrow{R_\infty})$. Results from the classical entailment justification algorithm cannot be used as the results for defeasible justification unless the necessary formulas are restored from their material counterparts. This motivates the need for the **Dematerialisation** algorithm.

Algorithm 10 Dematerialisation

```
1: Input: Set of justifications 𝒥 and defeasible knowledge base 𝒦
2: Output: Set of dematerialised justifications 𝒥
3: for j in 𝒥 do
4:     for α → β in j do
5:         if α → β ∉ 𝒦 then
6:             Replace α → β with α |∼ β
7:         end if
8:     end for
9: end for
10: Return 𝒥
```

The algorithm iterates through each justification j in $\mathcal{J}$. For each justification j, the algorithm looks at each formula. If the formula, $\alpha \rightarrow \beta$ is not in the original knowledge base, then it is updated to the defeasible implication, $\alpha \mathrel{|\sim} \beta$.

3.6 Justification Algorithm for Defeasible Entailment using the KLM Framework

In this section, we present algorithm 11, which computes defeasible justifications for a defeasible entailment using the KLM Framework. The algorithm takes a knowledge base $\mathcal{K}$ and a defeasible implication $\alpha \mathrel{|\sim} \beta$ as inputs. If $\mathcal{K} \mid\!\approx \alpha \mathrel{|\sim} \beta$, algorithm 11 produces a set of justification(s) for the entailment.

Otherwise, the algorithm produces a subset $S \subseteq \mathcal{K}$ such that $S \not\models \alpha \mid\sim \beta$ for non-entailment.

Algorithm 11 KLMDefeasibleJustification

```
Input: Defeasible knowledge base 𝒦 and query α |~ β
Output: Justification 𝒥
i := 0
𝒥 := ∅
entailment, (ℛ_0, ℛ_1, ..., ℛ_∞), rank :=
    RationalClosureForJustification(𝒦, α |~ β)
if !entailment then
    while i < rank do
        𝒦 = 𝒦 \ℛ_i
        i = i + 1
    end while
    return 𝒦 |≈ α |~ β
end if
if rank == 0 then
    𝒥 = ComputeAllJustifications(→𝒦, α → β)
    𝒥 = dematerialisation(𝒥, 𝒦)
    return 𝒥
end if
while i < rank do
    𝒦 = 𝒦\𝒦_i
    i = i + 1
end while
𝒥 = computeAllJustifications(→𝒦, α → β)
𝒥 = dematerialisation(𝒥, 𝒦)
return 𝒥
```

Consider the example we presented in sections 3.3 and 3.4. We have $\mathcal{K} = \{penguins \rightarrow birds, robins \rightarrow birds, specialpenguins \rightarrow penguins, birds \mid\sim fly, birds \mid\sim wings, penguins \mid\sim \neg fly, specialpenguins \mid\sim fly\}$ which is base ranked as shown in table 3.1. For each of the queries. $robins \mid\sim wings$, $penguins \mid\sim wings$ and $specialpenguins \mid\sim fly$, we have shown the output of the **RationalClosureForJustification** algorithm in section 3.3.2. We present the resulting defeasible justification(s) for each query by walking through the **KLMDefeasibleJustification** algorithm.

1. For query $robins \mid\sim wings$, $\mathcal{K} \mid\approx robins \mid\sim wings$ hold, and no formulas were discarded in the computations of **RationalClosureForJustification** algorithm. Therefore, line 15 of the **KLMDefeasibleJustification** algorithm calls **ComputeAllJustifications** with the parameters $\overrightarrow{\mathcal{K}}$ and $robins \rightarrow wings$. It is easy to see that $\mathcal{J} = \{\{robins \rightarrow birds, birds \rightarrow wings\}\}$. Since $birds \rightarrow wings \notin \mathcal{K}$, the **dematerialise** algorithm converts $\mathcal{J}$ to defeasible justification by replacing $birds \rightarrow wings$ with $birds \mid\sim wings$. Therefore the defeasible justification for the defeasible entailment $\mathcal{K} \mid\approx robins \mid\sim wings$ is $\{robins \rightarrow birds, birds \mid\sim wings\}$.

2. For query $penguins \mid\sim wings$, $\mathcal{K} \not\mid\approx penguins \mid\sim wings$. In the process of checking defeasible entailment, formulas in $\mathcal{R}_0$ were discarded. Therefore, the **KLMDefeasibleJustification** algorithm produces the following justification for the non-entailment: $\{penguins \rightarrow birds, robins \rightarrow birds, specialpenguins \rightarrow penguins, specialpenguins \mid\sim fly, penguins \mid\sim \neg fly, birds \mid\sim fly, birds \mid\sim wings\} \not\mid\approx penguins \mid\sim wings$

3. For query $specialpenguins \mid\sim fly$, $\mathcal{K} \mid\approx specialpenguins \mid\sim fly$ and formulas in both $\mathcal{R}_0$ and $\mathcal{R}_1$ were discarded in rational closure computations. The while loop on lines 18 -21 of **KLMDefeasibleJustification** removes the discarded formulas from $\mathcal{K}$. Now, $\mathcal{K} = \{penguins \rightarrow birds, robins \rightarrow birds, specialpenguins \rightarrow penguins, specialpenguins \mid\sim fly\}$. Then $\overrightarrow{\mathcal{K}}$ and $specialpenguins \rightarrow fly$ are passed into the **ComputeAllJustificiations** algorithm, which results in $\mathcal{J} = \{specialpenguins \rightarrow fly\}$. Lastly, the **dematerialisation** algorithm converts $\mathcal{J}$ to $\{specialpenguins \mid\sim fly\}$.

Example 3 demonstrates the difference between classical justification and defeasible justification. For the classical entailment $\overrightarrow{\mathcal{K}} \models specialpenguins \rightarrow fly$, there would have been two justifications for the entailment: $\{specialpenguins \rightarrow fly\}$ and $\{specialpengins \rightarrow penguins, penguins \rightarrow birds, birds \rightarrow fly\}$. However, in defeasible reasoning, $birds \mid\sim fly$ is discarded in the rational closure computations. As a result, there is only one defeasible justification.

Chapter 4

Implementation of Defeasible Justification Algorithm

4.1 Overview

In this chapter, we present the practical aspect of this dissertation. In addition to the theoretical formulation of a defeasible justification algorithm in Chapter 3, we implemented the algorithm as a software tool. We refer to the software tool as the KLMDefeasibleJustificationTool. The code for the tool is publicly available on GitHub[1].

This chapter is organised as follows: Section 4.3 presents the software architecture of the KLMDefeasibleJustificationTool. The software tool uses various external packages discussed in section 4.4. Section 4.2 specifies the input and output parameters. Section 4.5 describes the implementation of the algorithms presented in Chapter 3. Lastly, the test and the evaluation conducted for the KLMDefeasibleJustificationTool are discussed in section 4.6.

[1]The source code is available at https://github.com/SteveWang7596/DefeasibleJustificationForPropositionalLogic. Instructions on compilation and running the tool are provided in the readme.rm file.

```
Penguin => Bird
Robin=>Bird
SpecialPenguin=>Penguin
Bird~>Fly
Bird~>Wings
Penguin~>!Fly
SpecialPenguin~>Fly
```

Figure 4.1: Example input to the KLMDefeasibleJustificationTool

4.2 Input and Output Parameters

The **KLMDefeasibleJustificationTool** takes in two input parameters: a defeasible knowledge base text file and a query string. Figure 4.1 shows an example input for the knowledge base $\mathcal{K} = \{Penguin \rightarrow Bird, Robin \rightarrow Bird, SpecialPenguin \rightarrow Penguin, Bird \mid\sim Fly, Bird \mid\sim Wings, Penguin \mid\sim \neg Fly, SpecialPenguin \mid\sim Fly\}$. The query of the defeasible entailment is entered into the **query** text field on the GUI. The string "SpecialPenguin~>Fly" is an example input for the query $SpecialPenguin \mid\sim Fly$.

Given the above inputs, figure 4.2 displays the tool's output. The output text area presents the ranks of formulas discarded from the knowledge base and the remaining formulas in the knowledge base after each removal. Lastly, the tool presents valid defeasible justifications for the defeasible entailment $\mathcal{K} \mid\approx SpecialPenguin \mid\sim Fly$.

4.3 Software Architecture

The **KLMDefeasibleJustificationTool** is implemented in the Java programming language [3] and follows the Model View Controller (MVC) software architecture pattern as shown in figure 4.3 [24]. The implementation leverages two established external packages, further discussed in section 4.4. The **KLMDefeasibleJustificationTool** is composed of objects, a graphical user interface (GUI), and algorithm implementations.

The **KLMDefeasibleJustificationTool** uses Java's object-oriented programming to construct formulas, an HS-Tree and objects that encapsulate multiple return values. We use an existing package known as the Tweety Project for classical formula models, and the package is further extended to

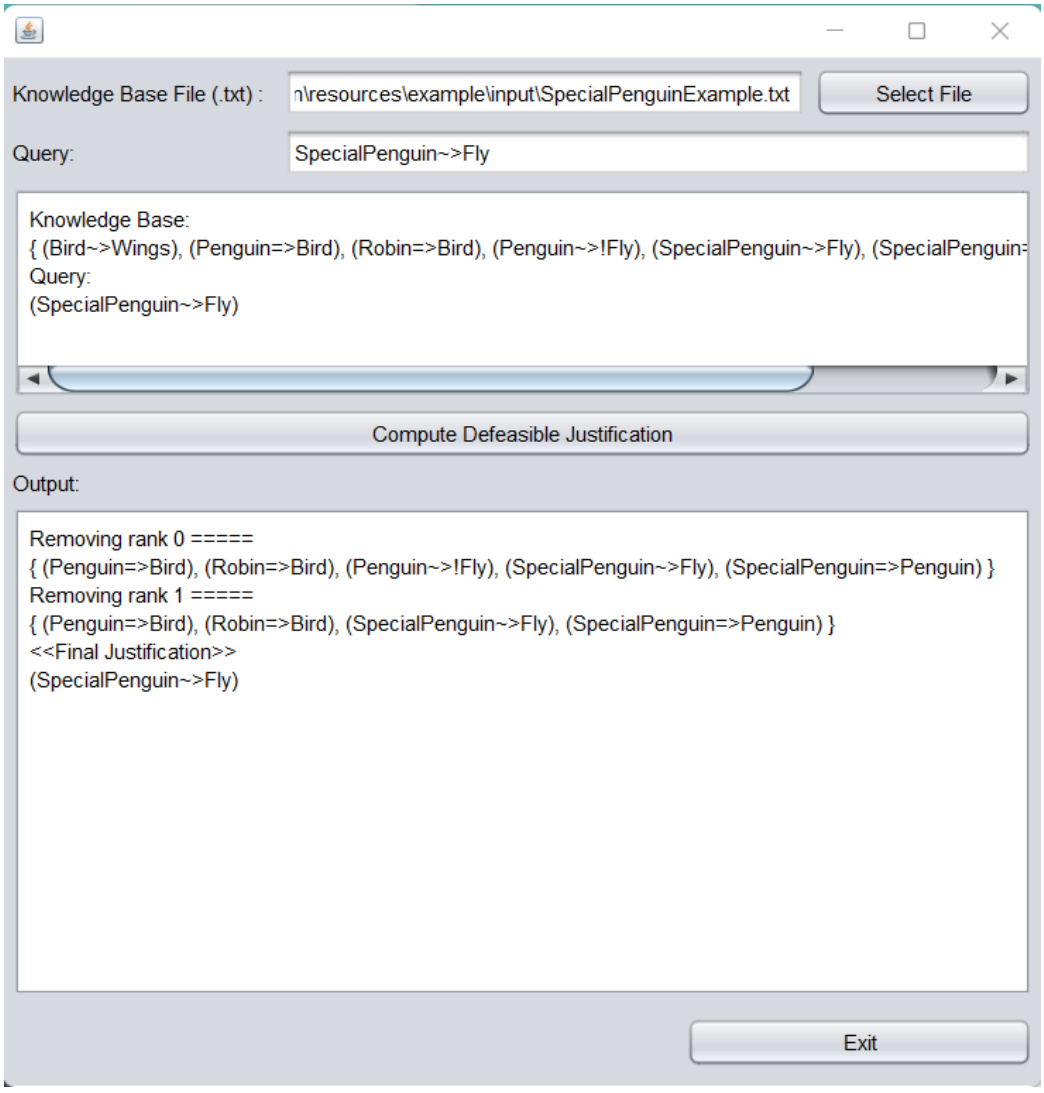

Figure 4.2: Example output to the KLMDefeasibleJustificationTool

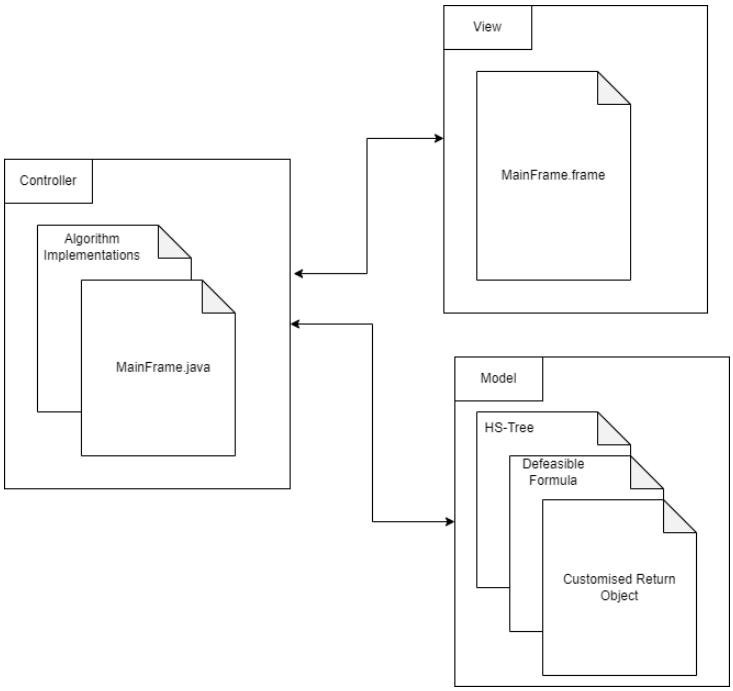

Figure 4.3: The architecture of the KLMDefeasibleJustificationTool based on the MVC pattern

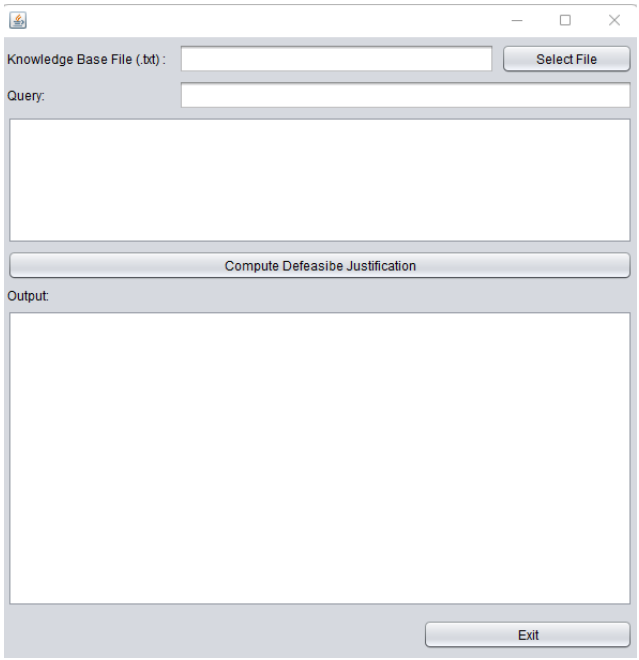

Figure 4.4: Graphical User Interface of the KLMDefeasibleJustificationTool

construct models for defeasible formulas. Horridge's algorithm to compute all justifications (algorithm 9) traverses an HS-Tree structure while keeping track of the traversed paths in memory. The implementation of algorithm 9 is simplified by using Java's object-oriented programming. A customised tree object, which corresponds to the HS-Tree, is used to find all justifications. Lastly, as shown in algorithm 5, the **RationalClosureForJustification** algorithm returns multiple values. Therefore, we use a customised return object encapsulating all the return values.

The **KLMDefeasibleJustificationTool** provides a GUI which allows users to interact with the tool. We use the Java Swing package to construct the GUI because it is simple and adequate for this tool. Figure 4.4 shows the GUI of the **KLMDefeasibleJustificationTool**.

The controller of the **KLMDefeasibleJustificationTool** orchestrates the program. It defines the actions performed based on user interactions. The following list of actions is performed based on user interactions.

- When the "Select File" button is clicked, the user is presented with the file chooser window.

- The user needs to select a text file that contains the knowledge base.
- The knowledge base in the text file is presented in the text area.
- The user then enters text that represents the query.
- When the "Compute Defeasible Justification" button is clicked, the program computes the defeasible justifications based on the user inputs.
- The resulting defeasible justification(s) is displayed in the output text area.
- The program terminates when the user clicks the "Exit button".

4.4 External Packages

The **KLMdefeasibleJustificationTool** uses two external packages to compute the defeasible justification: the Tweety Project [2] and the SAT4J SAT solver [1].

The Tweety Project is a software framework that provides models and operations for most First Order Logic (FOL), including PL. Our implementation extends the Tweety Project's PL models to construct models and operations required by the KLM Framework. Due to the restrictions and constants defined by the Tweety Project, we cannot denote the KLM Framework operations with the conventional symbols used in the literature. Instead, the negation symbol. $\neg$ is replaced with ! and the binary operations $\wedge, \vee, \rightarrow, \leftrightarrow$ and $\mid\sim$ are replaced with &&, ||, =>, <=> and ~>, respectively. Furthermore, we constructed a parser that reads strings such as "α ~> β" and produces an instance of *DefeasibleImplication*, which allows the software to perform operations such as materialisation.

The classical justification algorithm mentioned in section 2.3 and algorithm 5 (*RationalClosureForJustification*) require a tool to compute classical entailment. We used the SAT4J SAT solver [1] to perform classical entailment computations.

4.5 Algorithm Implementation

The implementation of the defeasible justification, base rank, rational closure, and dematerialisation follows from algorithms 11, 4, 5 and 10, respectively. Horridge's algorithms to find a single classical justification follow exactly from algorithms 6, 7 and 8. The implementation of algorithm 9 to find all classical justifications is aided by a customised tree object corresponding to the HS-Tree structure. Each node of the tree keeps track of the current knowledge base at the node, the justification for the entailment at the node, and a list of child nodes. The edge which connects a parent node to one of its child nodes represents a formula in the parent node's knowledge base, which is removed from the child node's knowledge base. This approach takes advantage of Java's object-oriented programming style and allows a clear structuring of the information required to find all justifications.

4.6 Testing and Evaluation

The **KLMDefeasibleJustificationTool** is tested with the defeasible justification examples mentioned in Chapter 3. Consider the inputs with the knowledge base $\mathcal{K} = \{penguins \rightarrow birds, robins \rightarrow birds, specialpenguins \rightarrow penguins, birds \mid\sim fly, birds \mid\sim wings, penguins \mid\sim \neg fly, specialpenguins \mid\sim fly\}$ and the queries $robins \mid\sim wings$, $penguins \mid\sim wings$ and $specialpenguins \mid\sim fly$. The **KLMDefeasibleJustificationTool** produces figures 4.5, 4.6, and 4.2 for each of the queries, respectively. The tool concludes with the correct defeasible justifications for each test case. These results correspond with the theoretical examples in Chapter 3.

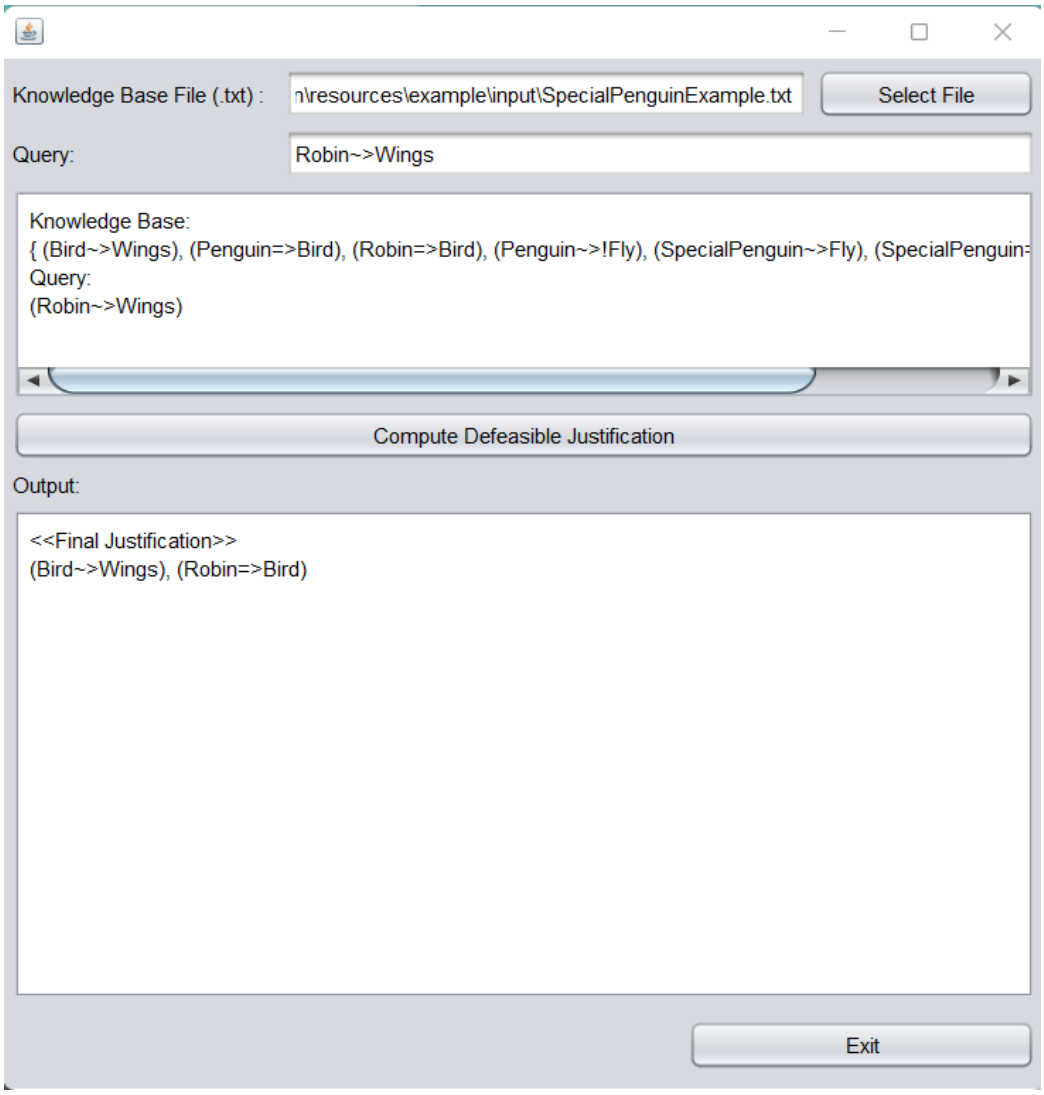

Figure 4.5: Robin has wings example

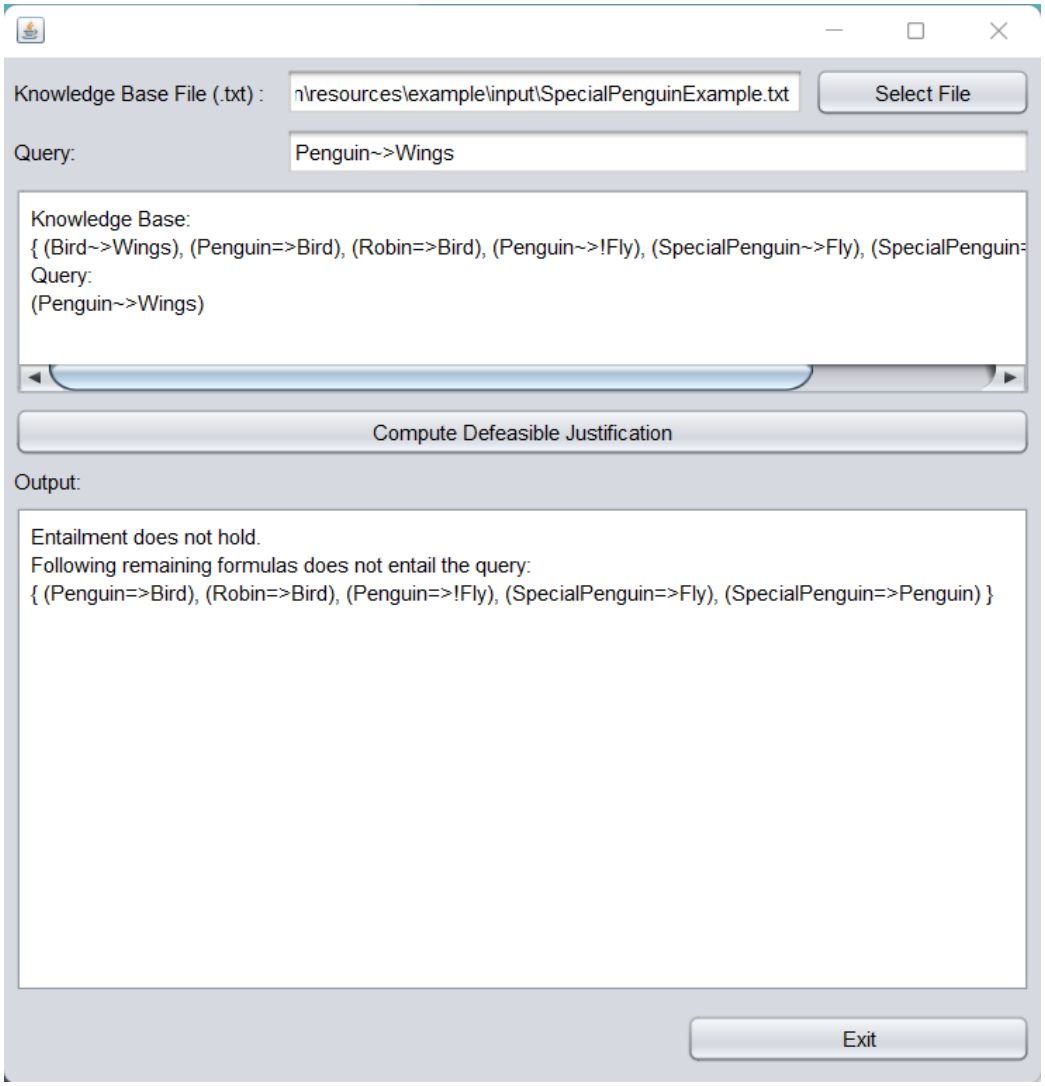

Figure 4.6: Penguin has wings example

Chapter 5

Conclusion and Future Work

This chapter concludes this dissertation by summarising the content and the contributions of this dissertation. Lastly, we propose some possible future work that can be branched off from this dissertation.

5.1 Summary and Contribution

This dissertation starts by introducing KR and reasoning. We then distinguish classical reasoning and defeasible reasoning. The conclusions drawn from defeasible reasoning are often challenging to understand. Defeasible justifications explain a defeasible entailment and aid users' understanding of defeasible reasoning. This dissertation contributes to such a need by presenting an algorithm that computes the justifications for a defeasible entailment. Furthermore, implementing the proposed algorithm forms the practical contribution of this dissertation.

To formalise a defeasible justification algorithm for the KLM Framework, we need relevant knowledge related to PL, classical justification, the KLM Framework, and rational closure. Before understanding justifications for defeasible reasoning, one requires prior knowledge about justification in the classical case. We then presented the KLM Framework which is an extension of PL equipped to perform defeasible reasoning. Lastly, we introduced a form of defeasible reasoning known as rational closure.

As the theoretical contribution of this dissertation, we presented an algorithm that computed defeasible justification using the KLM Framework. Chama proposed a defeasible justification algorithm for DL in her work [11].

We translated her algorithm to PL and made adjustments accordingly. Our justification algorithm comprises various sub-algorithms, including rational closure, classical justification, and dematerialisation algorithms. The rational closure sub-algorithm is a modified version of the original algorithm mentioned in Freund's paper [16]. Horridge proposed a classical justification algorithm which we use as a sub-algorithm [21]. Lastly, the dematerialisation sub-algorithm ensures the classical justifications are converted to defeasible justifications accordingly.

The practical contribution of this dissertation is a Java application which implements the proposed defeasible justification algorithm. The source code of the application is publicly available on GitHub. The software follows the MVC architecture and provides a GUI for user interaction. The implementation uses two external packages: The Tweety Project and the SAT4J SAT solver. The GUI allows users to input a knowledge base in the form of a text file and a query string. The text area displays the resulting defeasible justifications. The theoretical algorithms are the pseudo-code for the implementation except for ComputeAllJustifications where we used object-oriented programming to construct the HS-Tree. Finally, the defeasible justification tool was tested with three representative test cases.

5.2 Future Work

The possible future work based on this dissertation may extend in two aspects: theoretical and practical.

Casini et al. presented a restricted version of first-order logic equipped with KLM-style defeasibility [9]. The defeasible justification algorithm can be translated and applied to other logic systems with defeasibility. Applying the defeasible justification algorithm to Casini's restricted first-order logic requires detailed investigation.

The defeasible justification algorithms for both PL and DL are based on Chama's definition of defeasible justification [11]. However, the definition is very procedural, and further theoretical investigation is required to formulate the definition(s) of defeasible justification or defeasible explanation.

Currently, only experts in logic would understand the syntax and semantics used in this dissertation. Natural Language Processing (NLP) can generate human-understandable sentences from propositional formulas [36, 7]. The generated sentences allow less equipped users to use the defeasible jus-

tification tools.

The defeasible justification tool presented in Chapter 4 is a minimally viable product that meets the needs of this dissertation. The tool can be improved in terms of scalability and efficiency. An analysis of the software performance would indicate the limits of the software. Such investigations need to consider the size of the knowledge base, the complexity of identifying the justifications, and various other adjustable characteristics of the tool. Furthermore, the GUI can be improved to cater for a large knowledge base so the user can easily view the statements in the knowledge base.

Justification for entailment is similar to kernels for belief base revision [18]. There are existing works for belief base revision in Propositional Logic [38]. Therefore, a necessary comparison between the two is needed to identify correlations.

Bibliography

[1] SAT4J SAT Solver, `https://www.sat4j.org/index.php`, last accessed: 2022-08-29

[2] The Tweety Project, `https://tweetyproject.org/`, last accessed: 2022-08-29

[3] Arnold, K., Gosling, J., Holmes, D.: The Java programming language. Addison Wesley Professional (2005)

[4] Baader, F., Horrocks, I., Lutz, C., Sattler, U.: Introduction to description logic. Cambridge University Press (2017)

[5] Baader, F., Penaloza, R.: Axiom pinpointing in general tableaux. Journal of Logic and Computation **20**(1), 5–34 (2010)

[6] Ben-Ari, M.: Mathematical logic for computer science. Springer Science & Business Media (2012)

[7] Bird, S., Klein, E., Loper, E.: Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc." (2009)

[8] Büning, H.K., Lettmann, T.: Propositional logic: deduction and algorithms, vol. 48. Cambridge University Press (1999)

[9] Casini, G., Meyer, T., Paterson-Jones, G., Varzinczak, I.: KLM-style defeasibility for restricted first-order logic. In: International Joint Conference on Rules and Reasoning. pp. 81–94. Springer (2022)

[10] Casini, G., Meyer, T., Varzinczak, I.: Taking defeasible entailment beyond rational closure. In: European Conference on Logics in Artificial Intelligence. pp. 182–197. Springer (2019)

[11] Chama, V.: Explanation for defeasible entailment. Master's thesis, Faculty of Science, University of Cape Town (2020), https://open.uct.ac.za/handle/11427/32206

[12] Chávez-Bosquez, O., Parra, P.P., McAreavey, K.: On the development of a logic calculator: a novel tool to perform logical operations. In: Latin American Workshop on New Methods of Reasoning. pp. 9–16 (2016)

[13] Cheng, J.: EnCal: An automated forward deduction system for general-purpose entailment calculus. In: Advanced IT Tools, pp. 507–514. Springer (1996)

[14] Eén, N., Sörensson, N.: An extensible SAT-solver. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 502–518. Springer (2003)

[15] Etherington, D.W.: Formalizing nonmonotonic reasoning systems. Artificial Intelligence **31**(1), 41–85 (1987)

[16] Freund, M.: Preferential reasoning in the perspective of Poole default logic. Artificial Intelligence **98**(1-2), 209–235 (1998)

[17] Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M.: SAT competition 2020. Artificial Intelligence **301**, 103572 (2021)

[18] Gärdenfors, P.: Belief revision. No. 29, Cambridge University Press (2003)

[19] Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: Semantic characterization of rational closure: From propositional logic to description logics. Artificial Intelligence **226**, 1–33 (2015)

[20] Greiner, R., Smith, B.A., Wilkerson, R.W.: A correction to the algorithm in Reiter's theory of diagnosis. Artificial Intelligence **41**(1), 79–88 (1989)

[21] Horridge, M.: Justification based explanation in ontologies. Ph.D. thesis, University of Manchester UK (2011)

[22] Kaliski, A.: An overview of KLM-style defeasible entailment. Master's thesis, Faculty of Science, University of Cape Town (2020), https://open.uct.ac.za/handle/11427/32743

[23] Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: The Semantic Web, pp. 267–280. Springer (2007), https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=59c96e8db11c7127d01becb2b1952baf219d207b

[24] Krasner, G.E.: A cookbook for using model-view-controller user interface paradigm in smalltalk-80. Journal of Object Oriented Programming **1**(3), 26–49 (1988)

[25] Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence **44**(1-2), 167–207 (1990)

[26] Lehmann, D., Magidor, M.: What does a conditional knowledge base entail? Artificial Intelligence **55**(1), 1–60 (1992)

[27] McDermott, D., Doyle, J.: Non-monotonic logic I. Artificial intelligence **13**(1-2), 41–72 (1980)

[28] Meyer, T., Moodley, K., Sattler, U.: DIP: A defeasible-inference platform for OWL ontologies. CEUR Workshop Proceedings (2014)

[29] Minker, J.: An overview of nonmonotonic reasoning and logic programming. The Journal of Logic Programming **17**(2-4), 95–126 (1993)

[30] Moodley, K., Meyer, T., Varzinczak, I.J.: A defeasible reasoning approach for description logic ontologies. In: Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference. pp. 69–78 (2012)

[31] Mylopoulos, J.: An overview of knowledge representation. ACM SIGART Bulletin (74), 5–12 (1980)

[32] Pollock, J.L.: A theory of defeasible reasoning. International Journal of Intelligent Systems **6**(1), 33–54 (1991)

[33] Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence **32**(1), 57–95 (1987)

[34] Sloman, S.A.: The empirical case for two systems of reasoning. Psychological bulletin **119**(1), 3 (1996)

[35] Smullyan, R.M.: First-order logic. Courier Corporation (1995)

[36] Wang, P.: Natural language processing by reasoning and learning. In: International Conference on Artificial General Intelligence. pp. 160–169. Springer (2013)

[37] Wang, S., Meyer, T., Moodley, D.: Defeasible justification using the KLM framework. In: Artificial Intelligence Research: Third Southern African Conference, SACAIR 2022, Stellenbosch, South Africa, December 5–9, 2022, Proceedings. pp. 187–201. Springer (2022)

[38] Wassermann, R.: An algorithm for belief revision. In: Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning. pp. 345–352 (2000)

[39] Wotawa, F.: A variant of Reiter's hitting-set algorithm. Information Processing Letters **79**(1), 45–51 (2001)