Scalable Defeasible Reasoning

Project Proposal

Joel Hamilton
University of Cape Town
Cape Town, South Africa
HMLJOE001@myuct.ac.za

Daniel Park
University of Cape Town
Cape Town, South Africa
PRKJOO001@myuct.ac.za

Aidan Bailey
University of Cape Town
Cape Town, South Africa
BLYAID001@myuct.ac.za

ABSTRACT

Knowledge representation and reasoning (KRR) is an approach to artificial intelligence (AI) in which a system has some information about the world represented formally (a knowledge base), and is able to reason about this information. Defeasible reasoning is a non-classical form of reasoning that enables systems to reason about knowledge bases which contain seemingly contradictory information, thus allowing for exceptions to assertions. Entailment checking is the process of determining whether or not a statement can be inferred from a knowledge base. Currently, no programming framework exists that supports defeasible entailment for propositional logic. We aim to investigate the scalability of defeasible entailment algorithms, and implement these algorithms in a form that allows for incorporation into a larger framework. We also aim to implement a tool to generate knowledge bases which can be used for the testing of these defeasible entailment algorithms and the implementations thereof.

CCS CONCEPTS

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

KEYWORDS

artificial intelligence, knowledge representation and reasoning, defeasible reasoning, satisfiability solving

1 INTRODUCTION

Artificial Intelligence (AI) has been around for many years, with its two core aspects being machine learning (ML) and knowledge representation and reasoning (KRR) [2]. Our research will focus on the latter. Knowledge representation refers to the notion of using some formal set of symbols or notation in order to represent information about the world. Reasoning refers to the idea of drawing inferences from this information, with the key idea for this research being the use of algorithms to reason about information (i.e., automated reasoning). We will utilise logics (a mechanism for formalising ways to reason [1]) to represent information.

Propositional logic [1] is monotonic, which means the addition of new information cannot contradict any previous conclusions you could draw. This is not a "common-sense" approach to reasoning [3].

Reasoning in such a way limits the ability to model human reasoning, as the property of monotonicity does not hold in the way that humans think.

For example, if a human knows that it rains every Saturday, and that today is a Saturday, then it makes sense to conclude that it is raining today, however if we are then told that it is not raining today, a human would interpret the addition of this new fact to mean that we have simply come across an exception to the notion of it raining every Saturday.

Reasoning according to propositional logic simply notes that a contradiction has occurred, meaning our knowledge can never all be true, which means that any and all information can be inferred from what we know, rendering our knowledge useless. What may have allowed the additional information to not cause a contradiction is if the initial knowledge we had rather stated that "typically, it rains every Saturday". We will use a framework for nonmonotonic reasoning in which statements of the form "typically, something is the case" are allowed.

2 BACKGROUND

2.1 Propositional Logic

Propositional Logic [1] is a framework for modelling information about the world, in which statements (known as *formulas*) are built up using *propositional atoms*, which are sentences which can be assigned a *truth value* (true or false), and *Boolean operators* ($\neg, \wedge, \vee, \rightarrow, \leftrightarrow$). Formulas can be defined recursively as either being simply an atom (e.g. p), or if α and β are formulas in $\mathcal{L}$ (the set of all formulas), then so are $\neg\alpha, \alpha \wedge \beta, \alpha \vee \beta, \alpha \rightarrow \beta,$ and $\alpha \leftrightarrow \beta$.

An *interpretation* is a function $I : \mathcal{P} \rightarrow \{T, F\}$ which attributes a single truth value to each propositional atom, and $\mathcal{W}$ is the set of all interpretations. The truth value of a formula α under a given interpretation I is written as $I(\alpha)$, and if $I(\alpha)$ is true for some formula α, then we say I satisfies α, written $I \Vdash \alpha$. A *knowledge base* is a finite set of formulas, and it is said that an interpretation I *satisfies* a knowledge base $\mathcal{K}$ if for every $\alpha \in \mathcal{K}, I \Vdash \alpha$. A knowledge base that is satisfied by at least one interpretation I is said to be *satisfiable*, and I is then a *model* of $\mathcal{K}$. The set of all models of a knowledge base $\mathcal{K}$ or of all models of a formula α are denoted $Mod(\mathcal{K})$ and $Mod(\alpha)$ respectively.

2.2 Entailment

If a formula α is true in every model of $\mathcal{K}$ ($Mod(\mathcal{K}) \subseteq Mod(\alpha)$), then we say $\mathcal{K}$ *entails* α (denoted $\mathcal{K} \models \alpha$). Propositional entailment is monotonic, which means that the addition of new formulas to a knowledge base $\mathcal{K}$ should not contradict any previous inferences made from $\mathcal{K}$. This causes a problem when contradictory statements are added to a knowledge base, as it then means that there will be no models of $\mathcal{K}$, and thus any statement is true in every model of $\mathcal{K}$, so such a knowledge base is meaningless, as it entails everything.

We therefore require a framework for nonmonotonic reasoning. Our preferred approach to nonmonotonic reasoning is the KLM approach [11] [13].

Checking whether or not $\mathcal{K} \models \alpha$ can be reduced to checking the satisfiability of the knowledge base with $\neg\alpha$ added to it ($\mathcal{K} \cup \{\neg\alpha\}$), where the satisfiability check can be done using a satisfiability (SAT) solver, as discussed further in the related work section.

2.3 KLM Approach to Defeasible Reasoning

Defeasible reasoning is a form of reasoning which allows one to reason about knowledge bases which contain seemingly contradictory information, and allows for exceptions to general assertions. In the KLM Approach [11] [13], we have a way to model statements of the form $\alpha \mid\sim \beta$, which is read as "α typically implies β", which simply means that if α is true, then this is typically enough information to believe that β is also true. A detailed overview of this approach is provided by Kaliski in [10].

The notion of defeasible entailment (denoted $\mid\approx$) is not unique, i.e., there are many acceptable ways to infer information from a defeasible knowledge base (a knowledge base containing statements of the form $\alpha \mid\sim \beta$). Rational closure [13] and lexicographic closure [12] are two approaches to defeasible reasoning which both fall within the KLM approach, and relevant closure [4] and ranked entailment [13] are two which do not [5]. Thus, this research will focus on the first two.

2.4 Rational Closure

Rational closure is the most conservative form of defeasible entailment (i.e., it infers very little from a defeasible knowledge base), and can be defined both semantically and algorithmically, as laid out in [5].

2.4.1 Semantic Definition. A *ranked interpretation* is simply a ranking of all interpretations in $\mathcal{W}$ (the set of all interpretations), in order of typicality, starting at rank 0 (indicating the most typical interpretations) and ending at rank n, which will include those interpretations which are the least typical, but are still plausible), followed by a single infinite rank, indicating those interpretations which are impossible to occur. In ranked interpretations, no rank can be empty. Formally, a ranked interpretation is a function $R : \mathcal{W} \to \mathbb{N} \cup \{\infty\}$, such that $R(I) = 0$ for some $I \in \mathcal{W}$, and for every $r \in \mathbb{N}$, if $R(I) = r$, then for every j such that $0 \leq j \leq r$, there is an $I \in \mathcal{W}$ for which $R(I) = j$.

R *satisfies* a formula α if α is true in all non-infinite ranks of a ranked interpretation R, denoted $r \Vdash \alpha$. For defeasible statements (statements which use the $\mid\sim$ operator), we say R *satisfies* $\alpha \mid\sim \beta$ if in the lowest rank (the most typical rank) where α holds, β also holds.

We can construct an ordering on the set of all possible ranked interpretations for a knowledge base $\mathcal{K}$, where $R_1 \leq_{\mathcal{K}} R_2$ if for every interpretation $I \in \mathcal{W}$, $R_1(I) \leq R_2(I)$. There is a unique minimal element R_m of the ordering $\leq_{\mathcal{K}}$ such that for every other ranked interpretation R_i in the set of all ranked interpretations for a given knowledge base, $R_m \leq_{\mathcal{K}} R_i$, as shown in [9]. If $R_m \Vdash \alpha \mid\sim \beta$, then we say $\mathcal{K}$ defeasibly entails $\alpha \mid\sim \beta$ (denoted $\mathcal{K} \mid\approx \alpha \mid\sim \beta$).

2.4.2 Algorithmic Definition: Base Ranks Algorithm. Before showing the algorithm, we define the *materialisation* of a knowledge base $\mathcal{K}$ as

$$\overrightarrow{\mathcal{K}} = \{\alpha \rightarrow \beta : \alpha \mid\sim \beta \in \mathcal{K}\}$$

(1) First, we create a sequence of materialisations $E_0, E_1, ..., E_{n-1}, E_\infty$ where $E_0 = \overrightarrow{\mathcal{K}}$, and each $E_i = \{\alpha \rightarrow \beta \in E_{i-1} : E_{i-1} \models \neg\alpha\}$ for $i > 0$. (This essentially means that each E_i contains only those statements $\alpha \rightarrow \beta$ in E_{i-1} such that α can be proven false according to those statements not in E_{i-1}). This process terminates when $E_i = E_{i-1}$ (or if $E_{i-1} = \emptyset$) and we set n=i-1 and $E_\infty = E_n$.
(2) We then create a ranking such that $E_n \setminus E_{n-1}$ is in the bottom rank, (call this R_∞) $E_{n-2} \setminus E_{n-1}$ is in the next highest rank, and so on. The ranking becomes such that the classical statements in $\mathcal{K}$ are in the bottom rank (the infinite rank), and statements are more general the higher the rank they are in. If a statement $\alpha \rightarrow \beta$ is in a rank i, then it is said that $\alpha \rightarrow \beta$ has base rank i. What this means is that in every ranked interpretation r of E_i, $\alpha \rightarrow \beta$ will be true in at least one of the interpretations in the most typical rank of r.

2.4.3 Checking Defeasible Entailment. Input: a knowledge base $\mathcal{K}$ and a defeasible implication statement $\alpha \mid\sim \beta$.

(1) First, we check if $\neg\alpha$ is entailed by those statements in the infinite rank.
(2) We then remove sets of classical implications rank by rank, starting at the highest rank and working our way down until we find a rank such that all the statements in that rank and those remaining do not entail $\neg\alpha$ (thus, α is satisfiable w.r.t. the set of statements which are in the current rank and the remaining ranks).
(3) We then check whether or not these remaining statements entail $\alpha \rightarrow \beta$.
(4) If we get to the stage where we only have the bottom rank remaining, and the statements in this rank entail $\neg\alpha$ then we can simply conclude that it is the case that $\mathcal{K} \mid\approx \alpha \mid\sim \beta$.

This algorithm only returns that a statement is defeasibly entailed by a knowledge base if this is true in terms of the minimal ranked interpretation for the knowledge base [8].

2.5 Lexicographic Closure

Lexicographic closure [12] is a refinement of rational closure, in that the entailment check is done the same way, and only the ranking of statements is done differently. A detailed description of the intricacies of this approach is provided by Casini et al in [5]. Lexicographic closure is less conservative than rational closure in terms of how much it infers from a knowledge base.

Lexicographic closure uses a more refined ranking, where statements within ranks (according to the Base Rank algorithm) are ranked amongst themselves, thus preserving the original ranking. We consider all possible refined rankings of these statements, and in the case of determining whether or not $\mathcal{K} \mid\approx \alpha \mid\sim \beta$, when checking whether or not $\overrightarrow{\mathcal{K}} \models \neg\alpha$, we remove statements one at a time such that as few statements as possible are removed where the remaining statements do not entail $\neg\alpha$. If this is not able to occur,

then we remove the entire top rank and continue similarly with a refinement of those statements in the following rank, etc.

Once the remaining statements do not entail $\neg\alpha$, we check whether these statements classically entail $\alpha \rightarrow \beta$. If we reach the stage where only one rank is remaining and the remaining statements still entail $\neg\alpha$, then we can return that $\mathcal{K} \approx \alpha \mid\sim \beta$.

The following is an example which may better explain how the entailment checks for rational and lexicographic closure differ:

If we have a knowledge base

$$\mathcal{K} = \{b \mid\sim f, p \rightarrow b, b \mid\sim w, p \mid\sim \neg f\}$$

which has been ranked according to the Base Ranks algorithm as follows:

Rank 0	$b \rightarrow f, b \rightarrow w$
Rank 1	$p \rightarrow \neg f$
Rank ∞	$p \rightarrow b$

Figure 1: Base Ranking of Knowledge base $\mathcal{K}$

And we wish to investigate whether or not the statement $p \mid\sim w$ is entailed by the knowledge base, then:

2.5.1 *Entailment Check using Rational Closure.*

(1) We check if $\neg p$ is entailed by $\overrightarrow{\mathcal{K}}$. It is, so we throw away the top rank.

Rank 0	~~$b \rightarrow f, b \rightarrow w$~~
Rank 1	$p \rightarrow \neg f$
Rank ∞	$p \rightarrow b$

Figure 2: Removal of rank 0

(2) We then check if $\neg p$ is entailed by the remaining statements in ranks 1 and ∞. It is not, so we check whether the statement $p \rightarrow w$ is entailed by the remaining statements. It is not, so we conclude $\mathcal{K} \not\approx p \mid\sim w$.

2.5.2 *Entailment Check using Lexicographic Closure.*

(1) Consider all possible refined rankings of $\overrightarrow{\mathcal{K}}$ such that the statements in rank 0 have been ranked amongst themselves.

Rank 0	$b \rightarrow f$ $b \rightarrow w$
Rank 1	$p \rightarrow \neg f$
Rank ∞	$p \rightarrow b$

Figure 3: Refined Ranking A

Rank 0	$b \rightarrow w$ $b \rightarrow f$
Rank 1	$p \rightarrow \neg f$
Rank ∞	$p \rightarrow b$

Figure 4: Refined Ranking B

(2) We check if $\overrightarrow{\mathcal{K}} \models \neg p$. It does, so we remove the top statement from each of our possible refined rankings.

Rank 0	~~$b \rightarrow f$~~ $b \rightarrow w$
Rank 1	$p \rightarrow \neg f$
Rank ∞	$p \rightarrow b$

Figure 5: Refined ranking A with top statement removed

Rank 0	~~$b \rightarrow w$~~ $b \rightarrow f$
Rank 1	$p \rightarrow \neg f$
Rank ∞	$p \rightarrow b$

Figure 6: Refined ranking B with top statement removed

(3) We then check if the remaining statements in each of our refined rankings entail $\neg p$. We see that the remaining statements in refined ranking A do not entail $\neg p$, and thus we check if these statements entail $p \rightarrow w$. This is the case, so we conclude that $\mathcal{K} \approx p \mid\sim w$.

Thus, it is clear that rational closure (RC) is more conservative than lexicographic closure (LC), as we were able to conclude the statement $p \mid\sim w$ when using LC but not when using RC.

3 PROJECT DESCRIPTION

3.1 Overview of Problem

There currently exists no implementation of a satisfiability checker for defeasible reasoning in the propositional logic context. While there are many implementations of classical reasoners, this style of reasoning is very limited, due to the inability to maintain the property of monotonicity in the presence of contradictory information, as discussed in section 2.2.

While there exists a defeasible reasoner implementation which works for description logics [15] which could theoretically work for propositional logic knowledge bases (which have been encoded as description logics), translating large knowledge bases while preserving the meaning of the knowledge is a tedious task.

There has also been no work to improve the computational scalability of the defeasible reasoning approaches discussed in the previous section, namely rational closure [13] and lexicographic closure [12]. Another key issue that these two previous issues highlight is the lack of readily available propositional logic knowledge bases for testing purposes.

3.2 Why is it important?

Currently, there exists no tool for automated reasoning about propositional defeasible knowledge bases. Research thus far has focused largely on developing theoretical approaches to nonmonotonic reasoning [10], but an implementation of some of these approaches would result in a significant contribution to the logic-based AI research community, as it would allow research to focus more on the viability of applying these approaches in useful real-world contexts (e.g., in robots, or decision-making systems in industries such as law or medicine).

Also, characteristics of these theoretical approaches which hinder the implementational viability thereof may be identified, thus also contributing towards the study of theoretical approaches to nonmonotonic reasoning, and automated reasoning in general.

4 PROBLEM STATEMENT: RESEARCH QUESTIONS & AIMS

4.1 Research Questions

4.1.1 Joel Hamilton.

- How can the rational closure (RC) algorithm be optimised to result in better execution time (improved scalability) for larger knowledge bases (larger number of formulas)?
- How does the runtime of both naive and optimised RC implementations compare to an equivalent tool for description logics?
- How does the performance of these RC implementations change for knowledge bases of a constant size but with varying structures (e.g. number of ranks, distribution of ranks, etc)?

4.1.2 Daniel Park.

- How can the lexicographic closure (LC) algorithm be optimised to result in better execution time (improved scalability) for larger knowledge bases?
- How does the performance of these LC implementations change for knowledge bases of a constant size but with varying structures (e.g. number of ranks, distribution of ranks, etc)?

4.1.3 Aidan Bailey.

- How efficient, useful and easy to use is our approach to knowledge base generation such that it can be used to test the LC and RC implementations in terms of soundness and completeness?
- Which parameters found for knowledge base generation are most useful for testing the RC and implementations in terms of their efficiency, scalability, and robustness?

4.2 Project Aims

The key aims of this project are:

- To prototype a reasoner tool which reasons about defeasible knowledge bases according to rational closure.
- To improve the scalability of such a rational closure implementation through the use of optimisation approaches.
- To develop a prototype reasoner to perform entailment checking of defeasible knowledge bases according to lexicographic closure.
- To improve the scalability of such a lexicographic closure implementation through the use of optimisation approaches.
- To implement an experimental tool for the generation of knowledge bases to be used for reasoner testing purposes.
- To determine which factors contribute towards the number of ranks and the distribution of statements over the ranks in a ranked knowledge base.

5 PROCEDURES & METHODS

The implementational aspects of this project are divided into three major sections: A prototype implementation of defeasible reasoners using Rational Closure (RC) (1) and Lexicographic Closure (LC) (2), and the construction of a knowledge base generator (3). The OpenJDK toolkit will be used with the defeasible reasoner prototypes being built in Java and the knowledge base generator being built in Scala.

We will start by studying the KLM approach for extending classical propositional logic with defeasible reasoning. We will then develop simple prototype defeasible reasoners which use RC and LC, respectively, as in [5]. These tools will allow users to input a defeasible knowledge base together with a defeasible statement, and the reasoner will check if the statement is defeasibly entailed by the knowledge base (according to either RC or LC). A knowledge base generator will be constructed to aid in the testing of the prototype defeasible reasoners.

5.1 Knowledge Base Generator

In order to gain an understanding of how basic knowledge bases could be generated, the fundamental makeup of a classical knowledge base will be studied. We will begin by defining a suitable structure for a classical knowledge base generator. We will then analyse the structure of defeasible knowledge bases, the KLM rationality properties. Given that classical knowledge bases can easily be encoded as defeasible ones, we will similarly amend the initial classical knowledge base generator design in order to accommodate the KLM rationality properties and allow for simple defeasible knowledge base generation.

Further analysis will then be done on how lexicographic and rational closure work, in order to identify a collection of potential knowledge generation parameters. This analysis can be categorized into two key ideas, namely, how the ranking of a knowledge base takes place (1) , and how defeasible entailment is checked using this ranking (2). Measures of scalability and robustness will be identified pertaining to these two areas which will speak to which potential parameters could be incorporated into our knowledge base generator (e.g., number of ranks of a ranked knowledge base, distribution of formulas over ranks). A refined collection of these parameters will then be implemented in the knowledge base generator. The knowledge base generation tool will be evaluated based the extent that changing different parameters has on the prototype defeasible reasoners' test metrics.

5.2 Prototype Reasoners

The implementation of rational and lexicographic closure will require the study of their respective algorithms, and specifically, an understanding of their reducibility to classical satisfiability solving. The correctness of these implementations will be checked by seeing whether or not their output is correct (i.e., satisfies the KLM rationality properties) when provided a defeasible knowledge base (generated by our knowledge base generator) and a defeasible statement as input. After a satisfactory naïve implementation has been constructed, the next step will be to use different optimisation heuristics to obtain better performance. The efficiency, robustness and scalability of the reasoner will be measured in relation to a variety of different knowledge bases using test metrics such as resource usage and execution time. Comparisons will be made between the naïve implementation, the optimised implementation and the similar description logics tool [15]. The robustness and scalability will be tested by using different structures of knowledge bases, potentially involving different numbers of ranks and different rank sizes.

The process of acquiring these test metrics will be automated using a testbed.

A possible optimisation is the idea of trying different search algorithms other than linear search for determining which ranks to throw away for rational closure, or which statements to throw away in the case of lexicographic closure. Since lexicographic closure is essentially refinement of rational closure for entailment checking, we believe that some approaches to optimising rational closure may likely work for lexicographic closure as well, and potentially vice-versa.

Our prototypes will be command line executable. Provided that every other aspect of the project has been completed, we will likely wrap our tools in an API for the purpose of integration into a larger platform/framework.

5.3 Possible Challenges

- Finding an easy-to-use SAT solver library that accommodates a widely used propositional logic format (DIMACS CNF).
- Identifying useful optimization heuristics.
- Identifying a suitable generic approach to knowledge base generation that can be applied to different logics.
- Identifying useful parameters for knowledge base generation that will create knowledge bases with varying structures (numbers of ranks, distribution of formulas across ranks, etc).

6 ETHICAL, PROFESSIONAL, AND LEGAL ISSUES

The algorithms built upon were conceived in openly published academic literature and thus there is no issue of intellectual property right other than the required acknowledgements being given to the creators. The work produced will be freely available and will eventually be implemented into an open source logical reasoning tool.

The Java and Scala facilities provided by the OpenJDK will be used to develop the implementations of these algorithms. The OpenJDK has a "GPL v2 with the Classpath Exception" license that stipulates the license of the code and executables produced using the OpenJDK toolset is up to the user's discretion and as such there will be no legal issues here either.

There is no requirement of user (or animal) testing and no handling of private user data. This implies no possible breach of the POPI act or requirement of ethical clearance.

Conclusively, there are no foreseeable ethical, professional or legal issues within the scope of this project.

7 RELATED WORK

Rational closure (RC) and Lexicographic Closure (LC) both reduce to a series of classical entailment checks. This means it is at its core the problem of boolean satisfiability (SAT) and thus, research pertaining to this problem is highly relevant to our proposed project.

7.1 Classical Satisfiability Solving

A *satisfiability (SAT) solver* is a system which computes the existence of a satisfying valuation (i.e., a model) for a specified boolean formula. There are many such SAT solvers designed for classical propositional satisfiability checking, thus understanding the approaches in this field will aid in the search for a suitable SAT solver to be used in the prototype defeasible reasoners.

7.1.1 The DPLL Algorithm. *DPLL* [7] is a complete and sound SAT checking algorithm. DPLL uses backtracking (retracting the most recent assignment and performing a different assignment) to speculatively test various assignment combinations, along with Boolean Constant Propagation (BCP) which constantly ensures that all variables which need to be true to satisfy a problem are kept true. DPLL forms the basis of many modern SAT solvers.

7.1.2 The CDCL Process. GRASP [14, 17], introduced the *Conflict Driven Clause Learning (CDCL)* process, in which variable assignments which potentially cause conflicts are cached. This results in better execution, and unlike DPLL [7], this does not rely on chronological backtracking, and as such is more efficient.

7.1.3 Other Advancements. Several more advanced SAT solvers utilise low-level optimisations such as different storage structures and parallelism in order to obtain better performance [18], e.g., GridSAT [6] and Chaff [16]. Due to prolific research being done on developing efficient SAT solving algorithms, many efficient SAT solver implementations are now freely available.

7.2 Defeasible Satisfiability Solving

While several theoretical formal approaches to defeasible entailment exist, e.g, rational closure [13], lexicographic closure [12], there has been little work done studying the scalability and implementational viability of these algorithms.

There currently exists no SAT solver which determines whether or not a defeasible statement is in the rational or lexicographic closure of a given defeasible knowledge base. Some work has been done looking at implementing the ability to reason defeasibly in the description logics context [15], but similar implementations for the propositional case are absent from nonmonotonic reasoning literature, thus also opening up the opportunity for an investigation into the scalability of established LM-rational defeasible entailment algorithms such as rational closure [13] and lexicographic closure [12].

8 ANTICIPATED OUTCOMES

8.1 Experimental Tools Developed

A successful project will have contributed several experimental defeasible reasoner prototypes for use by researchers in the KRR field, in particular:

- One which uses a straightforward implementation of the rational closure (RC) algorithm as per [5].
- One which uses an optimised implementation of the RC algorithm, in order to provide better scalability for large knowledge bases, and robustness for knowledge bases with varying features (e.g., distribution of statements across ranks, differing numbers of ranks).
- One which uses a straightforward implementation of the lexicographic closure (LC) algorithm as per [5].

- One which uses an optimised implementation of the LC algorithm, in order to provide better scalability for large knowledge bases, and robustness for knowledge bases with varying features (e.g., distribution of statements across ranks, differing numbers of ranks).

The project will also deliver a knowledge base generator which is able to create knowledge bases according to prespecified criteria (e.g. the size of the knowledge base, the number of ranks when ranked according to the Base Rank algorithm, etc.)

8.2 Research Impact

The successful project will have determined:

- How the RC and LC algorithm implementations can be optimised in a way that results in better runtime for larger knowledge bases compared to their naive counterparts (where number of ranks are kept constant).
- How the runtime of the rational closure implementations (naive and optimised) compare to a similar tool which was developed for description logics [15].
- How the performance of the LC and RC implementations change according to parameters such as number of ranks.
- How useful and efficient our implementation of knowledge base generation is as a tool for testing prototype defeasible reasoner implementations.
- Which parameters for specifying knowledge base complexity are useful in terms of generating test cases for prototype defeasible reasoner implementations.

8.3 Success Factors

The project will be considered successful if:
Defeasible Reasoner Prototypes & Optimisation:

- The reasoner prototypes provide correct answers. This can be checked with test cases by using knowledge bases from the generator.
- The naïve and optimised prototypes using Rational Closure perform more efficiently than the existing equivalent description logics tool [15].
- The optimised reasoner prototypes which use the RC and LC algorithms perform more efficiently than their respective naïve implementations.
- We have identified useful optimisation approaches that result in improved scalability of these two defeasible entailment-checking algorithms.

Knowledge Base Generation

- Tool successfully generates knowledge bases which can be used to test the robustness and scalability of the rational and lexicographic closure defeasible reasoners.
- A collection of useful knowledge base generation parameters has been identified that result in a versatile and useful tool.

9 PROJECT PLAN

9.1 Risks

See 1.1 and 1.2 of the Appendix.

9.2 Timeline

See 2.2 of the Appendix.

9.3 Resources Required

The physical resources we will be using are textbooks, scholar papers and three computers. We will implement our algorithms using Java and Scala (OpenJDK), with appropriate IDE for Java and Scala development. We will need access to source code of publicly available SAT solvers, and a description logics defeasible reasoner [15] to compare our efficiency. We will be implementing our own knowledge base generator to produce testing data for our reasoners; thus, no knowledge base datasets will be required.

9.4 Deliverables

All common project deliverables, i.e., a literature review, project proposal, project proposal presentation, software feasibility demonstration, final project paper, final project demonstration, project poser, and project WebPage will all be produced. Multiple implementations for the rational closure and lexicographic closure algorithms will be produced. These will take the initial form of naive approaches, and then ones with optimisational heuristics that allow for large scale defeasible reasoning. A testbed and knowledge base generator will be constructed to be used for formalized testing of the defeasible reasoning algorithm implementations. Test metrics relating to each of these implementations, as well as a previous implementation of defeasible reasoning, will be produced. Finally, an ultimate comparison of the metrics collated from the naive implementations, previous implementations, and new efficient implementations will be produced, depicting the efficiency of the new implementations and speaking directly to the success of the project.

9.5 Milestones

See 2.1 of the Appendix.

9.6 Work Allocation

There are three major sections of project work that need allocation:

Student	Work Allocation
Joel Hamilton	The development of naïve and optimised defeasible reasoner prototypes using the Rational Closure algorithm, along with significant research on potential optimisation approaches.
Daniel Park	The development of naïve and optimised defeasible reasoner prototypes using the Lexicographic Closure algorithm, along with significant research on potential optimisation approaches.
Aidan Bailey	The construction of an experimental knowledge base generator to be used for providing test cases for testing the robustness and scalability of the defeasible reasoner prototypes, along with significant research on which parameters may result in knowledge bases differing in structure.

There is a possibility of implementing an API wrapper for our reasoner and generator. This will be done by all three members collectively.

REFERENCES

[1] Mordechai Ben-Ari. *Mathematical Logic for Computer Science, 3rd Edition*. Springer, 2012.

[2] Zied Bouraoui, Antoine Cornuéjols, Thierry Denœux, Sebastien Destercke, Didier Dubois, Romain Guillaume, João Marques-Silva, Jérôme Mengin, Henri Prade, Steven Schockaert, Mathieu Serrurier, and Christel Vrain. From shallow to deep interactions between knowledge representation, reasoning and machine learning (kay r. amel group), 12 2019.

[3] G. Casini, T. Meyer, K. Moodley, and I. Varzinczak. Towards practical defeasible reasoning for description logics. Jul 2013.

[4] Giovanni Casini, Thomas Meyer, Kodylan Moodley, and Riku Nortjé. Relevant closure: A new form of defeasible reasoning for description logics. In Eduardo Fermé and João Leite, editors, *Logics in Artificial Intelligence*, pages 92–106, Cham, 2014. Springer International Publishing.

[5] Giovanni Casini, Thomas Meyer, and Ivan Varzinczak. Taking defeasible entailment beyond rational closure. In Francesco Calimeri, Nicola Leone, and Marco Manna, editors, *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings*, volume 11468 of *Lecture Notes in Computer Science*, pages 182–197. Springer, 2019.

[6] Wahid Chrabakh and Rich Wolski. Gridsat: A chaff-based distributed sat solver for the grid. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, SC '03, page 37, New York, NY, USA, 2003. Association for Computing Machinery.

[7] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962.

[8] Michael Freund. Preferential reasoning in the perspective of poole default logic. *Artificial Intelligence*, 98(1):209–235, 1998.

[9] Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Gian Luca Pozzato. Semantic characterization of rational closure: From propositional logic to description logics. *Artif. Intell.*, 226:1–33, 2015.

[10] Adam Kaliski. An overview of klm-style defeasible entailment, 2020_.

[11] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial intelligence*, 44(1-2):167–207, 1990.

[12] Daniel Lehmann. Another perspective on default reasoning. *Annals of Mathematics and Artificial Intelligence*, 15(1):61–82, Mar 1995.

[13] Daniel Lehmann and Menachem Magidor. What does a conditional knowledge base entail? *Artificial intelligence*, 55(1):1–60, 1992.

[14] Joaão P. Marques-silva and Karem A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48:506–521, 1999.

[15] Kody Moodley, Thomas Meyer, and Ivan Varzinczak. A protégé plug-in for defeasible reasoning. volume 846, 06 2012.

[16] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, page 530–535, New York, NY, USA, 2001. Association for Computing Machinery.

[17] João P. Marques Silva and Karem A. Sakallah. Grasp—a new search algorithm for satisfiability. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '96, page 220–227, USA, 1997. IEEE Computer Society.

[18] Yakir Vizel, Georg Weissenbacher, and Sharad Malik. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035, 2015.

Scalable Defeasible Reasoning Proposal Appendix

1 Risks

1.1 Risk Identification

ID	Risk	Probability	Impact
1	Unavailability of supervisor	Very unlikely	Moderate
2	Loss of work due to computer failure	Unlikely	Significant
3	The group having poor communication leading to disagreement or conflict	Very Unlikely	Significant
4	Poor time management	Possible	Significant
5	A team member is unable to complete their assigned task	Possible	Significant
6	Unable to optimize the algorithm	Possible	Moderate
7	Not being able to generate knowledge bases which vary in terms of structure.	Possible	Moderate

1.2 Risk Mitigation, Monitoring, and Management

ID	Mitigation	Monitoring	Management
1	Check supervisor's availability regularly	Communicate through email and confirm meetings and agreements	Continue the project regardless
2	Save the work regularly and save it to the Cloud	Check for load-shedding schedule regularly	Restore as much data as possible and continue work with what is available
3	Keep communicating with partners through email or IM, making sure no one is missing out	Make sure everyone is agreed on a decision	Get advice from supervisors upon a disagreement, and resolve miscommunications clearly and as soon as possible.
4	Follow the timeline that every member is agreed on	Check if every member of the team is progressing according to the timeline	Create a new timeline with adjusted time estimates
5	Communicate and assist team members when they get stuck	Check partners progress regularly	Split the work evenly and independently
6	Study as many optimizations methods as possible and implement them to the algorithms	Check whether the algorithms are being optimised in the right way by comparing the performance with the previous versions	Move focus away from optimisation and focus on constructing a useful tool
7	Study how the algorithms work and attempt multiple approaches to construct knowledge bases of varying structures.	Keep checking whether the generated knowledge bases are differing in terms of structure (e.g. number of ranks, etc)	Come up with additional knowledge base characteristics which can test the robustness of our reasoners.

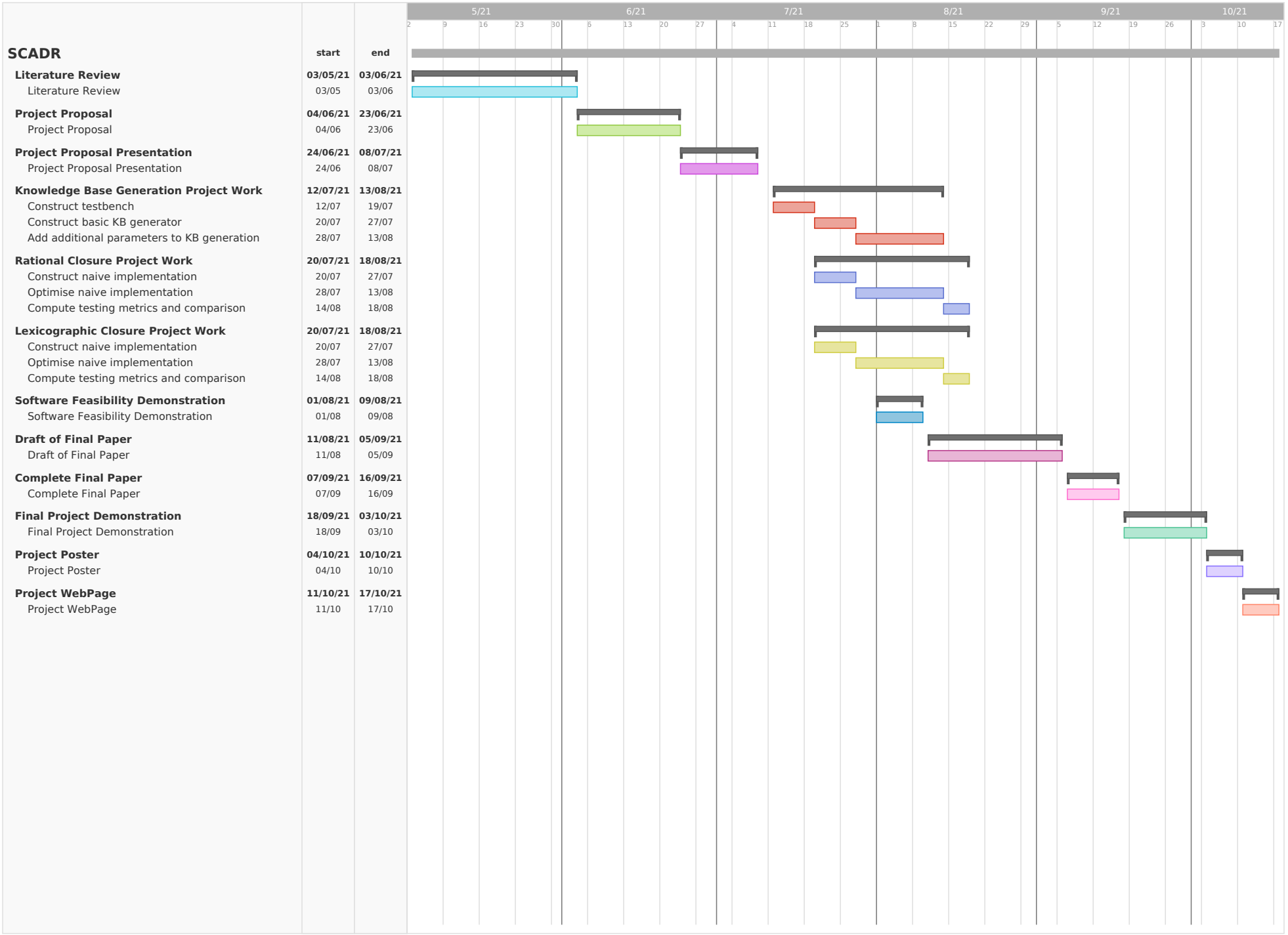
2 Timeline

2.1 Milestones & Tasks

MILESTONES	Start Date	End Date
PHASE 1: Project Planning	03/05/2021	08/07/2021
Literature Review	03/05/2021	03/06/2021
Project Proposal	04/06/2021	23/06/2021
Project Proposal Presentation	24/06/2021	08/07/2021
PHASE 2: Project Work	12/07/2021	18/08/2021
Knowledge Base Generation Project Work	12/07/2021	13/08/2021
Construct testbench	12/07/2021	19/07/2021
Construct basic knowledge base generator	20/07/2021	27/07/2021
Add additional parameters to knowledge base generation	28/07/2021	13/08/2021
Rational Closure Project Work	20/07/2021	18/08/2021
Construct naive implementation and base testing metrics	20/07/2021	27/07/2021
Optimise naive implementation	28/07/2021	13/08/2021
Compute new testing metrics and comparison	14/08/2021	18/08/2021
Lexicographic Closure Project Work	20/07/2021	18/08/2021
Construct naive implementation and base testing metrics	20/07/2021	27/07/2021
Optimise naive implementation	28/07/2021	13/08/2021
Compute new testing metrics and comparison	14/08/2021	18/08/2021
PHASE 3: Project Completion	01/08/2021	17/10/2021
Software Feasibility Demonstration	01/08/2021	09/08/2021
Draft of Final Paper	11/08/2021	05/09/2021
Complete Final Paper	07/09/2021	16/09/2021
Final Project Demonstration	18/09/2021	03/10/2021
Project Poster	04/10/2021	10/10/2021
Project WebPage	11/10/2021	17/10/2021

2.2 Gantt Chart

On final page of the appendix.



SCADR	start	end
Literature Review	**03/05/21**	**03/06/21**
Literature Review	03/05	03/06
Project Proposal	**04/06/21**	**23/06/21**
Project Proposal	04/06	23/06
Project Proposal Presentation	**24/06/21**	**08/07/21**
Project Proposal Presentation	24/06	08/07
Knowledge Base Generation Project Work	**12/07/21**	**13/08/21**
Construct testbench	12/07	19/07
Construct basic KB generator	20/07	27/07
Add additional parameters to KB generation	28/07	13/08
Rational Closure Project Work	**20/07/21**	**18/08/21**
Construct naive implementation	20/07	27/07
Optimise naive implementation	28/07	13/08
Compute testing metrics and comparison	14/08	18/08
Lexicographic Closure Project Work	**20/07/21**	**18/08/21**
Construct naive implementation	20/07	27/07
Optimise naive implementation	28/07	13/08
Compute testing metrics and comparison	14/08	18/08
Software Feasibility Demonstration	**01/08/21**	**09/08/21**
Software Feasibility Demonstration	01/08	09/08
Draft of Final Paper	**11/08/21**	**05/09/21**
Draft of Final Paper	11/08	05/09
Complete Final Paper	**07/09/21**	**16/09/21**
Complete Final Paper	07/09	16/09
Final Project Demonstration	**18/09/21**	**03/10/21**
Final Project Demonstration	18/09	03/10
Project Poster	**04/10/21**	**10/10/21**
Project Poster	04/10	10/10
Project WebPage	**11/10/21**	**17/10/21**
Project WebPage	11/10	17/10