Extending Defeasible Reasoning beyond Rational Closure

Julia Cotterrell
cttjul001@myuct.ac.za
University of Cape Town
Cape Town, South Africa

Nevaniah Gouden
gndnev002@myuct.ac.za
University of Cape Town
Cape Town, South Africa

Jethro Dunn
dnnjet002@myuct.ac.za
University of Cape Town
Cape Town, South Africa

ABSTRACT

Defeasible reasoning allows systems to make conclusions that can be withdrawn when exceptions arise, addressing limitations in classical logic. Rational Closure (RC) formalises this via ranked models, but current implementations lack scalability, usability, and interpretability. This project develops an optimised, modular RC implementation integrated with a topic-aware knowledge base generator and an interactive debugger. Key features include natural language explanations, a simplified user interface, and performance improvements through parallel entailment, SAT memoization, and hybrid rank transversals. The resulting tool enhances both practical reasoning and exploration of defeasible logic in complex domains.

KEYWORDS

artificial intelligence, knowledge representation and reasoning, nonmonotonic reasoning, defeasible reasoning, rational closure, lexicographic closure, knowledge base generator, user interface, debugger

1 INTRODUCTION

In real-world domains, reasoning often involves dealing with uncertainty, exceptions, and conflicting information. These are areas where classical logic falls short due to its rigid, monotonic nature [16]. Defeasible reasoning (DR) provides a more adaptable alternative, allowing conclusions to be revised as new information becomes available. Among the formal approaches to this kind of reasoning, Rational Closure (RC) stands out for its structured use of ranked models, which helps distinguish typical scenarios from exceptional ones and supports a more cautious inference of defeasible conclusions [4].

However, despite its solid theoretical foundation, existing implementations of RC face several practical challenges. They often struggle with inefficiency, poor scalability, and limited usability, especially for those without a background in formal logic [5]. There is also a gap between how knowledge is formally represented and how people naturally think about it, which creates barriers to adoption. While tools like SILK and OntoDebug offer helpful interfaces and debugging support, they primarily focus on ontologies and rule-based systems, rather than broader applications of DR [10, 22].

This project aims to overcome these challenges by developing a scalable, modular, and user-friendly DR system. It features an optimised implementation of RC, a topic-aware knowledge base generator (KBG), and an interactive debugger that provides clear, natural language explanations. By combining algorithmic enhancements, such as parallel entailment checking and SAT memoization, with a focus on user interaction and clarity, this system is designed to serve both researchers and domain professionals. Built on the TweetyProject framework, it aims to improve accessibility, interpretability, and performance in reasoning about defaults and exceptions [25].

2 BACKGROUND

2.1 Propositional Logic

Propositional logic provides a formal foundation for knowledge representation and reasoning. To achieve this, we first formalise the notion of truth by assigning values of *true* or *false* [11]. This enables the evaluation of statements without the ambiguity of natural language. However, its monotonic nature limits its ability to handle exceptions [16], highlighting the need for alternative reasoning approaches such as rational and lexicographic closure.

2.1.1 Syntax. Propositional formulas consist of two main components: *atoms* and *boolean operators*. Atoms, or atomic propositions, are unbounded symbols represented by lowercase Latin letters e.g., $p, q, r, s, \ldots$ forming the set $P = \{p, q, r, s, \ldots\}$. Boolean operators connect atoms to form formulas and include $\wedge$ (and), $\vee$ (or), $\rightarrow$ (if...then), $\leftrightarrow$ (if and only if) and $\neg$ (not). Complex formulas are constructed by combining atoms using these operators [2].

2.1.2 Semantics. Propositional logic uses *valuations* to assign truth values to atoms. A valuation u is a function $u : \mathcal{P} \rightarrow \{T, F\}$, where $\mathcal{P}$ is the set of propositional atoms, and each atom $p \in \mathcal{P}$ is mapped to *true* (T) or *false* (F) [11]. The set of all possible valuations is denoted by $\mathcal{U}$.

2.1.3 Entailment. Entailment is the logical consequence of a knowledge base, defined as a finite set of propositional formulas [11]. A formula α is entailed by a knowledge base (KB) $\mathcal{K}$ if every valuation that satisfies $\mathcal{K}$ also satisfies α [11]. Formally, $\mathcal{K} \models \alpha$ if and only if for all $u \in \mathcal{U}$, $u \Vdash \mathcal{K}$ implies $u \Vdash \alpha$.

2.2 Defeasible Reasoning

Classical propositional logic is monotonic, meaning that conclusions remain unchanged even when new information is added. However, this is not ideal for modelling the real world, where new information often alters conclusions. Real-world reasoning often involves exceptions and conclusions that may need to be retracted in light of new information. This motivates the need for *DR*, which allows such flexibility. The KLM framework formalises DR by introducing the notion of *typicality*, enabling statements such as "typically, α implies β" [12].

Using the KLM framework, ranked interpretations ($\mathcal{R}$) can be created, where the interpretations are ordered according to their degree of typicality [23]. Valuations are ranked from 0 to ∞ with 0 being the most typical and ∞ being impossible [15]. For example:

∞	$pq\bar{r}$
2	$pqr\ p\bar{q}r\ \bar{p}qr$
1	$\overline{pq}r\ p\overline{q}r$
0	$pq\bar{r}\ \overline{pq}r$

Figure 1: Possible ranked interpretation of $\mathcal{P}$

Defeasible entailment, denoted $\mid\!\sim$, is used at the meta-level (between statements) to express *"typical following"*. Although defeasible entailment remains monotonic, it was extended through the introduction of ranked entailment and two additional rules (inclusion and classical preservation) [5]. Rational extension, introduced by Casini et al. [4], creates a stronger class of defeasible entailment relations that incorporate RC as a non-monotonic core.

2.3 Rational Closure

RC is defined both semantically, using a minimal ranked model, and syntactically, through base ranks [4]. Both approaches support DR by ranking formulas based on their typicality, denoted $\mid\sim$.

2.3.1 Minimal Ranked Model. A minimal ranked model $\mathcal{R}^{RC}_{\mathcal{K}}$ assigns lower ranks to more typical valuations using ∞ for impossible ones [4, 9]. It satisfies all formulas in $\mathcal{K}$ while minimising exceptionality:

$$\forall \mathcal{R}' \text{ ranked model of } \mathcal{K}, \quad \mathcal{R}^{RC}_{\mathcal{K}}(u) \leq \mathcal{R}'(u) \text{ for all } u.$$

For example, given atoms c = car, p = petrol, e = electric car, with $\mathcal{K} = \{c \mid\sim p, e \mid\sim \neg p\}$, RC does not entail $e \mid\sim p$ because the minimal e-worlds falsify p [9, 11].

2.3.2 Base Ranks. Base ranks syntactically determine exceptionality. A formula α is exceptional if $\mathcal{K} \mid\!\approx_R \top \mid\sim \neg\alpha$, or if $\overrightarrow{\mathcal{K}} \models \neg\alpha$ [5, 11]. To compute base ranks, we define a sequence of KBs:

$$\mathcal{E}^{\mathcal{K}}_0 := \mathcal{K}$$

$$\mathcal{E}^{\mathcal{K}}_i := \varepsilon(\mathcal{E}^{\mathcal{K}}_{i-1}) \text{ for } 0 < i < n$$

$$\mathcal{E}^{\mathcal{K}}_\infty := \mathcal{E}^{\mathcal{K}}_n \text{ where } n \text{ is the smallest index for which } \mathcal{E}^{\mathcal{K}}_n = \mathcal{E}^{\mathcal{K}}_{n+1}$$

Here, $\varepsilon(\mathcal{E}^{\mathcal{K}}_{i-1})$ denotes the set of defaults with antecedents that are exceptional with respect to $\mathcal{E}^{\mathcal{K}}_{i-1}$ [5]. Since $\mathcal{K}$ is finite, this process always terminates. The *base rank* of a formula α, written $br_{\mathcal{K}}(\alpha)$, is the smallest index r such that α is no longer exceptional in $\mathcal{E}^{\mathcal{K}}_r$:

$$br_{\mathcal{K}}(\alpha) := \min\{r \mid \overrightarrow{\mathcal{E}^{\mathcal{K}}_r} \not\models \neg\alpha\}$$

For a default $\alpha \mid\sim \beta$, the base rank is assigned to the antecedent:

$$br_{\mathcal{K}}(\alpha \mid\sim \beta) := br_{\mathcal{K}}(\alpha)$$

A default is included in the RC if:

$$\mathcal{K} \mid\!\approx_{RC} \alpha \mid\sim \beta \iff br_{\mathcal{K}}(\alpha) < br_{\mathcal{K}}(\alpha \wedge \neg\beta) \text{ or } br_{\mathcal{K}}(\alpha) = \infty$$

There is also a direct link between base ranks and the minimal model $\mathcal{R}^{RC}_{\mathcal{K}}$:

$$br_{\mathcal{K}}(\alpha) = \min\{i \mid \exists v \in Mod(\alpha) \text{ with } \mathcal{R}^{RC}_{\mathcal{K}}(v) = i\}$$

This confirms that a formula's base rank matches the lowest rank among worlds that satisfy it [9].

Example: Given atoms c = car, p = petrol, e = electric car, and $\mathcal{K} = \{c \mid\sim p, e \mid\sim \neg p, e \rightarrow c\}$, e is initially exceptional, so $br_{\mathcal{K}}(e \mid\sim \neg p) = 1$, confirming that electric cars typically do not use petrol.

Note: This project uses RC. Other methods exist, such as lexicographic closure, which prioritises more specific defaults and refines RC by minimising violations of stronger defaults.

2.4 Knowledge base Generator

Knowledge base generation (KBG) is an integral part of the process of structuring how statements are ranked and presented. It focuses on creating logically correct statements that can be used to rank information [1]. KBG's are used by other algorithms such as RC to rank information and allow logical queries to be run and produce logically correct responses.

2.5 User Interface and debugger

The usability of DR systems is substantially influenced by the use of intuitive user interfaces (UIs) and debugging tools. These components help users interact with KBs, address queries, and identify inconsistencies in the reasoning output. Grosof et al. [10] developed SILK UI, which visually traces reasoning paths, enhancing interpretability. Schekotihin et al. [22] introduced OntoDebug, an interactive plug-in for Protégé that identifies missing information through user queries, while Coetzer et al. [6] improved upon this by addressing issues caused by multi-level exceptions in classical ontologies. The TweetyProject library [25] provides a foundational back-end for implementing reasoning systems, including support for propositional logic and defeasible entailments. This makes it a suitable foundation for building debuggers and interactive tools.

3 PROJECT DESCRIPTION

3.1 Problem Overview

There is significant potential to improve the current implementation of the RC algorithm, the UI/ debugger, and the KBG. This project aims to optimise the computational efficiency of the current RC algorithm, a central technique in DR. In addition to algorithmic improvement, the project will include developing a KBG that provides statements using natural language as well as atoms. To support usability and transparency, a UI and debugger will be implemented, allowing users to use a natural language (English) to interact with KBs, trace inferences, and identify conflicts or inconsistencies within the reasoning process.

3.2 Motivation of importance

RC offers a structured approach to reasoning with defaults and exceptions, but current implementations are often inefficient, particularly when dealing with large or complex knowledge bases. This is due to repeated exceptionality and entailment checks, which can result in redundant SAT solver calls and increased computation time. By optimising the underlying algorithm, the generation of KBs using natural language, and providing an intuitive, user-friendly interface with debugging capabilities, this project aims to allow DR to be more scalable, accessible, and transparent. These improvements are intended to benefit both researchers working in

nonmonotonic reasoning and end-users who require explainable, logic-driven systems.

4 PROBLEM STATEMENT

The RC algorithm becomes computationally expensive on large KBs due to repeated exceptionality and entailment checks, limiting scalability in practical reasoning systems. To address this, our project explores optimisations such as parallelised entailment checking, SAT sub-query memoization, and adaptive hybrid rank traversal to improve performance while keeping the implementation simple and efficient. Existing KBG produce synthetic, logically consistent datasets that are fast but lack meaning, making them hard to interpret. Our goal is to generate logically valid KBs that are also human-understandable. Finally, DR systems often yield unintuitive conclusions, with existing tools offering limited explanation or debugging, especially at scale. We propose a DR UI using natural language to improve interpretability, aimed at researchers working with complex KBs.

4.1 Aims

This project aims to enhance the scalability, usability, and accessibility of DR systems. On the back-end, parallelised entailment checking, SAT sub-query memoization, and adaptive hybrid rank traversal aim to reduce the execution time of the RC algorithm on large KBs, improving efficiency without compromising correctness.

Additionally, we aim to modify an existing KBG to map natural language phrases (from a given topic) to atoms. By doing this it can form logically correct statements which are reasonably understandable by a person. It also aims to integrate Large Language Models (LLMs) that will identify statements that make linguistic sense and those that do not.

An interactive, multi-modal defeasible UI will further improve transparency by visualising reasoning steps graphically, textually, and logically. If time permits, support for converting natural language into formal logic will be added, lowering the barrier for non-experts.

4.2 Research Questions

Optimisation:

1. How can the efficiency of the algorithm be improved without compromising integrity?
2. How would the performance of RC reasoning be affected by the impact of implementing parallelised entailment checking, SAT sub-query memoization, and adaptive hybrid rank traversal?
3. Given the different complexities of the KB, does the algorithm prove to be more efficient for all cases?
4. To what extent does implementing the optimisation techniques improve the scalability of RC across varying query set sizes and rank distributions?

KBG:

1. How can a KB be generated using natural language phrases which are coherent and logically correct?
2. How does the performance of a KB change with the additional process of allocating English phrases to atoms?
3. How can atoms efficiently be allocated natural language phrases?
4. Can LLM assist in verifying the coherency of KBs with natural language phrases?

UI/debugger:

1. In what ways can a defeasible debugger support users in understanding why specific conclusions are defeasibly entailed within a KB?
2. How do different explanation modalities (graphical, textual, and logical) compare in their effectiveness at conveying reasoning paths in defeasible logic systems?
3. What design strategies can be employed to ensure that defeasible debuggers remain scalable and performant when applied to large KBs?
4. Can be extended to include: What approaches can be used to transform natural language statements into formal DR rules?

5 METHODS AND PROCEDURES

5.1 Overview

This project will be executed based on an iterative approach. The focus of the project is to make the interaction with RC more intuitive and amiable. Algorithm design includes two main phases: computing base ranks and determining entailment, with optimisations to improve runtime efficiency. Phrase mapping, topic-aware labels, and coherence checks are integrated to enhance interpretability, supported by a simplified GUI. Implementation is in Java, using TweetyProject for logical operations, and optimisations such as parallel entailment checking, memoization, and rule indexing is incorporated. Evaluation will be conducted through benchmarking on varied KBs, measuring reasoning time, coherence, and explanation clarity, with statistical methods ensuring rigour and repeatability.

5.2 Development method

This is a research project, characterised by its iterative nature, adaptability, and emphasis on continuous feedback. The development will be structured around weekly meetings with our supervisor, serving as checkpoints. During these sessions, we will update our supervisor on progress, seek guidance for the upcoming sprint, and collaboratively decide on objectives. Given our differing timetables and workloads, development will be somewhat desynchronised, with team members working at varying paces. These weekly check-ins will ensure alignment, facilitate early identification of challenges, and help maintain steady momentum throughout the project.

5.3 Theoretical Foundation and Algorithm Design

Optimisation:

This project stems from the KLM framework, which models typicality using ranked interpretations, and supports the RC algorithm for DR [5, 14]. The KLM framework introduces a nonmonotonic consequence relation ($\alpha \mid\sim \beta$) governed by rationality postulates, particularly rational monotony (RM) [5, 14]. RC, proposed by Lehmann and Magidor [14], is a further extension which has been adopted. It extends the KLM framework by ranking valuations to differentiate typical from exceptional cases, enabling specific defaults to override general ones via minimal-ranked models. Its syntactic

form uses base ranks to assess exceptionality and prioritise defaults accordingly [5, 11].

The algorithm has two phases: *Base Rank* and *RC* [5]. In the Base Rank phase, defeasible implications are materialised to classical form, then ranked based on the exceptionality of their antecedents, forming partitions $\mathcal{R}_0, \mathcal{R}_1, \ldots, \mathcal{R}_\infty$ [11]. In the RC phase, a query $\alpha \mid\sim \beta$ is tested for exceptionality; if not exceptional, the algorithm checks whether $\alpha \rightarrow \beta$ is classically entailed by the relevant partitions.

KBG:

This part of the project will build on an existing KBG. Parallelisation and space-time trade-off optimisations have already been applied to this KBG, as well as some fine-tuning to create an efficient algorithm.

Initially, to replace atoms with natural language phrases, we will add dictionaries containing hardcoded phrases which will be automatically assigned to atoms when they are generated. These will be grouped by topic, where the user can select the topic. The next step is to use an API to a LLM where the program will send a statement with its English phrasing to check "Does this make sense for humans". If it does make sense then those phrases will be assigned to the atoms, otherwise, new phrases will be used and checked.

Additional features can be added such as a simple UI, alternative LLMs, improved query selection, and allowing dynamically generated phrases to be used.

UI/debugger:

This section of the project will initially focus on optimising the UI for DR systems. Building on prior work that explored logic-based user interfaces and debugging environments for ontologies and rule-based systems—such as the approaches proposed by Grosof et al. [10], Schekotihin et al. [22], and Coetzer et al. [6]. The project will design a multi-modal explanation environment capable of rendering reasoning steps graphically, textually, and logically.

The algorithmic back-end will be based on RC as implemented in the TweetyProject library [25], incorporating optimisations such as parallel entailment checking, SAT sub-query memoization, and hybrid traversal strategies to improve performance done in previous sections.

The natural language-to-formal logic transformation component will be introduced in a later development phase, once the interface and debugging tools are sufficiently mature. This phased approach ensures that the system remains credible, responsive, and interpretable before adding more complex natural language functionality.

5.4 Implementation and Optimisation

Optimisation:

This project builds on an existing Java implementation of the RC algorithm developed by previous students. The system will be modular, with separate components for base rank computation and defeasible entailment, using classical propositional logic with materialised defeasible rules. To improve efficiency and scalability, three key optimisations will be explored: *problem-independent* techniques, which improve performance across all KBs, and *problem-dependent* ones, effective based on KB structure.

Parallelised entailment checking and *SAT sub-query memoization* are problem-independent and reduce runtime via concurrency and redundancy elimination [8, 24]. *Adaptive hybrid rank traversal* is problem-dependent and optimises traversal based on the distribution of exceptionality [7].

Process: Entailment checking will be parallelised using Java's ForkJoinPool, running multiple queries concurrently. SAT memoization will use a thread-safe cache with canonicalised keys to eliminate repeated work. A hybrid traversal strategy will dynamically switch between binary and linear search depending on KB structure. Though these improvements add complexity in concurrency management and storage, the gains in speed, scalability, and adaptability justify the trade-off.

KBG:

This part of the project will be implemented primarily in Java, as the current KBG is coded in Java. Dictionaries will initially be implemented (using *HashMap*) and atoms with their assigned English phrases will also be stored in HashMaps. The API used will be Gemini 2.0 flash but this could be changed if a model that allows for more free requests is found or one that has faster response times.

Optimisation will primarily consist of attempting to reduce the number of queries sent to the API as this is likely to be the slowest part of the algorithm. Additional improvements to hash mapping and storing phrases will also be necessary to improve KBG speed.

UI/debugger:

The UI will be implemented in Java for the back-end, using libraries from the TweetyProject to handle logical representations and defeasible entailment computation [5]. It will utilise typescript for the front-end web design. The components will include a reasoning engine, an explanation generator, and an interface to interact with queries and entailments. This follows on from previous years honours and masters projects.

Testing will be conducted using JUnit and benchmarking with Java Microbenchmark Harness (JMH) to measure performance. Optimisations such as rule indexing and minimal model reuse will be explored to handle larger KBs efficiently [3, 17].

5.5 Evaluation and Comparison

This project will be evaluated across three components: optimisation strategies, the KBG, and the UI debugger. For optimisation, parallel entailment, SAT sub-query memoization, and hybrid rank traversal will be tested individually and together using varied knowledge bases (differing in size, rank depth, and distribution), with runtime averaged over multiple runs and analysed using non-parametric tests (e.g., Wilcoxon signed-rank). KBG evaluation will compare generation time to the original version, assess phrase coherence via human review and similarity checks, and use acceptance rates to measure the effectiveness of LLM-generated phrases. Unit tests will ensure logical correctness. The UI will be tested on diverse KBs and queries involving conflict overlap and varying entailment depths. Natural language transformation, explanation modes, and reasoning optimisations will be benchmarked against a baseline. Usability will be assessed using adapted heuristics from Rocha et al. [19] and expert feedback. Key metrics include reasoning time, transformation accuracy, and explanation clarity, ensuring a

thorough and repeatable assessment of scalability and user experience.

6 RELATED WORK

This project builds on the RC algorithm, introduced by Lehmann and Magidor as an extension of the KLM framework for nonmonotonic reasoning with ranked models [14], rooted in Reiter's default logic [18]. Casini et al. developed efficient syntactic methods using base ranks [5], while Giordano et al. proposed structural optimisations [9], and Kaliski clarified workflows for implementation [11]. The knowledge base generation has evolved through Lang's multithreaded generation of defeasible implications [13], and Sadiki's use of a KB database for efficient reuse [21], building on Bailey's distribution-based population methods [1]. Statements are generated pseudo-randomly with configurable ratios of classical to defeasible rules, forming structured rankings. On the UI side, tools like TweetyProject [25], OntoDebug [22], and SILK [10] offer limited support for defeasible logic and explanation. Coetzer et al. developed a debugger for RC with step-based tracing but without natural language support [6]. To address this, Wyner and Governatori [26] and Rudinger et al. [20] emphasised the need for systems that integrate natural language processing with defeasible reasoning to better capture exceptions and human-like reasoning styles.

7 ETHICAL, PROFESSIONAL AND LEGAL ISSUES

This project does not involve the collection of personal or sensitive user data, and thus avoids many privacy-related ethical concerns. However, it is important to ensure that the system's reasoning outputs and explanations are transparent, interpretable, and not misleading. Users must be made aware of the limitations of DR, particularly in ambiguous or conflicting scenarios.

All referenced work will be properly cited to maintain academic integrity. The project will also adhere to professional software development standards, including proper documentation, version control, and testing.

If external libraries (e.g. TweetyProject) or reasoning datasets are used, licencing terms will be reviewed to ensure compliance with open-source or academic use policies.

We will acknowledge the limitations of LLMs, making sure that users are aware that the output produced by our KBG is purely generative and may contain plausible sounding, but incorrect information (hallucinations). The output should not be taken as true in the real world and must not be used for advice.

8 ANTICIPATED OUTCOMES

The project is anticipated to yield a scalable, efficient, and user-friendly DR system that advances both performance and accessibility. The proposed optimisations: parallelised entailment checking, SAT sub-query memoization, and adaptive hybrid rank traversal, are expected to significantly improve execution time, reduce redundant SAT solver calls, and enhance query processing efficiency, particularly for large or structurally complex KBs, all while preserving correctness and system simplicity. In parallel, improvements to the KB generation process will allow users to view logic using natural languages rather than arbitrary symbols, supported by a simplified UI requiring simplified inputs, while maintaining generation integrity. The system will also function as a DR debugger, offering multi-modal explanations that clarify the RC process and lowers the barrier to understanding complex reasoning. Robust benchmarking and empirical validation will further ensure the tool's reliability, laying a foundation for future research and development.

8.1 Expected Impact

This project is expected to advance the field of nonmonotonic reasoning by delivering a more efficient and accessible implementation of the RC algorithm. It introduces three key optimisation techniques—parallelised entailment checking, SAT sub-query memoization, and adaptive hybrid rank traversal. Each of these techniques are expected to improve runtime performance while preserving algorithmic correctness, contributing to the scalability of reasoning systems in practical applications. These strategies may also inform enhancements to other rank-based or nonmonotonic inference methods. Beyond efficiency, the project is expected to enhance usability by bridging symbolic KBG with human interpretable semantics through topic aware labels and a more intuitive UI. It further integrates natural language transformation, enabling users unfamiliar with formal logic to construct and query KBs using everyday English. The inclusion of multi-modal explanations is expected to improve transparency and interpretability. While scalability optimisations are expected to extend the feasibility of reasoning over complex, real-world domains such as legal reasoning, policy management, and knowledge engineering. By building on prior work in natural language reasoning [20] and regulatory rule translation [26], the project will provide a foundation for future work combining formal reasoning techniques with a natural language.

8.2 Key Success Factors

Optimisation:
The success of this part of the project will depend on several key factors. Firstly, a correct and modular implementation of the RC algorithm is essential to ensure logical soundness and alignment with established theoretical principles. Secondly, the integration of the proposed optimisation techniques (parallelised entailment checking, SAT sub-query memoization, and adaptive hybrid rank traversal) must be implemented correctly to achieve the most accurate results. Regular supervisor meetings and agile development planning will be critical in ensuring that progress remains consistent and that any technical or conceptual challenges are addressed early. Additionally, comprehensive testing using a variety of KBs and query structures will be necessary to validate both correctness and efficiency. Lastly, clean code structure and well-maintained documentation will enhance the long-term utility of the project as a resource for learning, future development, and academic reference.
KBG:
The success of this section depends on multiple factors. Primarily success is determined by the correct allocation of phrases to atoms so that the knowledge base forms coherent statements. Additionally, effective integration with a LLM (using an API), which effectively

filters statements should be created. A usable and understandable KB should be generated and shown to the user. This can be presented with results and metrics such as algorithm run time.

UI/Debugger:

The success of this project will depend on several factors. Starting with the accuracy of natural language processing in translating natural language input into DR rules. The system's ability to provide effective explanations (graphical, textual, and logical) will be crucial for user understanding. Scalability will be a key factor, with the system needing to efficiently handle large KBs. The usability of the UI will determine how easily users can interact with the system and interpret reasoning steps. Lastly, performance improvements, measured through benchmarking, and the integration with existing frameworks such as TweetyProject, will also be critical for the system's success and wider adoption.

9 PROJECT PLAN

9.1 Risks and Contingencies

Refer to APPENDIX A - RISKS

9.2 Resources Required

9.2.1 Software. The programming language chosen would be Java, and React Native/ JavaScript and HTML executed through the VS Code IDE. We will leverage existing libraries and benchmarking/profiling tools. Git will be used for version control.

9.2.2 Hardware. Standard laptops and desktops are powerful enough to run our algorithms.

9.3 Milestones and Tasks

Refer to APPENDIX B - MILESTONES

9.4 Timeline

Refer to APPENDIX C - TIMELINE

9.5 Work Allocation

Our project is split into three sections, KB generation, RC and a UI. We will each take one of these parts and be the primary architect for these. Since these sections are all integrated, we will work together on overlapping problems so that we can integrate our work by the end of the project.

Student	Work Allocation
Jethro Dunn	Implementation of a KBG which uses both symbols and phrases to express logical statements. Connecting an API to check if statements are coherent. Optimisation of phrase storage, implementation and API querying. Testing the extended KBG with unit tests and comparative testing between the new extended KBG and the original.
Julia Cotterrell	The implementation of an intuitive UI. The interface will improve on communicating the steps and conclusions drawn through entailment. It will further work on the optimisation of the debugging process. This will be testing through user testing and unit testing to ensure reliability and performance optimisation.
Nevaniah Gounden	The implementation of both a baseline and an optimised RC algorithm. The optimised version incorporating parallelised entailment checking, SAT sub-query memoization, and adaptive hybrid rank traversal. This includes extensive testing across diverse KBs, comparative performance analysis between the baseline and optimised versions, and documentation of empirical findings and implementation strategies.

Table 1: Work distribution

9.6 Deliverables

Refer to APPENDIX D - PROJECT DELIVERABLES

REFERENCES

[1] Bailey, A. Scalable defeasible reasoning, 2021. honors paper.

[2] Ben-Ari, M. *Mathematical logic for computer science.* Springer Science & Business Media, 2012.

[3] Bryant, D., and Krause, P. A review of current defeasible reasoning implementations. *The Knowledge Engineering Review 23*, 3 (2008), 227–260.

[4] Casini, G., Meyer, T., and Varzinczak, I. Defeasible entailment: From rational closure to lexicographic closure and beyond. In *Proceeding of the 17th International Workshop on Non-Monotonic Reasoning (NMR 2018)* (2018), pp. 109–118.

[5] Casini, G., Meyer, T., and Varzinczak, I. Taking defeasible entailment beyond rational closure. In *Logics in Artificial Intelligence: 16th European Conference, JELIA 2019, Rende, Italy, May 7–11, 2019, Proceedings 16* (2019), Springer, pp. 182–197.

[6] Coetzer, S., and Britz, K. Debugging classical ontologies using defeasible reasoning tools. In *Formal Ontology in Information Systems (FOIS 2022)* (2022), vol. 344, IOS Press, pp. 97–106.

[7] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms.* MIT Press, 2009.

[8] Dovier, A., Formisano, A., Gupta, G., Hermenegildo, M., Pontelli, E., and Rocha, R. Parallel logic programming: A sequel. *ACM Computing Surveys (CSUR) 55*, 1 (2022), 1–39.

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

[10] Grosof, B., Burstein, M., Dean, M., Andersen, C., Benyo, B., Ferguson, W., Inclezan, D., and Shapiro, R. A silk graphical ui for defeasible reasoning, with a biology causal process example. In *9th International Semantic Web Conference ISWC 2010* (2010), Citeseer, p. 113.

[11] Kaliski, A. An overview of klm-style defeasible entailment. *NA* (2020).

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

[13] Lang, A. Extending defeasible reasoning beyond rational closure, 2023. honors paper.

[14] Lehmann, D. Another perspective on default reasoning. *Annals of mathematics and artificial intelligence 15* (1995), 61–82.
[15] Lehmann, D., and Magidor, M. What does a conditional knowledge base entail? *Artificial intelligence 55*, 1 (1992), 1–60.
[16] Levesque, H. J. Knowledge representation and reasoning. *Annual review of computer science 1*, 1 (1986), 255–287.
[17] Moisan, S. Generating knowledge-based system generators. *Insights into Advancements in Intelligent Information Technologies: Discoveries: Discoveries* (2012), 1.
[18] Reiter, R. A logic for default reasoning. *Artificial intelligence 13*, 1-2 (1980), 81–132.
[19] Rocha, L. C., Andrade, R. M., Sampaio, A. L., and Lelli, V. Heuristics to evaluate the usability of ubiquitous systems. In *Distributed, Ambient and Pervasive Interactions: 5th International Conference, DAPI 2017, Held as Part of HCI International 2017, Vancouver, BC, Canada, July 9–14, 2017, Proceedings 5* (2017), Springer, pp. 120–141.
[20] Rudinger, R., Shwartz, V., Hwang, J. D., Bhagavatula, C., Forbes, M., Le Bras, R., Smith, N. A., and Choi, Y. Thinking like a skeptic: Defeasible inference in natural language. In *Findings of the Association for Computational Linguistics: EMNLP 2020* (2020), pp. 4661–4675.
[21] Sadiki, M. A study of parameters and optimization strategies in defeasible knowledge base generation, 2024. honors project.
[22] Schekotihin, K., Rodler, P., and Schmid, W. Ontodebug: Interactive ontology debugging plug-in for protégé. In *Foundations of Information and Knowledge Systems: 10th International Symposium, FoIKS 2018, Budapest, Hungary, May 14–18, 2018, Proceedings 10* (2018), Springer, pp. 340–359.
[23] Shoham, Y. A semantical approach to nonmonotonic logics. In *Readings in Non-Monotonic Reasoning*, M. L. Ginsberg, Ed. Morgan Kaufmann, 1987, pp. 227–249.
[24] Shtrichman, O. Pruning techniques for the sat-based bounded model checking problem. In *Correct Hardware Design and Verification Methods: Proceedings of CHARME 2001, Livingstone, UK, September 2001* (2001), vol. 2144 of *Lecture Notes in Computer Science*, Springer, pp. 58–70.
[25] Thimm, M. The tweety library collection for logical aspects of artificial intelligence and knowledge representation. *KI-Künstliche Intelligenz 31*, 1 (2017), 93–97.
[26] Wyner, A. Z., and Governatori, G. A study on translating regulatory rules from natural language to defeasible logics. In *RuleML (2)* (2013), Citeseer.

APPENDIX A – RISKS

ID	Risk	Probability	Impact	Mitigation	Monitoring	Management
1	Underestimating complexity	High	High	Break down into smaller tasks and consult supervisor regularly	Weekly progress reviews	Adjust timelines and prioritise critical components
2	Skills Gap	High	High	Allocate tasks based on strengths and use teamwork to bridge knowledge gaps	Track task completion and make help requests early	Reassign/Simplify tasks, contact supervisor.
3	Falling behind schedule	High	High	Break work down into weekly subsections and rely on teamwork to ensure everyone's part gets done	Report progress during weekly meetings	Re-scope project and ensure essential features are implemented
4	Team coordination issues (poor communication)	Low	High	Communicate regularly and have a shared calendar	Look for inconsistent updates/duplicated work and follow-up immediately	Regular communication, have clear roles and documentation
5	Conflicting priorities	High	Medium	Schedule buffer days and keep due dates fixed, but schedules flexible	Keep other members updated on days when we're unavailable	Adjust timelines or workload distributions accordingly
6	Tooling or environment issues (e.g., Java compatibility)	Low	Medium	Use stable development environment and version control	Monitor system and library dependencies	Document setup and use virtual environments
7	Computer failure, resulting in loss of work	Low	High	Use Git and cloud storage (e.g., GitHub, Google Drive); autosave features	Check for recent commits/uploads from all members	Recover from backup; reassign work if recovery fails
8	A team member is unable to complete their task	Medium	Low	Adopt a buddy system to hold each other accountable	Flag blockers as soon as possible	Reallocate tasks
9	Supervisor unavailable	Very Low	Medium	Ask for a backup contact person, also plan questions ahead of time	Check supervisors availability ahead of time	Make decisions as a team and document them until supervisor is available
10	Insufficient testing	Medium	High	Write unit tests early; include testing in task planning; peer reviews	Track test coverage and bugs found late	Allocate time for testing at each stage; use test checklists

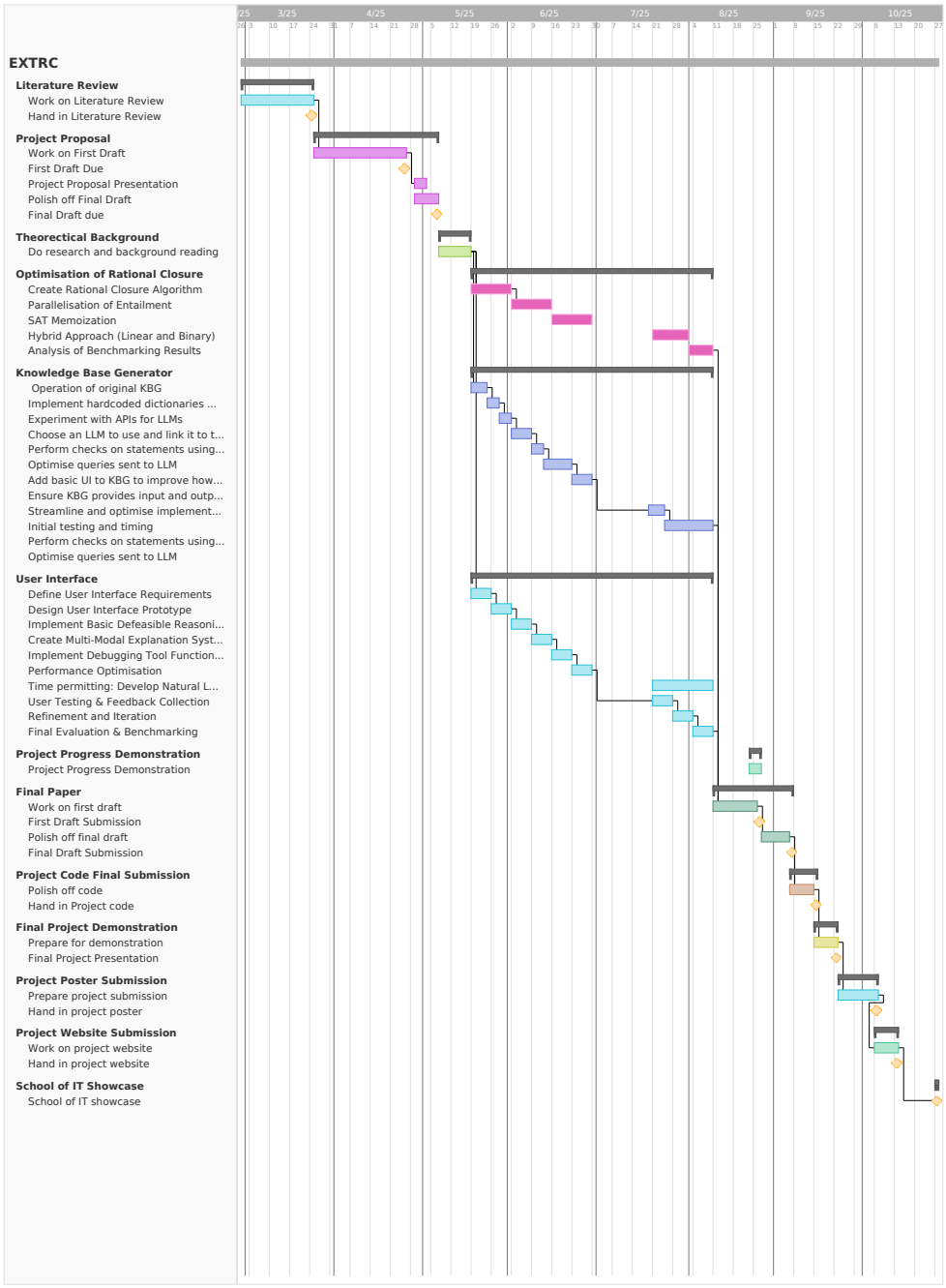
Table 2: Risk Management Table

APPENDIX B – MILESTONES

Tasks	Start Date	End Date
Literature Review	03/03	24/03
Project Proposal	25/03	06/05
First Draft	25/03	24/04
Project Proposal Presentation	29/04	02/05
Final Draft	02/05	06/05
Theoretical Background	06/05	18/05
Optimisation of Rational Closure	07/05	29/07
Create Rational Closure Algorithm	14/05	22/05
Parallelisation of Entailment	23/05	30/05
SAT Memoization	31/05	07/06
Hybrid Approach (Linear and Binary)	08/06	15/06
Analysis of Benchmarking Results	16/06	22/06
Knowledge Base Generator		
Operation of original KBG	07/05	13/05
Implement hardcoded dictionaries mapping to atoms	13/05	20/05
Experiment with APIs for LLMs	20/05	23/05
Choose an LLM to use and link it to the KBG	23/05	01/06
Perform checks on statements using LLM	01/06	04/06
Optimise queries sent to LLM	04/06	14/06
Add basic UI to KBG to improve how parameters are inputted	14/06	24/06
Ensure KBG provides input and output integrating with team members programmes	24/06	05/07
Streamline and optimise implementation and refactoring	05/07	10/07
Initial testing and timing	10/07	28/07
User Interface	14/05	29/07
Define User Interface Requirements	14/05	20/05
Design User Interface Prototype	21/05	27/05
Implement Basic Defeasible Reasoning Integration	28/05	05/06
Develop Natural Language Processing (NLP) Component	06/06	12/06
Create Multi-Modal Explanation System	13/06	19/06
Implement Debugging Tool Functionality	20/06	26/06
Performance Optimisation	27/06	03/07
User Testing & Feedback Collection	04/07	10/07
Refinement and Iteration	11/07	17/07
Final Evaluation & Benchmarking	18/07	24/07
Project Progress Demonstration	28/07	01/08
Final Paper		
First Draft Submission		26/08
Final Draft Submission		05/09
Project Code Final Submission		15/09
Final Project Demonstration	22/09	26/09
Project Poster Submission		06/10
Project Website Submission		13/10
School of IT Showcase		27/10

Table 3: Milestones Table

APPENDIX C: TIMELINE



APPENDIX D - PROJECT DELIVERABLES

#	Deliverable	Date(s)
1	Literature Review Due	Mon 24-Mar
2	Project Proposal presentations	Friday 25 April
3	Project Proposal due	Tue 6-May
4	Ethics applications deadline	Mon 12-May
5	Project Progress Demonstration	Mon 28-Jul to Fri 1-Aug
6	Complete Draft of final paper due (if not done -10%)	Tue 26-Aug
7	Project Paper Final Submission	Fri 5-Sep
8	Project Code Final Submission	Mon 15-Sep
9	Final Project Demonstration	Mon 22-Sep to Fri 26-Sep
10	Poster Due	Mon 6-Oct
11	Website Due	Mon 13-Oct
12	School of IT Showcase	Mon 27-Oct (TBC)

Table 4: Schedule of project deliverables