Defeasible Conditionals in Answer Set Programming Research Proposal

Racquel Dennison
dnnrac003@myuct.ac.za
University of Cape Town
Cape Town, Western Cape, South Africa

Jack Mabotja
mbtjac003@myuct.ac.za
University of Cape Town
Cape Town, Western Cape, South Africa

Sibusiso Buthelezi
bthsib016@myuct.ac.za
University of Cape Town
Cape Town, Western Cape, South Africa

ABSTRACT

Knowledge representation and reasoning (KRR) is a subfield of artificial intelligence (AI) whereby a system has some information about the world and can reason with this information. Defeasible reasoning is a form of reasoning that enables a system to infer conclusions from statements that are incomplete or contradictory. Entailment checking is the process of determining whether or not a statement can be concluded from a knowledge base. Implementations of defeasible entailment have been computed using imperative languages. Our project aims to explore the possible advantages of using a declarative language to compute defeasible entailment.

CCS CONCEPTS

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

KEYWORDS

Artificial Intelligence, Knowledge Representation and Reasoning, Defeasible Reasoning, Rational Closure, Answer Set Programming

1 INTRODUCTION

Artificial intelligence (AI) consists of many subfields, one of which is knowledge representation and reasoning (KRR) [16]. KRR is concerned with how knowledge can be expressed symbolically and manipulated in an automated way by reasoning services [11].

Classical propositional logic has traditionally been employed for reasoning and knowledge representation and has formed the foundation for more complex logic [11]. However, while it offers desirable properties, it inadequately models human reasoning, especially when dealing with contradictory or incomplete information.

One solution to this is *defeasible reasoning*, a pattern encompassing common sense reasoning. This allows for revising previously drawn conclusions in light of new information. A prominent approach to *defeasible reasoning* is the KLM framework defined by Kraus, Lehmann, and Magidor [12]. The KLM framework defined a set of properties that defeasible entailment relations should follow. *Rational closure* is a form of *defeasible entailment* that follows the properties outlined in the KLM framework. Computing *rational closure* is an NP-complete problem [11]. Efforts to enhance the efficiency of computing *rational closure* have primarily relied on imperative approaches, with no exploration of the declarative paradigm. Declarative programming languages hold promise in providing advantages for computing and allowing for easier modification of program constraints without overhauling the entire algorithm [14].

Our project aims to investigate the utility of a declarative programming language in modelling defeasible entailment.

2 BACKGROUND

2.1 Propositional logic

Propositional logic [1] is a logic framework for modelling information about the world. A finite set $\mathcal{P} = \{p, r, q, ...\}$ of *propositional atoms* represents the fundamental statements that can be assigned a value of true or false. *Boolean operators* ($\rightarrow, \leftrightarrow, \wedge, \vee, \neg$) are used recursively to define more complex formulas. Formulas allow us to represent information about the world. The set of all formulas over the set $\mathcal{P}$ is the *propositional language*, $\mathcal{L}$ [1]. Truth values are assigned to formulas using an *interpretation*, which is a function $\mathcal{I}: \mathcal{P} \rightarrow \{T, F\}$. The set $\mathcal{U}$ contains all possible interpretations. If a given formula α has a truth value of true under a given interpretation $\mathcal{I}$, then $\mathcal{I}$ is said to satisfy α (denoted as $\mathcal{I} \Vdash \alpha$) [1]. For a given interpretation $\mathcal{I}$ to satisfy a knowledge base $\mathcal{K}$, which is a finite set of propositional formulas, $\mathcal{I}$ should satisfy every formula in the knowledge base. The set of all interpretations that satisfy a knowledge base is mod($\mathcal{K}$) [11].

2.2 Entailment

Entailment forms the foundations of a reasoning system. Entailment allows one to determine if a given statement follows from a knowledge base. A statement α is entailed by a knowledge base $\mathcal{K}$ (denoted as $\mathcal{K} \models \alpha$) whenever $mod(\mathcal{K}) \subseteq mod(\alpha)$ holds [11]. For example, given the following $\mathcal{K} = \{bird \rightarrow fly, penguin \rightarrow bird\}$ and $\alpha = penguins \rightarrow fly$. From the definition of entailment, we see that $\mathcal{K} \models \alpha$.

2.3 The KLM Framework and Extensions

A shortfall of classical propositional logic is its inability to handle exceptionalities. This leads to difficulties when representing exceptional knowledge. To understand this better, consider the following.

Example 2.1. Suppose $\mathcal{K}_1 := \{birds \rightarrow fly, penguins \rightarrow birds, penguins \rightarrow \neg fly\}$. The statement $penguins \rightarrow \neg fly$ indicates that penguins are exceptional types of birds. Classical reasoning infers that penguins fly and penguins do not fly, so by its definition of entailment, the conclusion drawn is that penguins do not exist.

Example 2.1 highlights the limitations of *classical propositional logic*. A solution to this is *defeasible reasoning*, which is a form

of reasoning that captures the idea of uncertainty and allows for the retraction of statements when new information contradicts them [11]. Many approaches to *defeasible reasoning* have been explored, however, the approach defined by Kraus, Lehmann and Magidor (KLM) [12] has been predominately studied in the field. KLM defined a set of properties that defeasible entailment relations should adhere to. *Rational closure*, a conservative form of reasoning, adheres to these defined properties. This paper will outline our approach to modelling this entailment relation, however, we should first consider KLM as a reasoning framework.

2.3.1 Reasoning within the KLM framework. KLM extended the language $\mathcal{L}$ by introducing a *defeasible conditional*, $\mid\sim$. Statements of the form $\alpha \mid\sim \beta$ are read as α *typically implies* β. This allowed for defeasible knowledge bases to express statements that typically hold. Defeasible entailment is denoted as $\mid\approx$. Propositional statements can be encoded as defeasible statements. For example, given α, we can represent it as $\neg\alpha \mid\sim \bot$ ($\bot$ simply means false). This statement simply states that where we consider α, $\neg\alpha$ is typically false.

2.4 Rational Closure

Rational closure is a form of defeasible reasoning that was first defined by Lehmann and Magidor [3]. In many ways, it is the most simple and intuitive way of defining defeasible entailment. *Rational closure* can be defined semantically and algorithmically. In this project, we will be focusing on the latter.

We require all statements to be encoded in the form $\alpha \mid\sim \beta$ where $\alpha, \beta \in \mathcal{L}$.

The materialisation of a knowledge base $\mathcal{K}$ is defined as $\overrightarrow{\mathcal{K}} := \{\alpha \rightarrow \beta \mid \alpha \mid\sim \beta \in \mathcal{K}\}$. A classical formula $\alpha \in \mathcal{L}$ is said to be exceptional in a knowledge base $\mathcal{K}$ if $\overrightarrow{\mathcal{K}} \models \neg\alpha$ [11]. In the first step to computing *rational closure*, we will rank the formulas of a knowledge base based on their exceptionality. We define a sequence of materialisations $E_0, E_1, .., E_{n-1}, E_\infty$, where $E_0 = \overrightarrow{\mathcal{K}}$ and each subsequent $E_i = \{\alpha \rightarrow \beta \in E_{i-1} \mid E_{i-1} \models \neg\alpha\}$. Each E_i consists of statements $\alpha \rightarrow \beta$ such that α is exceptional. The sequence of materialised formulas terminates when $E_{i-1} == E_i$. Each sequence E is used to create a ranking of the statements in $\mathcal{K}$ which is obtained by setting $\mathcal{K}_i = E_i \setminus E_{i+1}$ for $0 \leq i \leq n-1$ and $\mathcal{K}_\infty == E_\infty$.

Example 2.2. Let $\mathcal{K} := \{b \mid\sim f, b \mid\sim w, \neg(p \rightarrow b) \mid\sim \bot, p \mid\sim \neg f\}$.

The associated ranking of the formulas is given in Table 1. All propositional statements will have a rank of ∞.

$\mathcal{R}_\infty$	$p \rightarrow b$
$\mathcal{R}_1$	$p \rightarrow \neg f$
$\mathcal{R}_0$	$b \rightarrow f, b \rightarrow w$

Table 1: Ranking $\mathcal{K}_0, K_1, ..., \mathcal{K}_\infty$ from 2.2

To compute if a defeasible knowledge base $\mathcal{K} \mid\approx \alpha \mid\sim \beta$, we will start by checking if the antecedent, α, is exceptional in the knowledge base. If α is exceptional in $\mathcal{K}$, we will repeat the process with $\mathcal{K} = \mathcal{K} \setminus \mathcal{K}_0$. This will continue until we have gone through all the ranks. If α is not exceptional in $\mathcal{K}$, then $\mid\approx$ is computed as $\overrightarrow{\mathcal{K}} \models \alpha \rightarrow \beta$.

Example 2.3. If we consider the query $p \mid\sim \neg f$ and the same knowledge base computed in Example 2.2. The antecedent p is exceptional for $\mathcal{K}$, thus $\mathcal{K} := \{\alpha \mid\sim \beta \mid \alpha \mid\sim \beta \notin \mathcal{K}_0\}$. Checking entailment again, we see that p is not exceptional in $\mathcal{K}$ and $\overrightarrow{\mathcal{K}} \models p \rightarrow \neg f$.

The above example has shown that defeasible entailment holds in rational closure, contrasted with classical propositional logic which would conclude that penguins do not exist.

2.5 Answer Set Programming

There are many ways in which computational problems can be encoded, two of which are declarative and imperative programming. An imperative approach involves defining a sequential set of steps of what needs to happen to solve a problem. Declarative approaches focus on describing the desired outcomes of a problem instead of telling a computer what steps to take. The disparity between imperative and declarative programming can be grasped by comparing imperative and declarative sentences in English. For instance, an imperative sentence issues a command that can be followed or ignored, such as "wash the dishes", while a declarative sentence merely states a fact, something that can either be true or false, like "it is raining." Answer set programming (ASP) is a declarative approach to solving computational problems [14]. Over the years, ASP has become particularly popular in modelling hard computational problems [2]. It has been used heavily in the field of logic and knowledge representation. The language allows for incomplete knowledge to be expressed and reasoned with. We are particularly interested in answering whether ASP is useful as an environment in which defeasible entailment can be computed.

3 PROJECT DESCRIPTION

3.1 Overview of the Problem

The work done in studying the KLM framework for *defeasible reasoning* has been largely theoretical. Defeasible entailment algorithms such as *rational closure* have been defined and studied for years, however, minimal work has been done in implementing and studying them practically. In SCADR [9] and SCADRv2 [15], *rational closure* was implemented and optimised using imperative programming languages. In this project, we aim to assess whether the ASP environment offers alternative advantages compared to imperative languages concerning the computation of defeasible conditionals. The focus will be on the language's capabilities in solving defeasible entailment and generating knowledge bases. To achieve this, the goal is to implement rational closure and a knowledge base generator using ASP. Additionally, we intend to develop a debugger capable of explaining the rational entailment process.

3.2 Motivation

Computing rational closure is an NP-complete problem, and while efforts have been made to optimise its efficiency, not many have explored declarative environments. ASP is a highly expressive language that provides an intuitive and natural way to represent problems [2]. Additionally, ASP is easy to use and very flexible [14], allowing for modifications and refinements to programs [2]. Due to these properties, ASP implementations of *rational closure* and

defeasible knowledge base generators may have the potential to be more computationally efficient in terms of runtime, allow for more flexibility when adding new features, and be more compact, which could reduce bad code smells.

4 RESEARCH QUESTIONS AND AIMS

4.1 Research Questions

- How does modelling defeasible conditionals using ASP, compare against an implementation using an imperative language, in terms of compactness and flexibility?
- How does the performance of an ASP environment compare with that of an imperative environment in terms of runtime when modelling defeasible conditionals?

4.2 Aims

The key aims of this project are:

- To develop a reasoner tool, using ASP, which reasons about defeasible knowledge bases through rational closure.
- To improve the runtime of our baseline implementation of rational closure by exploring different approaches to computation using ASP.
- To implement a debugger that will provide detailed explanations of the output of rational closure.
- To develop a knowledge base generator using ASP.
- To determine how to express defeasible conditionals as a set of ASP rules.

5 METHODS AND PROCEDURES

The work for this project is split into four components. The first will be a baseline implementation of rational closure using ASP. This will be collaborated on by the entire team. The three remaining components will be split between each team member. This includes optimisations on our initial implementation, a knowledge base generator and a debugger.

5.1 Naive rational closure

5.1.1 Implementation. Before we can extend the rational closure algorithm to ASP, we will first gain an in-depth knowledge of how rational closure functions. This will also include an in-depth understanding of the ASP language. We will do this by reading and understanding the relevant literature related to these topics. Following this, we will define a potential set of ASP rules capable of effectively modelling rational closure. To assess the efficacy of our defined rules, we will engage in discussions with our supervisors, presenting our work for their evaluation. Additionally, we will provide justifications for why we believe these rules are suitable for computing rational closure. If the outlined rules do not perform rational entailment as desired, we will address any identified issues and present our revised work to our supervisors for further review. We will proceed with our implementation after finalising a satisfactory set of rules that accurately model rational closure. Our implementation will utilise Clasp [4] as the solver and Gringo [6] as the grounder.

5.1.2 Testing and analysis. Validation of the implementation will involve two sets of tests. Initially, rational closure will be manually computed for a predetermined knowledge base and query. Subsequently, the ASP implementation will compute rational closure for the same input. A comparison between the manual and ASP-derived results will be conducted to assess the accuracy of the implementation. Additionally, a comparative analysis will be performed between our implementation and SCADR [10]. Both implementations will be given an identical knowledge base and query inputs. This comparative study aims to validate the correctness of our program by evaluating its output against a known implementation.

5.2 Optimisations

5.2.1 Implementation. The initial steps towards devising optimisation strategies for our ASP implementation will start once we have finalised our implementation strategy for the original version of our reasoner. This process will entail an examination of the imperative optimisations of rational closure carried out in SCADR [9] and SCADRv2 [15], to ascertain their potential applicability in a declarative programming paradigm. Additionally, we will conduct background research into ASP problem-solving methodologies to identify ASP constructs that can be leveraged to enhance our base implementation. The collaboration between our intended ASP implementation and the mechanics and structures of ASP will guide our optimisation endeavours.

Subsequently, we will formulate potential optimisation strategies based on previous optimisations, the inherent properties of the rational closure algorithm, and ASP's problem-solving approach. We will then assess the relevance of these strategies to our specific problem, discarding those that are not applicable. For the viable strategies, we will explore how they can complement each other when combined. Finally, we will outline an implementation approach and proceed with its execution.

5.2.2 Testing and analysis. The test for correctness will follow the same procedure as the naive implementation. For our analysis, we will begin by developing a benchmarking tool in Java. This tool will have the capability to load various rational closure implementations, execute identical entailment queries on them, and measure the time taken by each implementation to produce a response. The tool will utilise knowledge bases generated by a knowledge base generator from SCADR [9].

Once both the benchmarking tool and our ASP implementation are completed, we will benchmark our basic ASP rational closure implementation against the basic implementation from SCADR [9]. Following the completion of optimisations, the optimised versions will be benchmarked against the original version and the imperative implementations done in SCADR [9] and SCADRv2 [15], including their optimisations. Subsequently, the results of the benchmarking process will be analysed to determine if ASP provides any computational efficiency advantages in terms of runtime. Throughout this process, we will maintain continuous consultation with our supervisor.

5.3 Knowledge base generator

5.3.1 Implementation. To gain an understanding of how knowledge bases are generated, their fundamental makeup will be studied. This will consist of looking at the structure of classical statements in a knowledge base. Once this understanding is grasped, we will then move on to how to design our knowledge generator so that it allows for the generation of defeasible statements. We will focus on understanding the KLM properties [12] as these play an important role in defeasible knowledge bases. The initial implementation phase will consist of designing a set of rules that will generate knowledge bases which will consist of simple statements. There will be no limit on the amount of ranks and the knowledge base itself will take on a specified size. This will be our baseline. Once we have consulted our supervisors on our design and are satisfied, we will start with the implementation in ASP. Our implementation will undergo testing by analysing the output of the knowledge generator and ensuring that it is computing what is defined in the BaseRank algorithm. Thereafter, we will extend the baseline implementation. We will be adding additional parameters such as the number of ranks a ranked knowledge base should have and the distribution of formulas over the ranks.

5.3.2 Testing and analysis. The testing phase will primarily focus on verifying the correctness of the knowledge base generator implementation. This verification process will involve comparing the output of our implementation against that of a known benchmark, the SCADR implementation [10]. Subsequently, the analysis phase will assess the flexibility and compactness of the language. Flexibility will be evaluated based on the ease of adding additional features to the implementation compared to traditional imperative languages. This assessment will consider factors such as the complexity of integrating new functionalities into the existing codebase. Furthermore, computational efficiency in terms of clock time will undergo evaluation. This will involve comparing the computing time of the ASP knowledge base against that of the SCADR implementation [10]. Each iteration of testing will entail varying changes to parameters in the knowledge base distribution, facilitating a comprehensive analysis.

5.4 Debugger

5.4.1 Implementation. The implementation of the Rational Closure Debugger (RCD) will be developed in 4 iteration cycles:

- **Phase 0** - User Requirements: Here, the functionalities and features users of the debugger prioritise will be identified and gathered. Unified Modelling Language (UML) diagrams and design documents will be created and assessed for the proposed system.
- **Phase 1** - Graphical User Interface: This phase will include developing the graphical user interface base of the RCD, offering functionality to input queries and visualisation properties that represent inputs and outputs, knowledge bases, ranked knowledge bases, and a summary of the inference steps taken to evaluate the rational closure of the query. This will be developed to allow users to interface with our declarative implementation of the rational closure reasoner
- **Phase 2** - Step-by-step Execution feature: This phase will focus on developing the functionality to execute and visualise the computation of rational closure one step at a time, controlled by the user. This feature will allow users to navigate through ration closure computations with finer granularity.
- **Phase 3** - Explanations and Justifications: This phase will include developing functionality for the debugger to provide explanations and justifications for each inferred consequence or decision made during the computation of rational closure, helping users understand the reasoning behind the steps performed.

Each phase of development of the RCD will follow an iterative approach, allowing for consistent feedback and improvement to the system. The software, frameworks and architecture to develop the RCD will be determined after a more solid foundation of ASP has been formed, and what software and frameworks compatible with our implementation of rational closure have been accessed.

5.4.2 Testing and analysis.

- **Unit Testing**: Unit tests will be conducted throughout the development of the RCD to ensure its robustness and correctness as an explainer tool. The unit tests will ensure that the steps taken to arrive at a rational closure answer need to have the appropriate explanations for their execution. .
- **User Acceptance Testing**: The usability and usefulness of the RCD will be evaluated by conducting user acceptance tests after all new, major additions or modifications to the tool, with one final user acceptance test phase at the end of the tool's development for final feedback and modifications. The usefulness will be evaluated by how intuitive the users find the tool, and whether it performs as expected. Our supervisors will be the participants testing the RCD and their feedback will inform changes required and improvements for the tool.

6 ETHICAL, PROFESSIONAL AND LEGAL ISSUES

The software developed is not intended to be used by the general public, but only by experts within the field of KRR. Thus, we will only test our software tools with our supervisors.

Our project will use existing imperative implementations of rational closure as a benchmark against which we will measure our declarative implementation's performance. This software was created by previous University of Cape Town (UCT) honours students, which makes it the Intellectual Property of UCT. The authors of the software will be acknowledged and their work will be cited.

We will use Clingo for our ASP implementation of rational closure, which is part of the Potassco suite. The Potassco suite is a set of ASP tools that were developed at the University of Potsdam. Clingo is distributed under the MIT License[1], which means it is free for use.

[1] https://opensource.org/license/mit

7 RELATED WORK

7.1 Imperative Implementations of Rational Closure

7.1.1 SCADR. In SCADR [10], four defeasible reasoners were developed using Java. The first one implemented the rational closure as it was defined in the KLM framework [3], while the other three reasoners were optimisations of the original implementation. The first of the three used binary search to find the rank at which the antecedent of a query statement becomes consistent with the knowledge base when performing entailment checking (which will be referred to as the 'rank of consistency' for the rest of this section). The second stored ranks of consistency in a hash table. Finally, the third combined these two implementations. Replacing linear search with binary search improved the performance when the rank of consistency was large. Storing the ranks of consistency improved the performance for query sets where most of the antecedents were the same as the computation of the rank of consistency only needed to be done once for all statements of the same antecedent.

7.1.2 SCADRV2. SCADRv2 [15] extends SCADR [10] by adding further optimisations to the basic implementation. They optimise further by initially using a ternary search to compute the rank of consistency. The second optimisation included saving the computed ranks of consistency for a given antecedent and using parallelisation to perform entailment checks. This allows for the processing of multiple entailment queries at once. The ternary search approach only performed better than the binary search approach. The parallelisation technique improves performance by leveraging multiple cores on modern processors.

7.2 Classical SAT solvers

At its core, *rational closure* uses SAT Solvers when computing entailment on a particular rank level. SAT solvers compute the Boolean Satisfiability Problem, which is the problem of determining the existence of an interpretation that satisfies a given formula [7]. An example SAT solver is the DPLL algorithm [7], which solves the SAT problem by incrementally constructing an interpretation which satisfies a given formula.

7.3 Answer Set Solver

7.3.1 Clingo. This project will use the ASP system, *clingo*, to implement rational closure. *Clingo* is a combination of a grounder, *gringo* and a solver, *clasp*. *Clingo* takes an input program and passes it to *gringo*, which generates a propositional representation of the input program [5]. Gringo's output is then passed to *clingo* to compute the stable models of the propositional program which correspond to the solutions of the input program.

7.4 Knowledge Base Generator

7.4.1 KBGT. In SCADR [10], Bailey developed a defeasible knowledge base generator with parameterised processes. These parameters include the total number of defeasible ranks, the total number of defeasible statements, and the distribution of defeasible statements over the defeasible ranks among others. This implementation used defeasible clash graphs (DCG) to generate defeasible knowledge bases. The tool allows for five possible distributions of statements over the ranks when generating a knowledge base.

7.4.2 KBGenerator. In EXTDR [13], Lang created a non-deterministic defeasible knowledge base generator that can generate knowledge bases with different configurations. The approach taken to implement this knowledge base is different from the one used in SCADR [10]. Instead of using sophisticated data structures, Lang generates knowledge bases by first building defeasible implication statements using a table which starts at rank 0 and creates subsequent ranks using statements from preceding ranks. The distributions provided by KBGenerator [13] are different from those provided by KBGT [10]. The tool also has an optimised version, KBGeneratorThreaded. This uses multi-threading to improve the efficiency of defeasible implication statement generation.

7.5 KLMDEETool

The KLMDEETool [8] is a software system tool that provides both a graphical user interface and command-line interface for users to utilise an imperative implementation of defeasible entailment. It aims to facilitate users' understanding by providing explanations for results in a readily understandable human format. However, while the tool effectively performs computations for rational closure, its presentation of information and elements is not intuitive, necessitating additional guidance to use. The tool's visual appeal is lacking, and its interface could benefit from a cleaner design to enhance user experience.

8 ANTICIPATED OUTCOMES

At the end of this project, we expect to have a correct and fully functional defeasible reasoner, an optimised version of the reasoner, and a knowledge base generator. Furthermore, we expect to have a debugger for stepping through rational closure computations. The deliverables produced will allow us to evaluate the flexibility, compactness, and potential reduction in computational runtime of the ASP environment.

8.1 Key Success Factors

- A naive and correct implementation of rational closure developed using ASP.
- An optimised version of our naive implementation, in terms of its runtime.
- A correct implementation of a knowledge base generator developed using ASP.
- A qualitative evaluation of the flexibility and compactness of ASP for modelling defeasible conditionals.
- A minimum viable product (MVP) of the RCD. The features the MVP should possess will be defined in the User Requirements Phase of the project.

9 PROJECT PLAN

9.1 Deliverables

Deliverable	Due Date
Literature Review	25 March
Project Proposal (First Draft)	22 April
Project Proposal Presentations	22 - 25 April
Project Proposal (Final Paper)	30 April
Project Progress Demonstration	22 July - 26 July
Complete Draft of final paper due	23 August
Project Paper Final Submission	30 August
Project Code Submission	9 September
Final Project Demonstration	16 - 20 September
Project Poster	27 September
Project Website	4 October

9.2 Milestones and Tasks

Milestones and tasks identified found in Table 4.

9.3 Resources

In terms of the theoretical aspects of the project, access to relevant journals and research papers will be required. For the implementation aspect, we will require working laptops with a code editor. The laptops should also be capable of running Clingo [4]. We will also need access to previous students' projects for testing and benchmarking.

9.4 Risks

Risks identified found in Table 2.

9.5 Work Allocation

Student	Work Allocation
Racquel, Jack, Sibusiso	Naive implementation of rational closure
Jack	Optimisations
Racquel	Knowledge base generator
Sibusiso	Debugger

9.6 Time line

The Gantt chart (1) in the appendix indicates our planned timeline.

REFERENCES

[1] Mordechai Ben-Ari. 2012. *Mathematical logic for computer science* (3 ed.). Springer, London. 4–47 pages. https://doi.org/10.1007/978-1-4471-4129-7

[2] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. 2011. Answer set programming at a glance. *Commun. ACM* 54, 12 (dec 2011), 92–103. https://doi.org/10.1145/2043174.2043195

[3] Menachem Magidor Daniel Lehmann. May 1992. What does a conditional knowledge base entail? *Journal of Artificial Intelligence* Vol. 55 no.1 (May 1992), 1–60.

[4] Kaufmann B. Neumann A. Schaub T. Gebser, M. 2007. clasp: A Conflict-Driven Answer Set Solver. *Baral, C., Brewka, G., Schlipf, J. (eds) Logic Programming and Nonmonotonic Reasoning. LPNMR 2007. Lecture Notes in Computer Science(), vol 4483. Springer, Berlin, Heidelberg.* (2007).

[5] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. 2017. Multi-shot ASP solving with clingo. *CoRR* abs/1705.09811 (2017).

[6] Schaub T. Thiele S. Gebser, M. 2007. GrinGo: A New Grounder for Answer Set Programming. *In: Baral, C., Brewka, G., Schlipf, J. (eds) Logic Programming and Nonmonotonic Reasoning. Lecture Notes in Computer Science(), vol 4483. Springer, Berlin, Heidelberg.* (2007).

[7] Weiwei Gong and Xu Zhou. 2017. A survey of SAT solver. *AIP Conference Proceedings* 1836, 1 (06 2017), 020059. https://doi.org/10.1063/1.4981999 arXiv:https://pubs.aip.org/aip/acp/article-pdf/doi/10.1063/1.4981999/13742154/020059_1_online.pdf

[8] Chipo Hamayobe. 2023. The KLM Defeasible Entailment and Explanations Tool. *Faculty of Science, University of Cape Town, Rondebosch, Cape Town, 7700.* (2023). https://github.com/ChiefMonk/DEE

[9] Joel Hamilton. 2021. An Investigation into the Scalability of Rational Closure. *Faculty of Science, University of Cape Town, Rondebosch, Cape Town, 7700.* (2021). https://projects.cs.uct.ac.za/honsproj/cgi-bin/view/2021/bailey_hamilton_park.zip/files/SCADR_HMLJOE001.pdf

[10] Aidan Bailey Joel Hamilton, Joonsoo Park and Thomas Meyer. 2021. An Investigation into the Scalability of Defeasible Reasoning Algorithms. *SACAIR 2021 Organising Committee, Online, 235–251* (2021). https://protect-za.mimecast.com/s/OFYSCpgo02fL1l9gtDHUkY

[11] Adam Kaliski. 2020. An Overview of KLM-Style Defeasible Entailment. Master's thesis. *Faculty of Science, University of Cape Town, Rondebosch, Cape Town, 7700.* (2020). http://hdl.handle.net/11427/32743

[12] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. 1990. Nonmonotonic Reasoning, Preferential Models and Cumulative Logics. *Journal of Artificial Intelligence* 44 (1990), 167–207.

[13] Alec Lang. 2023. Extending Defeasible Reasoning Beyond Rational Closure. https://projects.cs.uct.ac.za/honsproj/cgi-bin/view/2023/lang_pullinger_slater.zip/resources/LNGALE007FinalPaper.pdf

[14] Benjamin Kaufmann Martin Gebser, Roland Kaminski and Torsten Schaub. 2012. Answer Set Solving in Practice. *Synthesis Lectures on Artificial Intelligence and Machine Learning* (2012). doi:10.2200/S00457ED1V01Y201211AIM019

[15] Evashna Pillay. 2022. *An Investigation into the Scalability of Rational Closure V2.* Honour's Project. Faculty of Science, University of Cape Town, Rondebosch, Cape Town, 7700. https://projects.cs.uct.ac.za/honsproj/cgi-bin/view/2022/pillay_thakorvallabh.zip/files/PLLEVA005_SCADR2_FinalPaper.pdf

[16] Hector J. Levesque Ronald J. Branchman. 2004. *Knowledge representation and reasoning.* Morgan Kaufmann Publisher.

10 APPENDIX

10.1 Risks

ID	Risk	Likelihood	Impact
1	Supervisor unavailable	Unlikely	Moderate
2	Team member drops out	Possible	Moderate
3	Conflict on how to design the initial implementation	Possible	Moderate
4	A team member fails to contribute in the initial phase of implementing rational closure in ASP	Unlikely	Moderate
5	Poor communication within the team	Rare	Significant
6	Difficulty understanding the theoretical aspects of the project	Possible	Significant
7	Difficulty implementing rational closure using ASP	Possible	Significant
8	Scope creep during the individual phases of the project	Possible	Significant

Table 2: Risk Identification

ID	Mitigation	Monitoring	Management
1	Ensure that our supervisor's schedule is checked regularly.	Constant communication with our supervisor which is conducted via emails and meetings.	Continue with the project and write down the questions to raise in the next meeting.
2	Ensure that there is an open dialogue between team members and our supervisor.	Regularly check in with team members about mental and physical well-being.	Reassess the project scope to ensure that the core aim of the project can still be delivered.
3	Ensure that there is a collaborative effort and that each team member contributes during the planning phase.	Confirm that each team member agrees with the proposed design before moving to the next stage.	Request that the supervisor assist in mediating the disagreements that have arisen.
4	Implement regular check-ins as a team to ensure that everyone is held accountable for their work.	Use of Kanban board to keep track of each member's progress	Consult with our supervisor and make them aware of the current situation about each team member's inputs.
5	Keep in constant communication with team members through platforms such as WhatsApp and email.	Ensure that everyone has had their say in aspects of the project.	Have an urgent meeting with team members and supervisor to resolve poor communication.
6	Review relevant literature papers on the theory implemented in the project.	Have regular meetings with our supervisor to discuss what we have done as a team and explain the theoretical components to them to ensure we understand it.	Consult with our supervisor to assist in helping our understanding.
7	Ensure that there is an understanding of rational closure and ASP.	Regular meetings our supervisor to discuss what has been achieved and any potential barriers we may be facing.	Consult with our supervisor to assist with our understanding.
8	Outline a project plan that is consistently updated with the progress made. Ensure the project plan outlines the requirements needed for the project to succeed.	Regularly evaluate work achieved and ensure it is in line with what is outlined in the project plan.	Discuss with our supervisor to realign the project and discuss what work should be discarded.

Table 3: Risk Mitigation, Monitoring and Management

10.2 Milestones and Tasks

Tasks	Start Date	End Date
Literature Review	19/02	25/03
Project Proposal	25/03	29/04
First Draft	25/03	22/04
Project Proposal Presentation	22/04	26/04
Final Draft	27/04	29/04
Theoretical Background	30/04	13/05
Algorithm Implementation	14/05	27/06
Outline of implementation approach	14/05	30/05
Implementation of Rational Closure	31/05	20/06
Test our implementation for correctness	21/06	27/06
Optimisation	31/05	05/08
Background Reading	31/05	06/06
Develop benchmarking tool	31/05	23/06
Plan Optimisation Approach	07/06	13/06
Optimisation Plan Implementation	24/06	18/07
Benchmark against implementation with SCADR and SCADRv2	28/06	10/07
Test optimisations for correctness	19/07	25/07
Benchmark optimisations against our implementation, SCADR and SCADRv2	26/07	31/07
Analysis of benchmarking results	31/07	05/08
Knowledge Base Generator	22/04	02/08
Background Reading	22/04	11/05
Outline Implementation Approach	13/05	25/05
Implement Knowledge Base Generator	27/05	02/07
Test the correctness of implementation	02/07	09/07
Extend implementation to produce a variety of different knowledge bases	09/07	30/07
Analyse implementations	30/07	10/08
Debugger	01/05	02/08
Background Reading and Research	01/05	08/05
UML Diagrams and Wireframes	09/05	20/05
Debugger Main User Interface	21/05	21/06
Implement Computation Step-Through functionality	23/06	07/07
Implement Process Computation Explanation functionality	08/07	28/07
Final User Acceptance Testing	29/07	02/08
Project Progress Demonstration	22/06	26/06
Final Paper		30/08
First Draft Submission		23/08
Final Draft Submission		30/08
Project Code Final Submission		10/09
Final Project Demonstration	16/09	20/09
Project Poster Submission		27/09
Project Website Submission		04/10

Table 4: Milestones and tasks

10.3 Gannt Chart

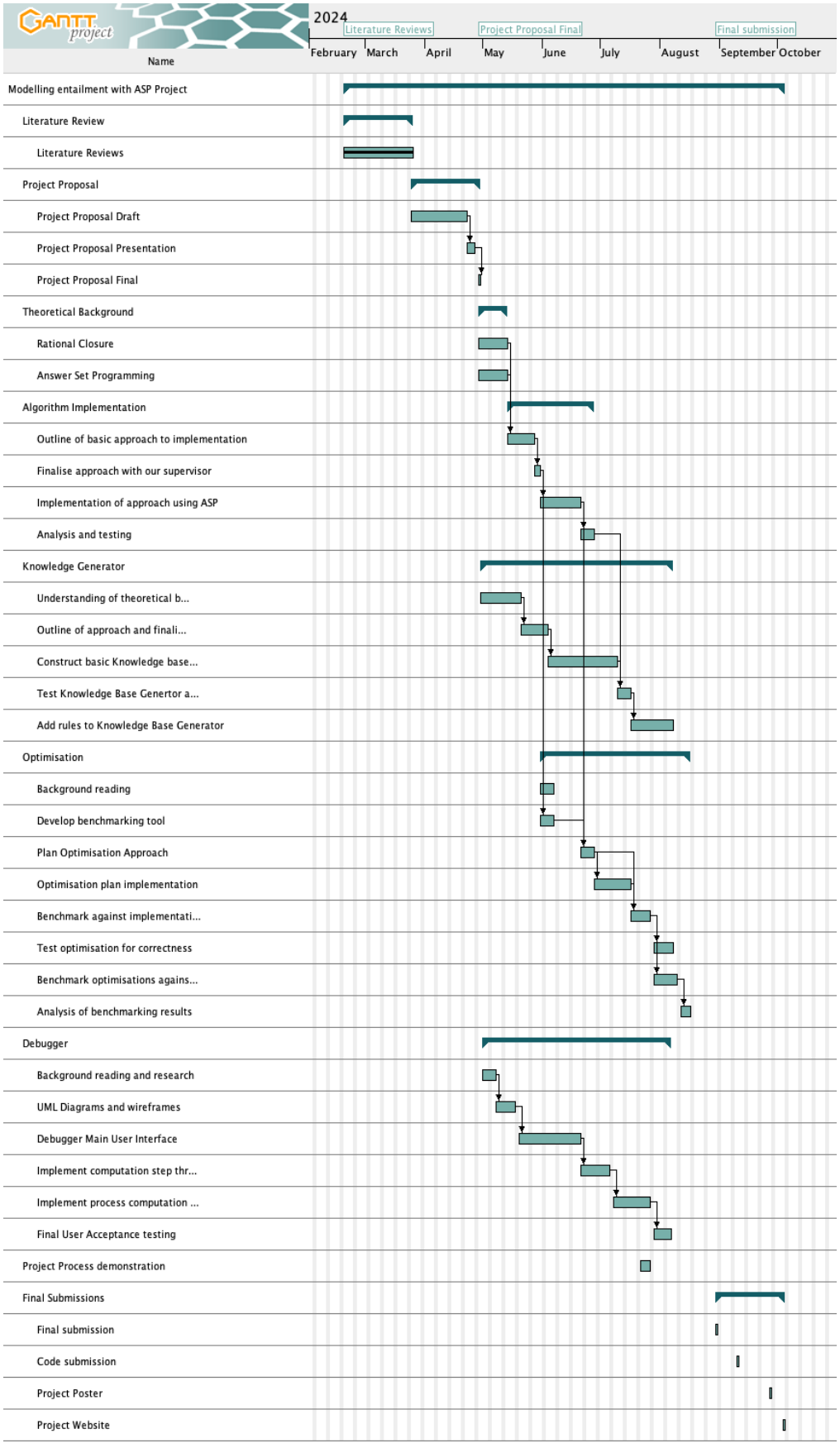

Figure 1: Project timeline for Defeasible conditionals in answer set programming project.