Preferential Reasoning for Ontologies

Project Proposal

Michael Harrison
Department of Computer Science
University of Cape Town

Reid Swan
Department of Computer Science
University of Cape Town

Supervisor: Tommie Meyer
Department of Computer Science
University of Cape Town

Second Reader: Deshen Moodley
Department of Computer Science
University of Cape Town

ABSTRACT

Non-monotonicity is a property desired in description logics to increase the scope and power of automated reasoners. This document proposes a project to find and implement an algorithm to compute the minimum possible modification of a user-defined ranking function of statements in a defeasible description logic knowledge base which will result in a rational preference relation. It provides introductory and background information on the relevant fields, including an overview of description logics, attempts to enrich these with non-monotonicity, a definition of a rational preference relation and an overview of the KLM postulates on which rational preference relations are based. The main problems to be solved are discussed, including the importance and potential impacts of solving them. Challenges that may arise in the course of solving them are also given. The project is then summarized in the following two research questions: "Can a provably correct algorithm be found to produce a rational preference relation given a user-informed preference relation?" and "How can this algorithm be implemented for use in an ontology editor?" An approach is given for solving the identified problems, as well as a timeline and manners in which to measure project success. A set of risks is identified and a plan to monitor, manage and mitigate them is provided, as well as a set of milestones and a timeline over which the project is to be completed. Finally, the work is logically divided into theoretical and implementation sections, with each section to allocated to an author.

1 INTRODUCTION

Description Logics (DLs) are fragments of first-order logic used for representing knowledge in a domain such that the knowledge can be reasoned with by an automated system [1]. Description logics are frequently used to represent ontologies in order to leverage the mature and efficient algorithms for reasoning with them.

A (classical) DL knowledge base is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a set of terminological axioms known as a TBox, which refer to concepts, and $\mathcal{A}$ is a set of assertional axioms known as an ABox, which refer to individuals which belong to those concepts [1].

A common prototypical description logic is $\mathcal{ALC}$, first introduced by Schmidt-Schauß and Smolka[20]. An $\mathcal{ALC}$ knowledge base is composed of a set of concepts, N_C, a set of individuals N_A which are objects belonging to the concepts, and a set of roles, N_R, which are collections of pairs of individuals and define relationships between them. For example, if *CarCompany* is a concept, then *audi* could be an individual belonging to this concept, denoted $audi : CarCompany$. If *volkswagen* is an individual that is another *CarCompany*, and $parentOf$ is a role, then *volkswagen* is in a $parentOf$ role with *audi*, denoted $(volkswagen, audi) : parentOf$.

It is possible to draw conclusions from a knowledge base following a universally accepted definition known as entailment and given the symbol $\models$. Essentially, a knowledge base entails some statement if and only if that statement is necessarily true in every interpretation of the knowledge base which does not violate any of the knowledge base's statements. For example, consider the knowledge base $\mathcal{K} = (\{FruitTree \sqsubseteq Tree, Tree \sqsubseteq Plant\}, \emptyset)$, which can be understood as "every FruitTree is a Tree; every Tree is a Plant"; then $\mathcal{K} \models FruitTree \sqsubseteq Plant$ since every FruitTree is a Tree which is a Plant. Entailment is a universally accepted definition for reasoning with a DL knowledge base; automated reasoners use this definition of entailment to efficiently reason with knowledge bases.

Entailment is monotonic, which means that any conclusions drawn from a knowledge base will never be invalidated by adding new axioms. It is desirable to enrich description logics with non-monotonicity, the ability to retract conclusions in light of new information. For example, suppose a knowledge base contains the statement that mammals dwell on land (which could be written $Mammal \sqsubseteq LandDweller$) and that whales are mammals ($Whale \sqsubseteq Mammal$) and that whales dwell in the sea ($Whale \sqsubseteq SeaDweller$). Also, of course, things that dwell in the sea do not dwell on the land ($LandDweller \sqsubseteq \neg SeaDweller$). If we are told that x is a *Mammal*, we conclude that x is a $LandDweller$. Given the further information that x is a $Whale$, we can only conclude that x is also a $SeaDweller$; but things that are $SeaDweller$s cannot also be $LandDweller$s and so our knowledge base is inconsistent. We would like to be able to retract the conclusion that x is a $LandDweller$, but under monotonicity this is not possible.

A variety of non-monotonic logics have been proposed, as well as ways in which to apply them to DLs, including circumscription[3, 14], default reasoning[2, 19], defeasible logic[18, 22], and preferential reasoning[4, 11, 13]. We restrict our focus to preferential reasoning.

Preferential reasoning was introduced by Kraus, Lehmann and Magidor[11] as a form of non-monotonic reasoning as an extension of classical propositional logic. They introduced a set of six inference rules, known as the KLM postulates, which it is argued that any preference relation should satisfy. If a preference relation satisfies all of the KLM postulates then it is a *rational* preference relation.

Enriching DLs with preferential reasoning requires the addition of a new operator, known as defeasible subsumption and given the symbol $\tilde{\sqsubseteq}$. $A \tilde{\sqsubseteq} B$ can be interpreted to mean the concept A is *typically* contained within the concept B. This was first proposed by Britz et al.[4]. Informally, defeasibility is defined as the ability to make a generalization or assumption which can then be revised in light of new information, and is exactly the property desired of non-monotonic logic. However, simply adding the defeasible subsumption operator and keeping the definition of entailment the same is insufficient and will not result in non-monotonicity. Therefore the definition of entailment must be changed; in order to inform this change, we look to the KLM postulates.

Rational closure was first given by Lehmann and Magidor[13] and applied to DLs by Casini and Straccia[7], and using the defeasible subsumption operator by Casini et al.[5]. A defeasible knowledge base is defined as a pair $\mathcal{K} = (\mathcal{T}, \mathcal{D})$ where $\mathcal{T}$ is a set of classical DL statements, the TBox, and $\mathcal{D}$ is a set of defeasible subsumption statements, the DTBox. Rational closure is an operation on such a defeasible knowledge base. The operation of rational closure has been proven to comply with all of the KLM postulates, so by definition it is a rational preference relation. It is not, however, the only such rational preference relation which can be computed from a given knowledge base. Instead, it represents a 'baseline', the most conservative rational preference relation, as indicated by Lehmann and Magidor[13].

Casini et al.[5] demonstrate that rational closure can be reduced to performing a number of classical entailment checks, allowing it to be performed by existing mature and efficient reasoning platforms. A procedure for computing the rational closure of a defeasible knowledge base has been implemented as a plugin for the Protégé ontology editing platform by Moodley[15], called the Defeasible Interface Platform, or DIP. The implementation takes advantage of this reduction to classical entailment checks, allowing for excellent real world performance[5]. DIP gives defeasible axioms a *ranking* based on how *exceptional* the axiom is. The more general an axiom, the lower its *ranking*. If the *rational closure* algorithm takes as input $C \tilde{\sqsubseteq} D$, it determines the set of most general axioms that do not imply the negation of C[6].

2 BACKGROUND

The KLM postulates can be expressed using the defeasible subsumption operator as follows:

$$\frac{A \equiv B, A \tilde{\sqsubseteq} C}{B \tilde{\sqsubseteq} C} \quad \textbf{(Left Logical Equivalence)}$$

$$\frac{A \tilde{\sqsubseteq} B, C \tilde{\sqsubseteq} A}{C \tilde{\sqsubseteq} B} \quad \textbf{(Right Weakening)}$$

$$A \tilde{\sqsubseteq} A \quad \textbf{(Reflexivity)}$$

$$\frac{A \tilde{\sqsubseteq} B, A \tilde{\sqsubseteq} C}{A \tilde{\sqsubseteq} B \sqcap C} \quad \textbf{(And)}$$

$$\frac{A \tilde{\sqsubseteq} C, B \tilde{\sqsubseteq} C}{A \sqcup B \tilde{\sqsubseteq} C} \quad \textbf{(Or)}$$

$$\frac{A \tilde{\sqsubseteq} B, A \tilde{\sqsubseteq} C}{A \sqcap B \tilde{\sqsubseteq} C} \quad \textbf{(Cautious Monotonicity)}$$

$$\frac{A \sqcap B \not\tilde{\sqsubseteq} C, A \not\tilde{\sqsubseteq} \neg B}{A \not\tilde{\sqsubseteq} C} \quad \textbf{(Rational Monotonicity)}$$

These can be understood as follows:

- **Left logical equivalence**: if A and B are identical and A is typically a C, then it is reasonable to conclude that B is typically a C. For example, if Wet and NotDry are equivalent, and Wet things are typically Slippery, then we can conclude that NotDry things are typically Slippery.
- **Right weakening**: if every A is a B and C is typically an A then it is reasonable to conclude that C is typically a B. For example, if every Human is a Biped, and Bipeds typically Walk, then it is reasonable to conclude that Humans typically Walk.
- **Reflexivity**: if something is an A, then it is typically an A.
- **And**: If an A is typically a B and an A is typically a C, then it is reasonable to assume that it is typically both a B and a C. For example, if a Car is typically Fast and a Car is typically Heavy then it is reasonable to assume that a typical Car is both Fast and Heavy.
- **Or**: If an A is typically a C and a B is typically a C then things that are either an A or a B are typically a C. For example, if Monkeys are typically TreeDwellers and Birds are typically TreeDwellers, then things that are either Monkeys or Birds are typically TreeDwellers.
- **Cautious Monotonicity**: If an A is typically a B and an A is typically a C, then something that is both an A and a B is typically a C. This is reasonable because the most typical As are Bs, so when we have something that is both an A and a B, we can assume it is typical; and if it is typical, then it must also typically be a C, by virtue of being an A.
- **Rational Monotonicity**: If something that is both an A and a B is not typically a C, and A is not typically not a B, then it would not be reasonable to conclude that an A is typically a C. For example, something that is both *Loud* and *Certain* is not typically *Correct*, and something that is *Certain* is not typically not *Loud*, so being *Certain* does not typically make you *Correct*. This makes sense, because if an object that has properties A and B does not typically have property C, and having property A does not

typically exclude property B, then we have no information on whether an object with property A has property B or not; and so we cannot deduce whether A and B are both present, so conclusions about the presence of property C cannot be made under typical circumstances.

3 PROJECT DESCRIPTION

3.1 The problem

Still missing from the literature of non-monotonic description logics is a formally proven algorithm for a computing rational preference relation from a conditional knowledge base when the ranking of the statements in the DTBox is supplied by the user. In a standard defeasible knowledge base, all statements in the DTBox are treated as having the same ranking; it is conceivable, however, that a user may have more information about the importance of statements in the knowledge base that can be conveyed by ranking statements, but arbitrary rankings are not guaranteed to be rational. Therefore, an algorithm must be designed which, given a user-defined ranking of statements in the DTBox, will modify them to produce one that is rational preference relation – in other words, one which satisfies all of the KLM postulates. This alteration must be the minimum possible alteration, for some appropriate definition of minimum, so that the resulting rational preference relation aligns as closely as possible with what the user intended. The correctness and complexity of this algorithm must be formally proven. Furthermore, the algorithm should be implemented in an ontology editor so that it can be used in a real world setting and as a proof of concept.

3.2 Why it's important

Solving this is important because it is a generalization of rational closure; there are many areas where classical reasoning is not appropriate and preferential reasoning is required, and this would give the ontology designer a finer-grained control over how the statements in their DTBox are ranked. Implementation would serve as a proof of concept, and allow preferential reasoning to be experimented with and utilized in real-world settings.

3.3 Possible issues or difficulties

3.3.1 Theory.

- There are no clear steps or guidelines to determine a new algorithm
- Finding a satisfactory definition of minimal modification of a user ranking function may be difficult
- It is difficult to estimate the time for finding an algorithm or its proof of correctness

3.3.2 Implementation.

- No specialized preferential reasoning tool exists, so there is no clear guide as to how to implement it
- Unable to incorporate the user-informed preferential ranking until it is completed
- Creating a new standalone tool could involves a lot of work that could introduce errors and complications

4 PROBLEM STATEMENT

4.1 Aims

The aims of this project are:

- to find a satisfactory definition of minimal changes to a user's ranking function
- to find an algorithm that performs the minimal changes in a way that results in a rational preference relation
- to prove the correctness and computational complexity of the algorithm
- to reduce the algorithm to a sequence of classical entailments
- to implement a standalone ontology edi0tor which allows users to reason with the algorithm

4.2 Research Questions

The problem to be solved is summarized in the following two research questions:

- Can a provably correct algorithm be found to produce a rational preference relation given a user-informed preference relation?
- How can this algorithm be implemented for use in an ontology editor?

5 PROCEDURES AND METHODS

5.1 Approach

The problem is divided into two logical sections: theory and implementation.

5.1.1 Theory. The theory will be approached by determining an algorithm that will generate a rational preference relation given a user's desired ranking; this must then be proven to be correct.

More specifically, it must be shown that given any ranked input, the output of the algorithm will always satisfy the KLM postulates, up to and including rational monotonicity. In addition to proving that the results of the algorithm satisfy these, it must also be shown that the modification of the user's given ranking is minimal, although the exact definition of this minimality is still to be determined. Computational complexity results of the algorithm are also to be proven.

5.1.2 Implementation. The implementation of preferential reasoning will be approached by developing a standalone ontology editing tool. This tool will allow a user to create a defeasible ontology, give a ranked preference of defeasible axioms, and perform preferential reasoning on the ontology that takes into account the user-provided ranked preferences. The ontology creation and editing aspect will draw inspiration from existing classical ontology tools, such as Protégé. The tool will provide an intuitive graphical user interface that will make constructing and editing ontologies quick and simple. The other, and more critical, aspect of the tool will be the capability to apply preferential reasoning to the defeasible ontology. The Defeasible Inference Platform will be incorporated into the tool to provide defeasible reasoning. Instead of using a *ranking* of the defeasible axioms based on *exceptionality* of the axioms, however, the *ranking* will be provided by Reid Swan's preferential relation algorithm that takes user-defined preferences into

account. The Defeasible Inference Platform will take this ranking and use a classical reasoner to perform classical entailment checks to whether new defeasible inferences are contained in the rational closure.

Initial testing and prototyping will be performed using Protéé ontology editor and plugins for Protégé that are developed using Java and compiled using Eclipse.

5.2 Testing Results

- **Theory** – The correctness of the algorithm will be given in its proof. It will not require testing, but rather verification by an authority on the subject that the proof is correct. The alteration made to the user-defined ranking must be minimal, and this minimality is to be defined; it must be ensured that the definition is an appropriate one.
- **Implementation** – Unit tests will be written for the software to ensure that it meets requirements at every stage of development and that features do not introduce regressions of the software at any stage.

  Data/ontologies for testing will be generated from existing ontologies that are sourced from existing (classical) ontology databases. The knowledge bases will be extended to be defeasible knowledge bases. User preferences will be read in and inform the algorithm when it reasons with the new defeasible knowledge base. They will be manually run through the reasoner and the output verified by hand to ensure it meets requirements.

  The ontology editor's user interface will be prototyped and tested with potential users to ensure the product is usable and intuitive. Students who completed the Ontology Engineering module this year are potential test users.

5.3 Measurements of Success

Success of the project will be measured by the following criteria:

- All unit test cases and all manual tests are passed.
- The following has been determined about the algorithm:
  - the algorithm is correct
  - the algorithm's computational complexity has been determined
  - a satisfactory definition of minimal modification of a user ranking function has been found and the algorithm complies with it
- The software is feature complete, containing at least the following features:
  - the ability to create, import and export preferential ontologies
  - the ability to reason and answer queries on preferential ontologies
  - a user-friendly interface to view and edit preferential ontologies

A successful project will have impacts in the field of knowledge representation and reasoning as a generalization of existing knowledge. Ontology engineering will be impacted through the provision of a new tool for working with ontologies and allowing the modeling of knowledge in a domain in ways which were previously not possible due to the limitations of classical reasoners, or allowing existing ontologies to be simplified through application of the tool. The scope of what automated reasoners can do will be increased by allowing user-informed rankings to be used in non-monotonic reasoning in a manner proven to be consistent with the KLM postulates. This will also allow experts in the field to have finer-grained control over how their preferential reasoners treat the defeasible knowledge.

6 ETHICAL, PROFESSIONAL AND LEGAL ISSUES

6.1 Ethical Issues

This project poses no inherent ethical issues. The only other possible ethical issues would arise during prototype issues if test users are exploited in any way.

6.2 Professional Issues

There are no foreseeable professional issues with the project. The project provides a basis for further possible study.

6.3 Legal Issues

All software that will be utilised is open source. The tool that will be created will also be open source software, as to avoid any possible legal issues and to allow for use in further research.

7 RELATED WORK

7.1 Theory

Kraus, Lehmann and Magidor [11] introduce preferential reasoning and the KLM postulates, and show that not all preference relations satisfy all of the KLM postulates, and so are not rational preference relations. Lehmann and Magidor [13] present the notion of rational closure, as well as an algorithm to compute it. It is shown that rational closure results in a rational preference relation, and that rational closure represents the most conservative such preference relation – any others would necessarily be a superset of rational closure. Britz et al.[4] introduce the defeasible subsumption operator, $\sqsubseteq$, for enriching description logics with preferential reasoning. Casini et al.[7] give an algorithm for rational closure for description logics, and it is shown to reduce to a sequence of classical entailment operations. Casini et al.[5] give the same algorithm reformulated using the defeasible subsumption operator of Britz et al. Lehmann[12] presents another procedure for computing a rational preference relation for propositional logic, known as the lexicographic closure, which entails a strict superset of everything entailed under rational closure. Casini et al.[8] give the algorithm for lexicographic closure for description logics with defeasible subsumption statements.

7.2 Implementation

Moodley's Defeasible Inference Platform (DIP) is the most closely related work to an implemented preferential ontology editor and reasoner. DIP is a plugin for Protégé ontology editor that provides a defeasible reasoning capability through multiple methods, namely: Preferential, Rational, and Lexicographic Closure[16]. The Rational Closure functionality will be utilised in the implementation of the standalone preferential ontology editor and reasoner to provide

defeasible reasoning once a ranking has been provided by the rational preference relation algorithm on the user-supplied preference relation.

8 PROJECT PLAN AND WORK ALLOCATION

8.1 Risk

A list of risks and their allocated identification numbers are given in Appendix A.

Appendix B shows a priority rating for each risk based on the probability of the risk occurring and the impact the risk would have on the project if it were to occur. As can be seen in Appendix B, the most important risks to address are users being unavailable for prototype testing and the users who are available not having the required knowledge to accurately test the prototypes.

Appendix C provides steps to mitigate, monitor, and manage each risk. The mitigation steps aim to reduce the likelihood and/or impact of the risks. The monitoring steps detail how to keep track of the likelihood of the the risks occurring. The management steps outline contingency plans to minimize the damage risks will cause after they occur.

8.2 Timeline

The project is split into the theory element of developing a rational user preference relation algorithm and the implementation element of implementing this algorithm within a standalone preferential ontology editing and reasoning tool. The only dependency in the project is the incorporation of the algorithm in the tool being dependent on the completion of the algorithm. The project timeline can be seen in the form of a Gantt chart in Appendix D.

8.3 Resources required

Various tools will be required to create the standalone preferential ontology editing and reasoning software. These tools will either be directly used in creating the software or as a reference.

- **OWL API** – The underlying ontology representation will be written in the OWL 2 web ontology language[21]. OWL 2 DL is a subset of OWL 2 that is based on description logics. The ontology needs to be represented in OWL to be reasoned with computationally. The OWL API is, therefore, required to support the creation and manipulation of OWL ontologies[10] from within the standalone program. The OWL API also supports the implementation of classical reasoners. The OWL API is implemented in Java.
- **Defeasible Inference Platform** – The Deafeasible Inference Platform is a Protégé plugin developed by Moodley [17]. This plugin allows for defeasible reasoning. This plugin computes a *ranking* of all the axioms in the knowledge base. The preferential algorithm will be integrated into this platform to produce a new *ranking* that will be used to perform preferential reasoning. The platform will need to be modified for use in the standalone preferential ontology editing and reasoning tool instead of Protégé.
- **Classical OWL Reasoner** – An existing reasoning platform will be performing the classical reasoning that is required in computing the *ranking*. The computation of the requirement includes performing classical DL entailment checks. The classical reasoner to be used is most likely to be HermiT. HermiT is a logical choice as it is implemented in Java, can perform reasoning on OWL 2 DL or SROIQ ontologies, uses OWL API 3.0, and is compatible with the OWLReasoner interface of the OWL API Glimm et al. [9].
- **Protégé Ontology Editing Platform** – The open source ontology editing platform, Protégé, will be used as a reference. It's source code will be used for inspiration and as a guide when designing the interface and functionality of the standalone preferential ontology editing and reasoning tool. Protégé will also be used for testing purposes due to the ease with which plugins can be developed and added to it.
- **Programming language and IDE** – The OWL API and the Defeasible Inference Platform are both written in Java. Using Java for the standalone preferential ontology editing and reasoning tool will make using the OWL API and integrating the Defeasible Inference Platform as seamless as possible. The Protégé ontology editor is also implemented in Java. Protégé's functionality and interface will be used as a guide and reference when creating the standalone preferential ontology editing and reasoning tool. This is further motivation to use Java, as any inspiration that is drawn from Protégé will not need to be rewritten in another language.

  The Eclipse IDE will be used because it provides support for building plugins. This feature allows for easy experimentation and greater control when developing the tool.

8.4 Deliverables

The only deliverable for the theory aspect of the project is the rational preference relation algorithm for user-informed preferential reasoning.

The deliverables for the implementation aspect include the initial working prototype of the standalone tool, the second prototype of the tool, and the final completed tool. The initial prototype will include the basic required interface, ontology editing capabilities, and the incorporation of the Defeasible Inference Platform. The second prototype will include the rational preference relation algorithm to provide a user-informed ranking. The final product will be a refinement of the second prototype

8.5 Milestones

The milestones for the theory aspect of the project are as follows:

- Investigate lexicographic closure and how it differs from rational closure, and discuss with supervisor
- Investigate the proof that rational and lexicographic closure satisfy the KLM postulates
- Attempt proof for general user-provided ranking to discover where it fails; investigate what modifications would prevent failure
- Determine a formal definition of what a minimal modification of a user ranking function is, based on intuition and what seems most natural if applied to a given ranking of statements

- Develop algorithm based on this modification
- Prove that the algorithm is the most minimal modification based on the definition of minimum chosen
- Prove that the resulting ranking function satisfies the KLM postulates
- Prove computational complexity results of the algorithm
- Show that the algorithm reduces to a set of classical entailments

The milestones for the implementation aspect of the project are as follows:

- Experiment and test with DIP and Protégé plugins
- Create a Java-based standalone ontology editor
- Incorporate DIP into the standalone tool
- Initial working standalone tool prototype for feasibility demonstration
- Incorporate rational preference relation algorithm into the tool
- Second working standalone tool prototype iteration
- Complete final standalone tool

8.6 Work Allocation

The project can be divided logically into two sections, namely theory and implementation.

The theory section involves developing the algorithm for altering a provided user ranking of statements of a conditional knowledge base into one which complies with the KLM postulates as previously mentioned, as well as proving the correctness thereof. This section will be performed by Reid Swan.

The implementation section involves implementing a standalone ontology editor with standard features, including an implementation of the algorithm developed for user-informed preferential reasoning. It will allow for creating, importing and exporting preferential knowledge bases, as well as displaying the contents of a preferential knowledge base, specifying the ranking of statements by the user and querying the reasoner. This section will be performed by Michael Harrison.

9 CONCLUSIONS

Description logics are a powerful formalism in knowledge representation that is used frequently for representing ontologies due to their ability to reason efficiently. However, the monotonicity property means that there are a number of situations which cannot be modeled in a simple way, if at all, in description logics at present. To this end, non-monotonicity is a desirable property. Preferential reasoning is a well-defined model for non-monotonic reasoning that has already been successfully applied to description logics through the procedure of rational closure. The next logical step is to generalize preferential reasoning by allowing users to supply their own ranking of statements in their defeasible knowledge base. However, a rational preference relation – one which satisfies all of the KLM postulates – is desired for reasoning purposes. An algorithm must be determined to perform the minimal modification of the user's given ranking function that will result in a rational preference relation; the definition of minimality must be determined, and the algorithm must be shown to adhere to it. This algorithm must also be implemented in a standalone ontology editor so that it can be experimented with and utilized in a real world context.

The preceding document has indicated a plan for accomplishing this, including an approach, a number of measurements of success, risks and a plan for their monitoring, mitigation and management, and a timeline over which the goals are to be accomplished. The impacts of a successful project include generalizing existing knowledge in the field of knowledge representation and reasoning, provision of a new tool for ontology engineers, and increasing the scope of how description logics can be used to model situations by providing a rigorously-defined version of non-monotonicity with fine-grained control over how the reasoning is performed. The work has an obvious division into theory and implementation sections, which can be performed largely in parallel.

A RISK TABLE

ID	Risk
1	Supervisor unavailable
2	Users unavailable for prototype testing
3	Lack of knowledge or understanding from test users
4	Partner dropping out
5	Partner not completing theory
6	Partner not completing tool (proof of concept)
7	Conflict in group
8	Underperformance
9	Poor communication/absence
10	Obtaining defeasible plug-in resource late/never
11	Poor time management

Table 1: List of Risks

B RISK PRIORITY TABLE

ID	Probability	Impact	Priority
1	Medium	Marginal	Averagely important
2	High	Critical	Very important
3	High	Critical	Very important
4	Low	Catastrophic	Moderately important
5	Low	Catastrophic	Moderately important
6	Low	Marginal	Not very important
7	Low	Critical	Averagely important
8	Low	Marginal	Not very important
9	Medium	Marginal	Averagely important
10	Medium	Critical	Moderately important
11	Medium	Marginal	Averagely important

Table 2: Risk Priority

C RISK MANAGEMENT TABLE

ID	Mitigation	Monitoring	Management
1	Plan meetings ahead of time	Be in regular contact	Consult each other and email supervisor
2	Be proactive and search for users early on	Have backup users to test if original users drop out	Contact wider student base for possible users
3	Actively seek out users with domain knowledge	Keep track of the number of informed users who have confirmed to be in the testing	Train new users before they test the prototype
4	Separate the work as much as possible	Check on partner's status regularly	Obtain missing information from supervisor
5	Enforce strict deadlines	Check progress regularly	Obtain required theory form Tommie
6	Enforce strict deadlines	Check progress regularly	Prove in another way
7	Have group conduct rules	Be open and honest regarding group sentiments	Approach partner and attempt to resolve in a civil manner
8	Enforce deadlines and drafts	Observe performance and provide feedback to drafts	Help partner and enforce extra work
9	Enforce group conduct rules	Keep track of offenses	Raise issue and report if necessary
10	Follow-up with Dr Moodley	Ask Dr Moodley for progress	Use alternative or fix plug-in
11	Enforce group conduct rules	Keep track of offenses	Raise issue and report if necessary

Table 3: Risk Management

D GANTT CHART

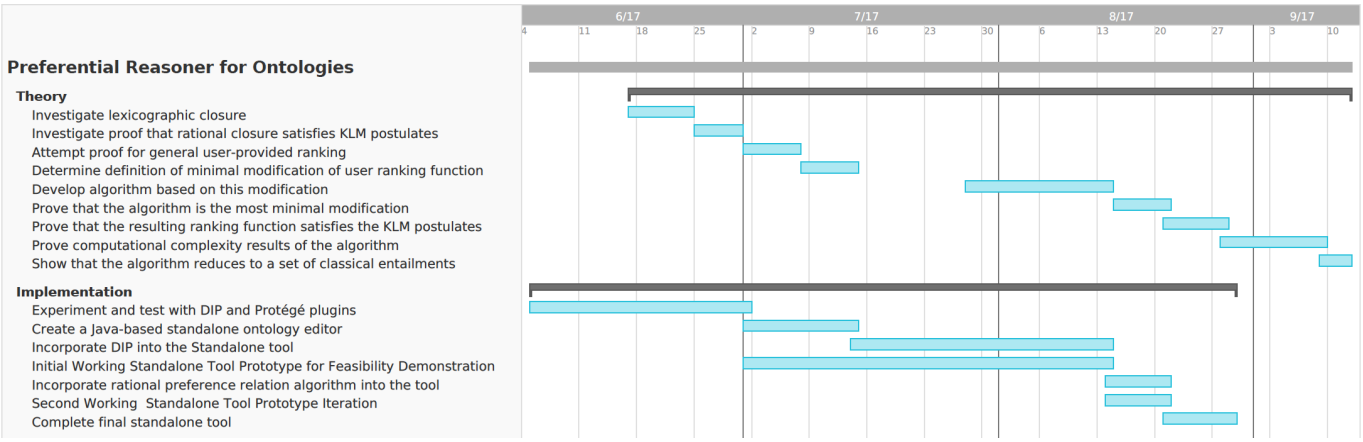

Figure 1: Project Timeline

REFERENCES

[1] Franz Baader, Diego Calvanese, Deborah L McGuinness, Daniele Nardi, and Peter F Patel-Schneider. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Vol. 32. 622 pages.

[2] Franz Baader and Bernhard Hollunder. 1995. Embedding defaults into terminological knowledge representation formalisms. *Journal of Automated Reasoning* 14, 1 (1995), 149–180.

[3] Piero A Bonatti, Carsten Lutz, and Frank Wolter. 2006. Description Logics with Circumscription.. In *KR*. 400–410.

[4] Katarina Britz, Thomas Meyer, and Ivan Varzinczak. 2011. Semantic foundation for preferential description logics. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 7106 LNAI. 491–500.

[5] Giovanni Casini, Thomas Meyer, Kody Moodley, Uli Sattler, and Ivan Varzinczak. 2015. Introducing Defeasibility into OWL Ontologies. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 9367. 409–426.

[6] Giovanni Casini, Thomas Meyer, Kody Moodley, and Ivan Varzinczak. 2013. Towards Practical Defeasible Reasoning for Description Logics. In *Proceedings of the 26th International Workshop on Description Logics*. 587–599.

[7] Giovanni Casini and Umberto Straccia. 2010. Rational Closure for Defeasible Description Logics. In *European Workshop on Logics in Artificial Intelligence*. Springer, 77–90.

[8] Giovanni Casini and Umberto Straccia. 2012. Lexicographic closure for defeasible description logics. In *Proc. of Australasian Ontology Workshop*, Vol. 969. 28–39.

[9] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. 2014. HermiT: An OWL 2 Reasoner. *Journal of Automated Reasoning* 53, 3 (2014), 245–269. https://doi.org/10.1007/s10817-014-9305-1

[10] Matthew Horridge and Sean Bechhofer. 2011. The OWL API: A Java API for OWL ontologies. *Semantic Web* 2, 1 (2011), 11–21.

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

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

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

[14] John McCarthy. 1980. Circumscription âĂŤ a form of non-monotonic reasoning. *Artificial intelligence* 13, 1 (1980), 27–39.

[15] Kody Moodley. 2015. *Practical Reasoning for Defeasible Description Logics*. Ph.D. Dissertation. University of KwaZulu-Natal.

[16] Kody Moodley, Thomas Meyer, and Ivan Varzinczak. 2012. A Defeasible Reasoning Approach for Description Logic Ontologies. In *South African Institute for Computer Scientists and Information Technologists Conference, SAICSIT 2012*. 69–78.

[17] Kody Moodley, Thomas Meyer, and Ivan Varzinczak. 2012. A Protege Plug-in for Defeasible Reasoning. In *Proceedings of the 25th International Workshop on Description Logics*. 486–496.

[18] Donald Nute. 1993. Defeasible logic. *Handbook of Logic in Artificial Intelligence and Logic Programming* 3 (1993).

[19] Raymond Reiter. 1980. A logic for default reasoning. *Artificial intelligence* 13, 1-2 (1980), 81–132.

[20] Manfred Schmidt-Schauß and Gert Smolka. 1991. Attributive concept descriptions with complements. *Artificial intelligence* 48, 1 (1991), 1–26.

[21] W3C OWL Working Group. 2012. OWL 2 Web Ontology Language Document Overview. *OWL 2 Web Ontology Language* December (2012), 1–7.

[22] Kewen Wang, David Billington, Jeff Blee, and Grigoris Antoniou. 2004. Combining description logic and defeasible logic for the semantic web. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web*. Springer, 170–181.