On the Derivability of Defeasible Logic

Ho Pun Lam

A thesis submitted for the degree of Doctor of Philosophy at The University of Queensland in April 2012

School of Information Technology and Electrical Engineering
Declaration by author

This thesis is composed of my original work, and contains no material previously published or written by another person except where due reference has been made in the text. I have clearly stated the contribution by others to jointly-authored works that I have included in my thesis. I have clearly stated the contribution of others to my thesis as a whole, including statistical assistance, survey design, data analysis, significant technical procedures, professional editorial advice, and any other original research work used or reported in my thesis. The content of my thesis is the result of work I have carried out since the commencement of my research higher degree candidature and does not include a substantial part of work that has been submitted to qualify for the award of any other degree or diploma in any university or other tertiary institution. I have clearly stated which parts of my thesis, if any, have been submitted to qualify for another award.

I acknowledge that an electronic copy of my thesis must be lodged with the University Library and, subject to the General Award Rules of The University of Queensland, immediately made available for research and study in accordance with the Copyright Act 1968.

I acknowledge that copyright of all material contained in my thesis resides with the copyright holder(s) of that material.

Statement of Contributions to Jointly Authored Works Contained in the Thesis

This thesis contains the following joint works:

- [145]. I was responsible for the system design, implementation and drafting the paper.

Statement of Contributions by Others to the Thesis as a Whole

No contributions by others.

Statement of Parts of the Thesis Submitted to Qualify for the Award of Another Degree

None
Published Works by the Author Incorporated into the Thesis

- [143, 144] *Partially incorporated as sections in Chapter 3*
- [142, 141] *Partially incorporated as sections in Chapter 4*
- [146, 145] *Partially incorporated as sections in Chapter 5*

Additional Published Works by the Author Relevant to the Thesis but not Forming Part of it

- [75]
- [140]
- [203]
Acknowledgements

My heartfelt gratitude goes to my supervisors Dr. Renato Iannella and Dr. Guido Governatori, who provided me with guidance, insights and their constant support. Both were always there, ready to discuss about any questions and helped/guided me to fix the problems. For the four years since I commence my PhD Study, they have been my role model for how to do rigorous research. Their continuous support and our discussions during the last few years inspired me in many aspects. It is my fortune to have them be my supervisors in my life. Their generous support had given me the peace of mind that was much needed in my graduate study. Without their help, this work would not be possible.

I would also like to thank members of my thesis committee, especially Dr. Antonis Bikakis and Prof. Dr. rer. nat. Adrian Paschke, for their time on reviewing the thesis, and for the rich, accurate and valuable technical feedback they have provided.

My special thanks to Prof. Ken Satoh, who hosted me in his research laboratory during the time when I was at the National Institute of Informatics, Tokyo, Japan, undertaking research studies in the area of speculative computation.

I also wish to thank members of the Data and Knowledge Engineering (DKE) research group. Their weekly group seminar (and discussions) gave me many valuable ideas and insights.

I gratefully acknowledge all of my friends and colleagues at National ICT Australia Queensland Research Laboratory (NICTA - QRL) for the open and friendly environment in which it was really a pleasure to work in. In particular, I would like to thank Babara Duncan, Mark Medosh and Sarah Sims for providing me their administrative support, and Subhasis Thakur for the valuable discussions on the applications of Defeasible Logic.

I would like to express a final acknowledgment to all those that throughout my life, at different levels and by different means, had contributed to shape my personality and help me to be the person I am. Thank you all!

I dedicate this thesis to my parents.
Abstract

In this thesis we focus upon the problem of derivability of rules from Defeasible Logic. Derivability of rules, as defined here, comprises the concept of an extension from a defeasible theory [167] as well as the classical notion of derivability of rules in logic. The idea of localness of reasoning, reasoning with a limited access to rules, is realized by the concept of relative derivability [85]. Starting with the derivability of rules, we next touch upon the questions of the activation of rules and (in)consistency of rules in defeasible logic. We will discuss the problems of computing extensions of defeasible logic under different intuitions, with particular interest in ambiguity blocking, ambiguity propagation, well-founded semantics, and their combinations, and will present algorithms to these problems. In addition, we will also discuss the deficiency of the current inference process and will present a new theorem and an algorithm to enhance the process. Matters related to the implementations issues of the algorithms and experimental results will also be discussed in this thesis.

Next we will discuss the problem of ambient intelligence: the imperfect nature of context, the dynamic nature of ambient environment, and special characteristics of the devices involved, which imposed new challenges in the field of distributed Artificial Intelligence. This thesis proposes a solution based on Multi-Context paradigm with non-monotonic features. We will model ambient agents as logic-based entities and present a framework based on the argumentation semantics of defeasible logic. We also extends the semantics of modal defeasible logic in handling violations and preferences, and can derive the conclusion with the avoidance of transformations. We also present an operational model in the form of distributed reasoning algorithm for query processing with iterative revision based on speculative computation. Issues such as inconsistent arguments handling and agents context preferences will also be discussed.

Last but not least, a prototypical implementation showcasing the wealth of our approach will also be presented at the end of the thesis.

Keywords: defeasible logic, non-monotonic reasoning, ambiguity propagation, well-founded semantics, ambient intelligence, artificial intelligence

Australian and New Zealand Standard Research Classifications (ANZSRC):

- 080101 - Adaptive Agents and Intelligent Robotics: 30%.
- 080203 - Computational Logic and Formal Languages: 70%.
# Contents

Declaration .................................................. i  
Acknowledgements ........................................ iii  
Abstract .................................................. v  
Table of Contents ......................................... vii  
List of Figures ............................................ xi  
List of Tables .............................................. xiii  
List of Algorithms ......................................... xiv

1 Introduction ............................................ 1  
1.1 General issues ....................................... 1  
  1.1.1 Limitation of current reasoning approach .......... 2  
  1.1.2 Reasoning with Distributed Context ............... 4  
1.2 Research aims ....................................... 5  
1.3 Thesis outline ....................................... 7  
1.4 Bibliography notes ................................... 7

2 Background ............................................. 9  
2.1 Preliminaries on Defeasible Logic ................... 10  
  2.1.1 An Informal Presentation ......................... 10  
  2.1.2 Formal Definition ................................ 11  
  2.1.3 Proof Theory .................................... 12  
  2.1.4 Bottom-Up Characterization of DL ............... 14  
  2.1.5 Strong Negation Principle ....................... 15  
  2.1.6 Coherence and Consistency ...................... 16  
2.2 Variants of Defeasible Logic ....................... 17  
  2.2.1 Removing Team Defeat ........................... 18  
  2.2.2 Ambiguity Blocking and Ambiguity Propagation ... 18  
  2.2.3 Well-Founded Semantics ......................... 21  
  2.2.4 Combination ................................... 22  
2.3 Argumentation Semantics ............................ 23  
  2.3.1 Conflicting Arguments: Attack and Undercut ...... 25  
  2.3.2 The Status of Arguments ....................... 26  
2.4 Extending Defeasible Logic with Modal Operators ... 27
2.5 Modal Defeasible Logic with Interactions .......................... 32
  2.5.1 Conflicts ............................................. 32
  2.5.2 Conversions ............................................ 33
2.6 Existing Reasoners for Defeasible Logic ............................. 34
  2.6.1 d-Prolog ............................................... 35
  2.6.2 Deimos and Delores ..................................... 35
  2.6.3 DR-Prolog .............................................. 37
  2.6.4 DR-DEVICE ........................................... 37
2.7 Discussion .................................................. 38
  2.7.1 Comparison of DL with other approaches ...................... 39
  2.7.2 Features of DL .......................................... 39
    2.7.2.1 Provability ........................................ 39
    2.7.2.2 Preference Handling ................................ 41
    2.7.2.3 Reasoning algorithm ................................ 42
3 Inferencing with Defeasible Theory .................................. 45
  3.1 Background ............................................... 45
  3.2 The Linear Algorithm ...................................... 46
    3.2.1 Problems of present approach ........................ 46
    3.2.2 Transformations ...................................... 47
  3.3 Inferiorly defeated rules ................................... 51
  3.4 New Inference algorithm ................................... 52
  3.5 Inferencing Defeasible Theory under different variants ....... 56
    3.5.1 Ambiguity Propagation ................................ 56
    3.5.2 Well-Founded Semantics ............................... 61
  3.6 Summary .................................................. 67
4 SPINdle Defeasible Reasoner ......................................... 69
  4.1 SPINdle Architecture ....................................... 71
    4.1.1 Base Components ...................................... 72
      4.1.1.1 I/O Manager ...................................... 73
      4.1.1.2 Theory modeling .................................. 76
    4.1.2 Reasoning Components .................................. 77
      4.1.2.1 Theory Normalizers ................................ 78
      4.1.2.2 Inference Engines ................................. 81
    4.1.3 Extending SPINdle ..................................... 83
  4.2 Performance Evaluation ...................................... 85
    4.2.1 Design of the Experiments ............................. 85
    4.2.2 Configuration for the Experiments ...................... 86
    4.2.3 Experimental Results .................................. 87
  4.3 Summary .................................................. 92
# List of Figures

1.1 Defeasible theory inferencing process ........................................... 2
1.2 Inheritance network of Example 1.1 (Presumption of Innocence) .......... 3

2.1 Inheritance network of defeasible theory in Example 2.1 .................... 19
2.2 Inheritance network of defeasible theory in Example 2.1 under ambiguity blocking and ambiguity propagation. .................................................. 20

3.1 Inheritance network of theory in Example 3.3 ................................. 57
3.2 Results of theory in Example 3.3 under different variants ................. 58

4.1 SPINdle Theory Editor ............................................................... 70
4.2 SPINdle System Architecture ...................................................... 71
4.3 Sample theory of Example 4.1 in XML format ............................... 75
4.4 Data structure for literals-rules association .................................... 77
4.5 Modified defeasible theory inference process ................................... 81
4.6 Reasoning time and memory usage of theory “chain” under different approaches 88
4.7 Reasoning time and memory usage of theory “circle” under different approaches 88
4.8 Reasoning time and memory usage of theory “dag” under different approaches 89
4.9 Reasoning time and memory usage of theory “trees” under different approaches 89
4.10 Reasoning time and memory usage of theory “teams” under different approaches 89
4.11 Reasoning time and memory usage of theory “supTest” under different approaches 90
4.12 Reasoning time for Theories with Undisputed Inferences ...................... 91
4.13 Reasoning time for Theories with Disputed Inferences ....................... 92

5.1 Graphical representation of the context-aware mobile phone scenario .... 99
5.2 City Map model ............................................................................ 102
5.3 City with UAVs ............................................................................ 102
5.4 Behavior control under an perception-action cycle ............................ 103
5.5 A magic box ................................................................................. 105
5.6 Contextual Reasoning model ......................................................... 106
5.7 Ordinary computation ................................................................... 108
5.8 Speculative Computation (simplified) ............................................ 109
5.9 Arguments contained in \( \text{Args}_c \) (Example 5.2) .......................... 128
5.10 Conspire architecture ................................................................. 139
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.11</td>
<td>Message flow between agents</td>
<td>140</td>
</tr>
<tr>
<td>5.12</td>
<td>User agent: mobile phone</td>
<td>141</td>
</tr>
<tr>
<td>5.13</td>
<td>Message flow between agents (conflict message)</td>
<td>142</td>
</tr>
<tr>
<td>5.14</td>
<td>User agent: mobile phone (conflict message)</td>
<td>142</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Strong Negation formula ........................................ 16
4.1 Size of test theories ........................................... 86
4.2 Reasoning time used in UAV navigation theory ............... 92
C.1 Rule Type-Symbol association list ................................ XIV
E.1 Reasoning time for Theories with Undisputed Inferences .... XIX
E.2 Reasoning time for Theories with Disputed Inferences ....... XX
# List of Algorithms

<table>
<thead>
<tr>
<th>Section</th>
<th>Algorithm Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Inference Algorithm for $+\vartheta$ and $-\vartheta$</td>
<td>54</td>
</tr>
<tr>
<td>3.2</td>
<td>Inference Algorithm for $+\vartheta_{ap}$ and $-\vartheta_{ap}$</td>
<td>60</td>
</tr>
<tr>
<td>3.3</td>
<td>Inference Algorithm for Well-Founded Defeasible Logic ($\vartheta$)</td>
<td>65</td>
</tr>
<tr>
<td>3.4</td>
<td>Computation of $\vartheta$-unfound set</td>
<td>65</td>
</tr>
<tr>
<td>3.5</td>
<td>Strongly Connected Literals computation</td>
<td>66</td>
</tr>
<tr>
<td>5.1</td>
<td>ComputeDefinite</td>
<td>118</td>
</tr>
<tr>
<td>5.2</td>
<td>Discard</td>
<td>120</td>
</tr>
<tr>
<td>5.3</td>
<td>ComputeDefeasible</td>
<td>121</td>
</tr>
<tr>
<td>5.4</td>
<td>Process reduction phase: New query arrival</td>
<td>135</td>
</tr>
<tr>
<td>5.5</td>
<td>Process reduction phase: Iterative step</td>
<td>136</td>
</tr>
<tr>
<td>5.6</td>
<td>Answer Arrival Phase</td>
<td>137</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Defeasible logic (DL) is a sceptical non-monotonic formalism originally proposed by Nute [167] and belongs to a family of approaches that are based on the idea of non-monotonicity without negation as failure or logic, such as Courteous Logic [110] and LPwNF (Logic Programming with Negation as Failure) [66]. DL is a simple rule-based reasoning approach to reason with incomplete and contradictory information while preserving low computational complexity [151], and is suitable to model situations where conflicting rules may appear simultaneously. In [15, 16], its flexibility has been advocated by proposing simple ways to tune it, which thus generated different variants to capture various intuitions.

Over the years, DLs have attracted considerable interest. Theoretically, they have been studied deeply in terms of proof theory [18], proof theoretic [152, 109], semantics [90, 101], its relations to other logic programming approach with negation as failure [17]; and has been extended to capture the temporal aspects of several specific phenomena, such as legal positions [103], norm dynamics [106, 97] and deadlines [105]. Recently, [41] investigated the relationships among several variants of DL, capturing the intuitions of different reasoning issues, such as ambiguity blocking and ambiguity propagation, and team defeat. In addition, its use has been advocated in various application domains, such as modeling of regulations and business rules [14], legal reasoning [97, 103, 176], agent modeling and agent negotiation [69, 94, 92], modeling of contracts [112, 88], actions and planning [78, 72, 61, 62] and applications to the Semantic Web [137, 10, 23].

1.1 General issues

There are five kinds of features in DL: fact, strict rule, defeasible rules, defeaters and superiority relations among rules. Essentially defeater is used to prevent some conclusions from being drawn; while superiority relation provides information about the relative strength of rules, i.e., it provides information about which rules can overrule other rules when both are applicable. A program or knowledge base that consists of these items is called a defeasible theory.

Typically, the process of deriving conclusions from a defeasible theory includes the use of a
series of pre-processing transformations that transform the theory into normal form [9] – an equivalent theory without superiority relations and defeaters, and then applies the reasoning algorithm [151] to the transformed theory (Figure 1.1).

It is, in general, accepted that such transformation-based approach can have many benefits. First, the transformed theory supports the understanding and assimilation of new concepts because they allow one to concentrate on certain forms and key features only. Secondly, it can be instrumental in the implementation of logic since assuming that a theory has a specific form can actually facilitate and simplify the development of algorithms, which is what the reasoning algorithm in [151] is based on.

It is also clear that these transformations were designed to provide incremental transformations to the theory and would systematically introduce new literals and rules to emulate the features removed. That is, the transformations can be performed on a bit-by-bit basis and an update in the original theory should have cost proportional to the change, without the need to transform the entire updated theory anew [18], which is an important properties for implementations.

1.1.1 Limitation of current reasoning approach

The treatment of ambiguity also differentiates DL from other non-monotonic formalisms. Technically speaking, in non-monotonic reasoning, a literal \( p \) is ambiguous if there is a chain of monotonic reasoning pro \( p \) while another con \( p \) (or alternatively pro \( \sim p \)). Consider the example below:

**Example 1.1** (Presumption of Innocence [89]). Let us suppose that a piece of evidence \( A \) suggests that the defendant in a legal case is not responsible while a second piece of evidence \( B \) indicates that he/she is responsible; and the sources are equally reliable. According to the underlying legal system a defendant is presumed innocent (i.e., not guilty) unless responsibility
has been proved (without any reasonable doubt).

Given both evidences $A$ and $B$, ambiguity exists as it is unclear to us on whether the defendant should hold any responsibility to the case or not. Consequently, under this situation, with the underlying legal system, we should conclude that the defendant is not guilty.

However, if we allow propagation of the ambiguity of responsibility to the next level of the reasoning chain, then the defendant’s guiltiness becomes ambiguous; hence an undisputed conclusion cannot be drawn. (Figure 1.2 shows the inheritance network of this example.)

Indeed, variants have been defined to captured both intuitions in DL$^1$. In the first case we speak of *ambiguity blocking*, in the latter case of *ambiguity propagation*. In an ambiguity blocking setting, given the sceptical nature of the reasoning, conclusions of conflicting literals are considered both as not provable, and we will ignore the reasons why they were when we use them as premises of further arguments. On the other hand, in an ambiguity propagation setting, since we were not able to resolve the conflict, the ambiguity will then propagate to conclusions depending on the ambiguous literals. (Details of the ambiguity blocking and the ambiguity propagation variant of DL will be presented in Chapter 2)

However, as pointed out in [151], although the aforementioned transformations can be applied in one pass, they are profligate in their introduction of propositions and generation of rules, which would result in an increase in theory size by at most a factor of 12 [48, 154]. Besides, even though the inference process mentioned above works absolutely well in the ambiguity blocking variant of DL, it is not the case in other other variants. As it will be described in Chapter 3, there are situations where important representation properties of DL cannot be preserved during the transformation process, which subsequently affects the inference process such that improper conclusions may be drawn.

---

$^1$Several variants (such as *ambiguity propagation*, *well-founded semantics*) of DL have been proposed to capture the intuitions of different non-monotonic reasoning formalism. Readers interested please refer to [15, 16] for details.
1.1.2 Reasoning with Distributed Context

The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it. [219]

Computing is moving towards pervasive in which devices, software agents, and services are expected to integrate and cooperate seamlessly in support of human objectives, anticipating needs and provide appropriate services and/or information to the intended person in the right context, at the right time, for the right purpose and with the right format. It constitutes, albeit invisible, part of our physical environment.

With this in mind, Pervasive Computing and Ambient Intelligence are considered as an emerging discipline for research in the future development and use of ICT. As described in [1], Ambient Intelligence has been defined as the field to study and create embodiments for smart environments that not only react to human events through sensing, interpretation and service provision, but also learn and adapt their operation and services to the users over time. These embodiments employ contextual information when available, and offer unobtrusive and intuitive interfaces to their users. Through a user-oriented employment of communication links, these systems can also offer ambient communication and media delivery options between users allowing for seamless multi-party interactions and novel social networking applications.

One important aspect of Ambient Intelligence is the level of interactivity between users and the devices. As discussed in [21], on one hand there is a motivation to reduce explicit human-computer interaction as it is supposing that the system should be able to use its intelligence to infer the situations and user needs from the observed activities. On the other hand, a diversity of users may need or seek direct interaction with the system to indicate their preferences, needs, etc. Moreover, it is expected that the system should be able to provide intelligence access to relevant knowledge or proper services to users, in a sense that users may not be interested to know what type of technologies or what kind of sensors are used behind the scene.

Hence, an Ambient Intelligence system needs to demonstrate high levels of performance related to complex intellectual task, which require a deeper understanding of the nature and limits of computations [125].

To reason about the context, we have to exploit the true meaning of raw context data; to process, combine and translate the low level sensors data into valuable information, based on which the system can determine the states of the context, and then react to the changes accordingly [31]. However, the highly dynamic and imperfect nature of the environment, and the special characteristics of the entities that operate in the environment, have introduced lots of challenges in this task.

So far, the reasoning approaches proposed either neglect to address the problem of imperfect nature of context, or handle them by building additional reasoning mechanism on top of logic models that cannot deal with the problem of uncertainty, ambiguity and inconsistency, which
may inherit from the environment [30]. Moreover, most frameworks developed have been based on fully centralized architectures for context management which require the existence of a central entity, to transform the imported context data into a common format and deduce the higher-level context information from the raw context data.

To this, DL was designed with an ambition to provide reasoning support with incomplete and contradictory information with low computational complexity. However, the majority of current practical or theoretical research in DL has been put into providing reasoning service in a local manner. Little has been done on using DL to provide reasoning support in a dynamic and/or distributed environment.

Therefore, due to the deficiencies stated above and the inability of current inferencing process, we believe that more effort should be made to improve the inferencing process of DL, such that theory transformation(s) should only be done whenever necessary, and to devise efficient methods/algorithms in computing the extensions of defeasible theories. In addition, with the strength that comes with DL in handling incomplete and contradictory information, we believe a more thoughtful discuss and theoretical development of Ambient Intelligence based on DL should benefit the logic society at large.

1.2 Research aims

Broadly speaking, this research concerns the derivability and applicability of defeasible logic. Our research questions are:

- How to improve the current reasoning algorithm so as to optimize/reduce the computational complexity of the inference process;
- How to derive conclusions of defeasible theories under different variants – with special focus on the design of algorithms for computing conclusions of DL in ambiguity blocking, ambiguity propagation variant, well-founded semantics, and their combinations; and
- How defeasible logic can be used to represent and reason about the state of agents in an ambient environment.

The main focus of this thesis is two fold. The first aim is to study the derivability of defeasible logic in a local sense. We will study algorithms that can be used in inferencing from a defeasible theory under different variants and the ways to improve the deficiency of the current inferencing algorithms in deriving the defeasible conclusions; while the second aim is to study the derivability of defeasible theory in a totally distributed ambient environment. So, in other words, the research objective of this thesis is mainly on the computational aspect of inferencing with defeasible theory, under different situations.

We have devised a linear-time algorithm to compute the extension of the ambiguity propagation variant and well-founded variants of DL. For the well-founded variant we have established
a way to compute the unfounded set of a defeasible theory. In addition, by combining the algorithms together, we can also handle the well-founded variants of ambiguity blocking and ambiguity propagation in polynomial time.

In addition, through defining the notion of *Inferiorly Defeated Rules* and its properties in defeasible theory, we have simplified the reasoning algorithm stated in the previous section such that the removal of superiority relations in a defeasible theory is no longer necessary. The new algorithm that we defined can not only reduce the size of theory size increase due to transformation to a maximum factor of 4 so that the reasoning process can be improved substantially, but can also preserve the representational properties of defeasible theory in all variants of DL such that the algorithms that we devised in computing the ambiguity propagation and well-founded semantics can be used directly, without any modifications.

On the other hand, the study of ambient intelligent and pervasive computing has introduced lots of research challenges in the field of distributed artificial intelligence during the past few years. These are mainly caused by the imperfect nature of the environment and the special characteristics of the entities that process and share the context information available [31].

Ambient agents that work in such environment can be developed independently and are autonomous in nature, and can always be in total control of their own resources (*local resources*). They are expected to have their own goals, perception and capabilities, and can collaborate with each other to achieve their common goals. They are depending on each other for querying information, computing resources, forwarding requests, etc [163]. When a change in the world is detected, they may need to update/revise their own local knowledge, which may subsequently affect the answers computed for a received query.

To this end, a framework of *Distributed Defeasible Speculative Reasoning* (DDSR) has been proposed and developed to reason about agents among multiple groups. In our framework, each agent is equivalent in nature and can have its own local knowledge and perception about the current environment. Agents can receive requests from users or other agents and respond to them based on the information that they aggregated. Besides, agents in the framework can also collaborate with (or query on) each other to achieve their common goals.

Through using the DDSR algorithm, uncertain or incomplete information will first be substituted by some *default answers* such that the inquired agent can start the reasoning process in advance while waiting for the response from other agents. When the answers are available (either new or revised), the agent then revise its reasoning process by either continuing the current computations or starting a new computation entailed by the new answer beyond defaults. An example showing how DDSR algorithm can be integrated with a context-aware mobile phone will be presented at the end of the thesis.

It is important to note that the discussions in this thesis will be restricted to essentially propositional Defeasible Logic [18], and do not take in account function symbols. Rules with free variables are interpreted as the set of their ground instances; in such cases the Herbrand universe is assumed to be finite. It is also assumed that the reader is familiar with the notation
and basic notions of propositional logic. That is, if $q$ is a literal, $\sim q$ denotes the complementary literal (if $q$ is a positive literal $p$ then $\sim q$ is $\neg p$; and if $q$ is $\neg p$, then $\sim q$ is $p$).

### 1.3 Thesis outline

The thesis is structured into seven chapters.

Chapter 2 gives a general overview of defeasible logic and some of its extensions that we can use in capturing different intuitions of the problems. An extension that incorporate the use of modal operator in defeasible logic (Modal Defeasible Logic) will also be discussed at the end of this chapter.

Chapter 3 presents the algorithms we devised for computing the extensions of ambiguity propagation and well-founded semantics variants of defeasible logic. The notion of Inferiorly Defeated Rules, its properties, and the new algorithm for deriving the conclusions of a defeasible theory will also be discussed here.

Chapter 4 presents the architecture of the defeasible reasoner, SPINdle, we developed based on the algorithms devised in Chapter 3, and the experimental results that we obtained when we compared it to other algorithms.

Chapter 5 presents the issues arising when capturing information in an ambient environment, and introduces our Distributed Defeasible Speculative Reasoning framework. An extension of modal defeasible logic and its application in modeling agent belief will also be discussed.

Chapter 6 concludes the thesis with a summary of the main contributions and a discussion of future works.

### 1.4 Bibliography notes

Most of the research presented in this thesis has been published in some form. The main result of Chapter 3 on deriving the conclusions of defeasible theory under ambiguity propagation variants and well-found semantics has been accepted at the 4th International Web Rule Symposium: Research Based and Industry Focused (RuleML 2010); and the new inferencing algorithm computing extensions of defeasible theory has been accepted at the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-11).

The UVA navigation scenario and the modal defeasible logic extension presented in Chapter 5 has first been accepted as a conference paper at the 3rd International RuleML-2009 Challenge, and later as a journal paper in the Journal of Logic and Computation.

The defeasible logic reasoner (current version 2.0.5) presented in Chapter 4 has also been accepted at the 2009 International Symposium on Rule Interchange and Applications (RuleML 2009), and is now open sourced\(^2\) to all researchers/students who are interested in studying the

\(^2\)SPINdle: http://spin.nicta.org.au/spindle
semantics/behavior of defeasible logic, or software developers who already have some knowledge in defeasible logic and wish to incorporate defeasible logic formalism into their applications.
Chapter 2

Background

Logics for knowledge representation and, in particular, non-monotonic logic have developed greatly over the past 20 years. Many logics have been proposed, and a deeper understanding of the pros and cons of particular logics has been accumulated. Despite some indications have already shown that these logics can be usefully applied to solve our daily problems [176, 165], most non-monotonic reasoning systems have high computational complexity which seems to be contrary to the original motivation of “jumping to conclusions” [154].

Moreover, it appears that there is no single logic that is appropriate in all situations or for all purposes. History already indicated that while one logic may achieve desired outcomes in some situations, it may not be the case in the others. This is, no doubt, one reason for the proliferation of non-monotonic reasoning.

Furthermore, even with the same language and a common motivating intuition, reasonable people can disagree on the semantics of the logic. This can be seen, for example, in the literature of logic program with negation. In [207] this point was made more sharply where a “clash of intuitions” was demonstrated in several different ways for a simple language describing multiple inheritance with exceptions. So, it appears no single logic, with a fixed semantics, will be appropriate.

One way to overcome this problem is to formalize logics that are “tunable” to the environment. That is, to develop a framework of logics in which an appropriate logic can be designed. In fact, families of approaches have emerged around the classical non-monotonic systems of circumscription [158] and default logic [182]. Besides, given the framework, we also have to develop a methodology for designing logics, and the capabilities of employing more than one such logic in a representation.

In the light of this, Defeasible Logic (DL), as a non-monotonic formalism designed in preparation to sacrifice expressive power in favor of simplicity, efficiency, and easy implementability, has proven able to deal with many different intuitions of non-monotonic reasoning [16]. In this chapter, we present the essential concepts of DL, including its language, proof conditions, strong negation principle, argumentation semantics, some of its variants (that can be tuned to capture different intuitions according to the different context), as well as the modal extension. We then present and discuss several implementations of DL reasoners, followed by a discussion.
2.1 Preliminaries on Defeasible Logic

2.1.1 An Informal Presentation

As usual with non-monotonic reasoning, we have to specify (1) how to represent a knowledge base, and (2) the inference mechanism used to reason with the knowledge base.

Accordingly a defeasible theory $D$ is a triple $(F, R, >)$ where $F$ and $R$ are finite set of facts and rules respectively, and $>$ is an acyclic superiority relation on $R$. The language of DL consists of a finite set of literals, where a literal is either an atomic proposition or its negation. Given a literal $l$, $\sim l$ denotes its complement. That is, if $l = p$ then $\sim l = \neg p$, and if $l = \neg p$ then $\sim l = p$. Example of facts and rules below are standard in the literature of the field.

Facts are logical statements describing indisputable facts, represented either in form of states of affairs (literal or modal literal (cf. section 2.4)) or actions that have been performed, and are considered to be always true. For example, “John is a human” is represented by: $human(John)$.

A rule $r$, on the other hand, describes the relations between a set of literals (the antecedent $A(r)$, which can be empty) and a literal (the consequence $C(r)$). We can specify the strength of the rule relation using the three kinds of rules supported by DL, namely: strict, defeasible, and defeater.

Strict rules are rules in the classical sense: whenever the premises are indisputable (e.g. a fact) then so is the conclusion. For example,

$$human(X) \rightarrow mammal(X)$$

which means “Every human is a mammal”.

It is worth to mention that strict rules with empty antecedents can be interpreted the same way as facts. However, in practice, facts are more likely to be used to describe contextual information; while rules, on the other hand, are more likely to be used to represent the reasoning underlying the context.

Defeasible rules are rules that can be defeated by contrary evidence. For example, typically mammal cannot fly, written formally:

$$mammal(X) \Rightarrow \neg flies(X)$$

The idea is that if we know that $X$ is a mammal then we may conclude that it cannot fly unless there is other, not defeated, evidence suggesting that it may fly (for example that the mammal is a bat). Defeasible rule with empty antecedent can be considered as a presumption.

Defeaters are rules that cannot be used to draw any conclusions. Their only use is to prevent some conclusions, i.e., to defeat some defeasible rules by producing evidence to the contrary. For example the rule:

$$heavy(X) \sim \neg flies(X)$$

states that an animal is heavy is not sufficient enough to conclude that it does not fly. It is
only evidence against the conclusion that a heavy animal flies. In other words, we don’t wish to conclude that \( \neg \text{flies} \) if \text{heavy}, we simply want to prevent a conclusion \text{flies}.

In all its variants, DL is “directly sceptical” non-monotonic logic meaning that it does not support contradictory conclusions. Instead DL seeks to resolve conflicts. In case there is a combination of reasoning chains leading to a contradiction, i.e., there is support for concluding \( A \) but also support for concluding \( \neg A \), DL does not conclude either of them. However, if the support for \( A \) has priority over the support for \( \neg A \) then \( A \) is concluded.

In the light of this, a \textit{superiority relation} on \( R \) is used to define priorities among rules, i.e., where one rule may override the conclusion of another rule when both apply. When \( r_1 > r_2 \) then \( r_1 \) is called \textit{superior} to \( r_2 \), and \( r_2 \) \textit{inferior} to \( r_1 \). This expresses that the conclusion derived by \( r_1 \) may override the conclusion derived by \( r_2 \).

For example, given the following facts:

\[
\begin{align*}
\text{bird}(X) \\
\text{brokenWing}(X)
\end{align*}
\]

and defeasible rules:

\[
\begin{align*}
r: & \quad \text{bird}(X) \Rightarrow \text{flies}(X) \\
r': & \quad \text{brokenWing}(X) \Rightarrow \neg \text{flies}(X)
\end{align*}
\]

which contradict one another if “Beauty” is a bird with a broken wing, they do not if we state that \( r' > r \), with the intended meaning that \( r' \) is strictly stronger than \( r \), then we can indeed conclude that “Beauty” cannot fly.

Notice from the example above that cycle in the superiority relation is counterintuitive. That is, it makes no sense to have both \( r' > r \) and \( r > r' \). Thus, this thesis will only focus on cases where superiority relation is \textit{acyclic}.

Another point worth noting is that, in DL, priorities are \textit{local} in the following sense [18]: two rules are considered to be competing with one another only if they have complementary heads. Thus, since the superiority relation is used to resolve conflicts among competing rules, it is only used to compare rules with complementary heads; the information \( r' > r \) for rules \( r, r' \) without complementary heads may be part of the superiority relation, but has no effect on the proof theory.

\textbf{2.1.2 Formal Definition}

As stated in previous chapter, in this thesis, we consider only propositional version of DL, and do not take in account function symbol. Rules with free variable are interpreted as rule schemas, that is, as the set of all ground instances; in such cases the Herbrand universe is assumed to be finite.

Rules are defined over a \textit{language} \( \Sigma \), the set of propositions (atoms) and labels that may be used in the rule. In cases where it is unimportant to refer to the language of \( D \), \( \Sigma \) will not be mentioned.
Definition 2.1. A rule $r : A(r) \rightarrow C(r)$ consists of its unique label $r$, its antecedent $A(r)$ ($A(r)$ may be omitted if it is an empty set) which is a finite set of literals, an arrow $\rightarrow$ (which is a placeholder to be introduced later), and its head (or consequence) $C(r)$ which is a literal. In writing rules the set notation for antecedents and rule label can be omitted if it is unimportant or not relevant for the context.

The three different kinds of rules, each represented by a different arrow: strict rules use $\rightarrow$, defeasible rules use $\Rightarrow$, and defeaters use $\sim$.

Given a set of rules, we denote the set of all strict rules by $R_s$, the set of all defeasible rules by $R_d$, and the set of all strict and defeasible rules by $R_{sd}$; and name $R[q]$ the set of rules in $R$ with head $q$.

Definition 2.2. A superiority relation on $R$ is a relation $> \subseteq R \times R$. Where $r_1 > r_2$, then $r_1$ is superior to $r_2$, and $r_2$ is inferior to $r_1$. Intuitively, $r_1 > r_2$ expresses that $r_1$ overrules $r_2$, should both rules be applicable.

We assume $>$ to be acyclic (that is, the transitive closure of $>$ is irreflexive).

Consequently, and formally, we have the following definition.

Definition 2.3. A defeasible theory $D$ is a triple $(F,R,>)$ where $F$ is a finite set of literals (called facts), $R$ a finite set of rules, and $> \subseteq R \times R$ an acyclic superiority on $R$.

Definition 2.4 (Regular Form Defeasible Theory). [18] A defeasible theory $D = (F,R,>)$ is regular (or in regular form) iff the following three conditions are satisfied.

1. Every literals is defined either solely by strict rules, or by one strict rule and other non-strict rules.
2. No strict rule participates in the superiority relation $>$. 
3. $F = \emptyset$.

2.1.3 Proof Theory [18, 41]

Provability of DL is based on the concept of a derivation (or proof) in $D = (F,R,>)$. A derivation is a finite sequence $P = (P(1),\ldots,P(n))$ of tagged literals. A tagged literal consists of a sign (“+” denotes provability, “−” denotes finite failure to prove), a tag and a literal.

The tags are not part of the object language; intuitively, they indicate the “strength” of the conclusions they are attached to, and correspond to different classes of derivations. Initially we will consider two tags: $\Delta$ denotes definite provability based on monotonic proofs, and $\partial$ denotes defeasible provability based on non-monotonic proofs. The interpretation of the proof tags is as follows:
$+\Delta q$ meaning that $q$ is definitely provable in $D$ (i.e. using only facts and strict rules).

$-\Delta q$ meaning that $q$ is definitely rejected in $D$.

$+\partial q$ meaning that $q$ is defeasibly provable in $D$.

$-\partial q$ meaning that $q$ is defeasibly rejected in $D$.

There are different and sometimes incompatible intuitions behind what counts as a non-monotonic derivation. For a sequence of tagged literals to be a proof, it must satisfy certain conditions:

$+\Delta$) If $P(n + 1) = +\Delta q$ then either

(1) $q \in F$ or

(2) $\exists r \in R_s[q], \forall a \in A(r) : +\Delta a \in P(1..n)$.

That means, to prove $+\Delta q$ we need to establish a proof of $q$ using only facts and strict rules. This is a deduction in the classical sense - no proofs for the negation of $q$ need to be considered (in contrast to defeasible provability below where opposing chain s of reasoning must be taken into account). Thus it is a monotonic proof.

$-\Delta$) If $P(n + 1) = -\Delta q$ then

(1) $q \notin F$ and

(2) $\forall r \in R_s[q], \exists a \in A(r) : -\Delta a \in P(1..n)$.

To prove $-\Delta q$, i.e., that $q$ is not definitely provable, $q$ must not be a fact. In addition, we need to establish that every strict rule with head $q$ is known to be inapplicable. Thus in such rules there must at least one literal $l$ in the antecedent for which we have established that $l$ is not definitely provable ($-\Delta l$).

It is worth noticing that this definition of nonprovability does not involve loop detection. Thus if $D$ consists of the single rule $p \rightarrow p$, we can see that $p$ cannot be proven, but DL is unable to prove $-\Delta p$.

Defeasible provability requires considering the chain of reasoning for the complementary literal and possible resolution using superiority relation. Thus the inference rules for defeasible provability are bit more complicated than that of definite provability.

$+\partial$) If $P(n + 1) = +\partial q$ then either

(1) $+\Delta p \in P(1..n)$; or

(2) (2.1) $\exists r \in R_{sd}[q], \forall a \in A(r), +\partial a \in P(1..n)$, and

(2.2) $-\Delta \sim q \in P(1..n)$, and

(2.3) $\forall s \in R[\sim q]$ either

(2.3.1) $\exists a \in A(s), -\partial a \in P(1..n)$; or

(2.3.2) $\exists t \in R_{sd}[q]$ such that

$\forall a \in A(t), +\partial a \in P(1..n)$ and $t > s$.

$^1P(1..n)$ denotes the initial part of the sequence of length $n$. 
To show that \( q \) is defeasibly provable we have two choices: (1) we show that \( q \) is already definitely provable; or (2) we need to argue using the defeasible part of \( D \) as well. In particular, there must be a strict or defeasible rule with head \( q \) which can be applied (2.1). But now we need to consider possible “attack”, i.e., reasoning chains in support of \( \sim q \). To be more specific: to prove \( q \) defeasibly provable we must show that \( \sim q \) is not definitely provable (2.2). Also, (2.3) we must consider the set of all rules which are not known to be inapplicable and which have head \( \sim q \). Essentially, each such rule \( s \) attacks the conclusion \( q \). For \( q \) to be provable, each such rule \( s \) must be counterattacked by a rule \( t \) with head \( q \) with the following properties: (i) \( t \) must be applicable at this point, and (ii) \( t \) must be stronger than \( s \). Thus each attack on the conclusion \( q \) must be counterattacked by a stronger rule.

\[-\partial \) If \( P(n + 1) = -\partial q \) then
\[(1) \quad -\Delta q \in P(1..n), \text{ and}
\[(2) \quad (2.1) \forall r \in R_{sd}[q] \exists a \in A(r), -\partial a \in P(1..n); \text{ or}
\[(2.2) \quad +\Delta q \in P(1..n); \text{ or}
\[(2.3) \quad \exists s \in R[\sim q] \text{ such that}
\[(2.3.1) \quad \forall a \in A(s), +\partial a \in P(1..n), \text{ and}
\[(2.3.2) \quad \forall t \in R_{sd}[q] \text{ either}
\[\exists a \in A(t), -\partial a \in P[1..i]; \text{ or not}(t > s).
\]

Lastly, to prove that \( q \) is not defeasibly provable, we must first establish that it is not definitely provable. Then we must establish that it cannot be proven using defeasible part of the theory. There are three possibilities to achieve this: either (2.1) we have established that none of the (strict and defeasible) rules with head \( q \) can be applied; or (2.2) \( \sim q \) is definitely provable; or there must be an applicable rule \( s \) with head \( \sim q \) such that no applicable rule \( t \) with head \( q \) is superior to \( s \).

### 2.1.4 Bottom-Up Characterization of DL

In contrast to the proof theory which provides the basis for a top-down (backward-chaining) implementation of the logic, a bottom-up (forward-chaining) implementation provides us a bridge to decompose the logic into meta-program such that the structure of defeasible reasoning and a semantics for meta-language (logic programming) can be specified.

In the light of this, in [153], Maher and Governatori have characterized a defeasible theory \( D \) with an operator \( T_D \) which works on 4-tuples of sets of literals.
\( T_D(\Delta, -\Delta, +\partial, -\partial) = (+\Delta', -\Delta', +\partial', -\partial') \)

\[
\begin{align*}
+\Delta' &= F \cup \{q \mid \exists r \in R_s[q] \ A(r) \subseteq +\Delta\} \\
-\Delta' &= -\Delta \cup \{\{q \mid \forall r \in R_s[q] \ A(r) \cap -\Delta \neq \emptyset\} - F\} \\
+\partial &= +\Delta \cup \{q \mid \exists r \in R_{sd}[q] \ A(r) \subseteq +\partial, \text{ and} \}
&& \sim q \in -\Delta, \text{ and} \\
&& \forall s \in R[\sim q] \text{ either} \\
&& \quad A(s) \cap -\partial \neq \emptyset, \text{ or} \\
&& \quad \exists t \in R[q] \text{ such that } A(t) \subseteq +\partial \text{ and } t > s \} \\
-\partial &= \{q \in -\Delta \mid \forall r \in R_{sd}[q] \ A(r) \cap -\partial \neq \emptyset, \text{ or} \}
&& \sim q \in +\Delta, \text{ or} \\
&& \exists s \in R[\sim q] \text{ such that } A(s) \subseteq +\partial, \text{ and} \\
&& \forall t \in R[q] \text{ either} \\
&& \quad A(t) \cap -\partial \neq \emptyset, \text{ or} \\
&& \quad t \not> s \} 
\end{align*}
\]

The 4-tuples set of literals, also called an extension, forms a complete lattice under the pointwise containment ordering\(^2\), with \( \bot = (\emptyset, \emptyset, \emptyset, \emptyset) \) as its least element. The least upper bound operation is the pointwise union, which is represented by \( \cup \). It can be shown that \( T_D \) is monotonic and the Kleene sequence from \( \bot \) is increasing. Thus the limit \( F = (+\Delta_F, -\Delta_F, +\partial_F, -\partial_F) \) of all finite elements in the sequence exists, and \( T_D \) has a least fixpoint \( L = (+\Delta_L, -\Delta_L, +\partial_L, -\partial_L) \). When \( D \) is a finite propositional defeasible theory, then \( F = L \). Besides, the extension \( F \) captures exactly the inferences described in the proof theory.

**Theorem 2.5.** [153] Let \( D \) be a finite propositional defeasible theory and \( q \) a literal. Then:

- \( D \vdash +\Delta q \iff q \in +\Delta_F \)
- \( D \vdash -\Delta q \iff q \in -\Delta_F \)
- \( D \vdash +\partial q \iff q \in +\partial_F \)
- \( D \vdash -\partial q \iff q \in -\partial_F \)

The restriction of Theorem 2.5 to finite proposition defeasible theory derives from the formulation of proof theory; proofs are guaranteed to be finite under this semantics. However, the bottom-up semantics does not need this restriction.

### 2.1.5 Strong Negation Principle

The principle of Strong Negation [20] defines the relation between positive and negation conclusions. It preserves the coherence and consistency of the conclusions being derived.

\[^2\text{(a}_1, a_2, a_3, a_4) \leq (b_1, b_2, b_3, b_4) \text{ iff } a_i \subseteq b_i \text{ where } 1 \leq i \leq 4\]
In DL, the purpose of the $-\Delta$ or $-\partial$ inference rules is to establish that it is not possible to prove a corresponding positive tagged literal. As shown in the proof theory, a negative conclusions can only be concluded when all its positive counterparts are not derivable, i.e, for a literal $q$, the rules are defined in such a way that all the possibilities for proving $+\partial q$ are explored and shown to fail before $-\partial q$ can be concluded.

As a result, there is a close relationship between the inference rules for $+\Delta$ and $-\Delta$, and $+\partial$ and $-\partial$. The structure of the proof conditions are the same but the conditions are negated in some sense. We say that the proof condition for a tag is the strong negation for its complement. That is, $+\partial$ (or respectively $-\partial$) is the strong negation of $-\partial$ (or $+\partial$). Table 2.1 below defines the formulas for strong negation ($\text{sneg}$). The strong negation of a formula is closely related to a function that simplifies a formula by moving all negations to an innermost position in the resulting formula.

\[
\begin{align*}
\text{sneg}(+\partial p \in X) & = -\partial p \in X \\
\text{sneg}(-\partial p \in X) & = +\partial p \in X \\
\text{sneg}(A \land B) & = \text{sneg}(A) \lor \text{sneg}(B) \\
\text{sneg}(A \lor B) & = \text{sneg}(A) \land \text{sneg}(B) \\
\text{sneg}(\exists x A) & = \forall x \text{ sneg}(A) \\
\text{sneg}(\forall x A) & = \exists x \text{ sneg}(A) \\
\text{sneg}(\neg A) & = \neg \text{sneg}(A) \\
\text{sneg}(A) & = \neg A \quad \text{where } A \text{ is a pure formula}
\end{align*}
\]

Table 2.1: Strong Negation formula

A pure formula is a formula that does not contain a tagged literal. The strong negation of the applicability condition of an inference rule is a constructive approximation of the conditions where the rule is not applicable.

**Definition 2.6. The Principle of Strong Negation** is that for each pair of tags such as $+\partial$ and $-\partial$, the inference rule for $+\partial$ should be the strong negation of the inference rule of $-\partial$ (and vice versa).

Clearly, DL satisfies this principle. And in fact, all DL variants discussed in this thesis satisfy it. However, some variants proposed by Nute [167] may violate it.

### 2.1.6 Coherence and Consistency

There are two other important properties that DL may have: coherence and consistency. A theory is coherent if, for a literal $p$, we cannot establish simultaneously that $p$ is both provable or unprovable [35]. That is, we cannot derive from the theory that $D \vdash +\Delta p$ and $D \vdash -\Delta p$, or $D \vdash +\partial p$ and $D \vdash -\partial p$. On the other hand, consistency says that a literal and its negation can both be defeasible provable only when it and its negation are definitely provable; hence defeasible reasoning does not introduce inconsistency. A logic is coherent (consistent) if the meaning of each theory of the logic, when expressed as an extension, is coherent (consistent).
Theorem 2.7. [107] Let \( L \) be a DL where all proof tags are defined according to the principle of strong negation. Let \(+\#\) and \(-\#\) be two proof tags in \( L \) and \( D \) be a defeasible theory. There is no literal \( p \) such that \( D \vdash_L +\#p \) and \( D \vdash_L -\#p \).

Intuitively the above theorem states that no literal is simultaneously provable and demonstrably unprovable (with the same strength). So, in the light of this, we can then establish that DL is both coherent and consistent.

Proposition 2.8. [35] Defeasible logic is coherent and consistent.

We conclude this section with a few remarks. First, strict rules are used in two different ways. When we try to establish definite provability, strict rules are used as in classical logic: if the antecedents of the rules can be proved with the right strength, then they are applied regardless of any reasoning chains with the opposite conclusions. But strict rules can also be used to show defeasible provability, given that some other literals are known to be defeasible provable. In this case, strict rules are used exactly like defeasible rules. For example, a strict rule may be applicable, yet it may not fire because there is a rule with the opposite conclusion that is not weaker. Also, even though it may look a bit confusing, strict rules are not automatically superior to defeasible rules.

The elements of a derivation \( P \) in \( D \) are called lines of derivation. In the above definition often we refer to the fact that a rule is currently applicable. This may create the wrong impression that this applicability may change as the proof proceeds (something found often in non-monotonic proofs). But the sceptical nature of DL does not allow for such situation. For example, if we have established that a rule is currently not applicable because we have \(-\partial a\) for some antecedent \( a \), this means that we have proven at a previous stage that \( a \) is not provable from the defeasible theory \( D \) per se. The rule then cannot become applicable at a later stage of the proof or, indeed, at any stage of any proof based on the same defeasible theory.

We say that a literal \( l \) is provable in \( D \), denoted \( D \vdash l \) iff there is a line of derivation in \( D \) such that \( l \) is a line of \( P \). We also say that two defeasible theories \( D_1 \) and \( D_2 \) are conclusion equivalent (denoted \( D_1 \equiv D_2 \)) iff \( D_1 \) and \( D_2 \) have identical set of conclusions, that is \( D_1 \vdash l \) iff \( D_2 \vdash l \).

Finally, DL is closely related to several non-monotonic logics [17]. In particular, the “directly sceptical” semantics of non-monotonic inheritance networks [119] can be considered as an instance of inference in DL once an appropriate preferences between rules are fixed [39]. Besides, DL is a conservative logic (without negation as failure) in the sense of Wagner [213] and can be characterized by Kunen semantics [153, 16].

2.2 Variants of Defeasible Logic

In [153, 16] the authors have proposed a framework for the definition of variants of DL, which allow us to “tune” the logic to deal with different non-monotonic phenomena. In this section we
briefly discuss some intuitions of non-monotonic reasoning, based on the above structures, and we present formal definition of the conditions on proof corresponding to the various intuitions. The various instances are obtained by varying the concrete definitions of applicable, discarded, supported, unsupported and defeated. Since the notions are used in the concrete instances of $+\partial q$ and $-\partial q$, applicable and discarded are the strong negation of each other and so are supported and unsupported.

From now on, for a defeasible theory $D$, any tag $d$ and any literal $q$ we write $D \vdash dq$ iff $+dq$ appears in a derivation in $D$ and write $+d(D)$ to denote $\{q | D \vdash dq\}$, and similarly for $-d$.

### 2.2.1 Removing Team Defeat

The DLs we have discussed so far incorporate the idea of team defeat. That is, an attack on a rule with head $p$ by a rule with $\sim p$ may be defeated by a different rule with head $p$. Even though the idea of team defeat is natural, many other non-monotonic formalism and most argumentation frameworks do not adopt this idea.

However, it is possible to define a variant of DL without team defeat by changing the inference condition $+\partial$ in clause (2.3.2), as follows:

\[(2.3.2) \quad r > s\]

In other word, an attack on rule $r$ by rule $s$ can only be defended by $r$ itself, in the sense that $s$ is weaker than $r$. We use the tag $\partial_{ntd}$ to refer to defeasible provability in this variants.

### 2.2.2 Ambiguity Blocking and Ambiguity Propagation

We call a literal $p$ ambiguous if there is a chain of reasoning that support the conclusion that $p$ is true, and another one that supports the conclusion that $\sim p$ is true, and the superiority relation does not resolve the conflict.

**Example 2.1** (Nixon Diamond Problem [183]). The following is a classic example of non-monotonic inheritance .

\[
\begin{align*}
  r_1: & \quad \text{quaker} \Rightarrow \text{pacifist} \\
  r_2: & \quad \text{republican} \Rightarrow \sim\text{pacifist} \\
  r_3: & \quad \text{republican} \Rightarrow \text{footballFan} \\
  r_4: & \quad \text{pacifist} \Rightarrow \text{antimilitary} \\
  r_5: & \quad \text{footballFan} \Rightarrow \sim\text{antimilitary}
\end{align*}
\]

The superiority relation is empty.

Figure 2.1 shows the inheritance network of the theory.

Given both quaker and republican as facts, the literal pacifist is ambiguous since there are two applicable rules ($r_1$ and $r_2$) with the same strength, each supporting the negation of the other. Similarly, the literal antimilitary is ambiguous since $r_4$ support antimilitary while $r_5$ support $\sim$antimilitary.
In DL the ambiguity of *pacifist* results in the conclusions $-\partial$*pacifist* and $-\partial$–$\neg$*pacifist*. That is, DL refuses to draw any conclusion on whether the literal *pacifist* is provable or not. Consequently this results in no applicable rule supporting the verdict of *antimilitary*(r₄) and DL will conclude $+\partial$–$\neg$*antimilitary* since there is no valid path from the *quaker* to *pacifist* to *antimilitary*. This behavior is called *ambiguity blocking*, since the ambiguity of *antimilitary* has been blocked by the conclusion $-\partial$*pacifist* and an unambiguous conclusion about *antimilitary* has been drawn (Figure 2.2a).

However, it may be preferable for ambiguity to be propagated from *pacifist* to *antimilitary* since we are reserving the judgment of whether the literal *pacifist* is provable or not, but possibly it could be. In that sense it could possibly be anti-military. That is, being a football fan, it could also be not anti-military, so we could also reserve the judgment of whether the literal *antimilitary* is provable or not. This behavior is called *ambiguity propagation* since the ambiguity of a literal will be propagated along the line of reasoning (Figure 2.2b).

Ambiguity propagation behaviour in DL can be achieved through modifying the definition of the key notions of *supported* and *unsupported* rules: by making it more difficult to cast aside a competing rule, which subsequently making it easier to block a conclusion. In Example 2.1, ambiguity was blocked by using the non-provability of *pacifist* to discard r₄. We can achieve ambiguity propagation behaviour in this example by not rejecting r₄ though its antecedent is not provable. To do so another kind of proof, called *support* [16], is introduced.

$$+\Sigma) \text{ If } P(n+1) = +\Sigma q \text{ then either}$$

(1) $+\Delta q \in P(1..n)$ or

(2) (2.1) $-\Delta\neg q \in P(1..n)$; and

(2.2) $\exists r \in R[q]$ such that

(2.2.1) $\forall a \in A(r), +\Sigma a \in P(1..n)$

(2.2.2) $\forall s \in R[\neg q]$ either

$\exists a \in A(s), -\Sigma a \in P(1..n)$, or $s \neq r$. 

19
Figure 2.2: Inheritance network of defeasible theory in Example 2.1 under ambiguity blocking and ambiguity propagation.

\[-\Sigma\) If \(P(n + 1) = -\Sigma q\) then
\((1) +\Delta q \notin P(1..n); \text{ and}\n\((2) (2.1) +\Delta \sim q \in P(1..n) \text{ or}\n\quad (2.2) \forall r \in R[q] \text{ either}\n\qquad (2.2.1) \exists a \in A(r), -\Sigma a \in P(1..n); \text{ or}\n\qquad (2.2.2) \exists s \in R[\sim q] \text{ such that} \n\quad \forall a \in A(s), +\Sigma a \in P(1..n), \text{ and } s > r.\]

Support for a literal \(p\) consists of a chain of reasoning that would lead us to conclusion \(p\) in the absence of defeasibly derivable attack on some reason step.

Example 2.1 (continuing from p. 18). Given both \(\text{quaker}\) and \(\text{republican}\) as facts, all literals \((\text{pacifist}, \sim \text{pacifist}, \text{antimilitary}, \text{and } \sim \text{antimilitary})\) are supported. But if \(r_2 > r_1\) is added, then neither \(\text{pacifist}\) nor \(\text{antimilitary}\) are supported.

A literal that is defeasibly provable is supported, but a literal may be supported even though it is not defeasibly provable. Hence we have the following proposition.

Proposition 2.9. Support is a weaker notion than defeasible provability.
Consequently, with this proposition, the ambiguity propagation variant of DL can be easily achieved by making minor changes to the inference conditions for $+\partial$ and $-\partial$, and is defined as follow [16].

$+\partial_{ap}$) If $P(n + 1) = +\partial_{ap}q$ then either

1. $q \in F$; or
2. ($2.1$) $\sim q \notin F$ and
   2.1. $\exists r \in R[q] \forall a \in A(r) : +\partial_{ap} \in P(1..n)$ and
   2.2. $\forall s \in R[\sim q]$ either
      2.2.1. $\exists a \in A(s) : -\Sigma a \in P(1..n)$ or
      2.2.2. $\exists t \in R[q]$ such that
            $\forall a \in A(t) : +\partial_{ap} \in P(1..n)$ and $t > s$.

$-\partial_{ap}$) If $P(n + 1) = -\partial_{ap}q$ then

1. $q \notin F$ and
2. ($2.1$) $\sim q \in F$ or
   2.1. $\forall r \in R[q] \exists a \in A(r) : -\partial_{ap}a \in P(1..n)$ or
   2.2. $\exists s \in R[\sim q]$ such that
      2.2.1. $\forall a \in A(s) : +\Sigma a \in P(1..n)$ and
      2.2.2. $\forall t \in R[q]$ either
            $\exists a \in A(t) : -\partial_{ap}a \in P(1..n)$ or $t \neq s$.

The proof tags $+\partial_{ap}$ and $-\partial_{ap}$ capture defeasible provability using ambiguity propagation [16, 15, 41]. Their explanation is similar to that of $+\partial$ and $-\partial$. The major different is to prove $p$ this time we only make easier to attack it ($2.3$). Instead of requiring that the arguments attacking it are justified arguments, i.e., rules whose premises are provable, we just ask for defensible arguments (i.e., credulous arguments), that is rules whose premises are just supported.

We will extend the discussion of ambiguity propagation in Chapter 3 and an algorithm for computing the extensions of ambiguity propagation variant of DL will be presented in section 3.5.1.

### 2.2.3 Well-Founded Semantics

**Example 2.2.** Consider the following theory $D$:

$$
\begin{align*}
  r & : p \Rightarrow p \\
  r' & : \Rightarrow \neg p \\
  r > r'
\end{align*}
$$

Here the theory $D$ with Kunen semantics fails to derive $+\partial p$. The reason is that it does not able to detect that the first rule can never be applied. The aforementioned provability only works for “well-behaved” defeasible theory, a defeasible theory which constitute *Cumulativity* [155], which is called *well-founded* [153]. Thus if the defeasible theory $D$ consists of a single rule $p \rightarrow p$
(or respectively \(p \Rightarrow p\)), we can see that \(p\) cannot be proven definitely (defeasibly), but DL is unable to conclude \(-\Delta p\) \((-\partial p\)) and so as \(\neg p\) in the example above.

Recently, several efforts have been made to incorporate “loop-checking” into the inference mechanism of the logics, either to support the intuition that circular reasoning should be recognized as failure \((-\partial\)) to ensure the termination of inference, or to increase the power of the logics (i.e., so that more conclusions can be drawn). Hence, building on the bottom-up definition of the proof conditions and inspired by the work of [209], a well-founded defeasible logic (WFDL) was introduced [153], as an example demonstrating the power of a decomposition of inference in DL into two parts: inference structure and failure detection.

There loop checking was subsumed into the detection of unfounded sets, and could be implemented by the use of the well-founded semantics of logic programs [209]. Other work [16, 101, 41] investigated DLs with various inference structures under the original notion of failure detection [167, 18] (corresponding to Kunen semantics [139]), but not under the unfoundedness notion.

Nute and his colleagues, in a series of works [67, 169, 156, 155], have investigated DL with explicit loop-checking mechanisms and their relationship to well-founded semantics of logic programs. In particular they have defined two DLs, namely: NDL and ADL, and shown to be equivalent to well-founded semantics, and other properties were proved.

Concurrently, Billington [36, 37] has developed several logics involving incorporating loop checking mechanisms. Finally, work on courteous logic program [110, 111, 215] is based on the well-founded semantics of logic programs.

In the light of this, in this thesis, we will extend this study to logics employing the various inference structures under the unfounded notion of failure. In Chapter 3, further properties about the WFDL, especially properties of the unfounded set, will be discussed. Algorithms for computing the extensions of WFDL and unfounded set will also be presented in section 3.5.2.

### 2.2.4 Combination

It is worth knowing that several features can be easily integrated in our framework. In [16], the authors have showed the design of ambiguity propagation of DL without team defeat, as shown below:

\[ +\partial_{ap,ntd} \text{ If } P(n+1) = +\partial_{ap,ntd} \text{ then either} \]

\[ (1) +\Delta q \in P(1..n) \text{ or } -\Delta \sim q \in P(1..n); \text{ and} \]

\[ (2) (2.1) \exists r \in R_{sd}[q] \forall a \in A(r) : +\partial_{ap,ntd}a \in P(1..n) \text{ and} \]

\[ (2.2) -\Delta \sim q \in P(1..n) \text{ and} \]

\[ (2.3) \forall s \in R[\sim q] \text{ either} \]

\[ (2.3.1) \exists a \in A(s) : -\Sigma a \in P(1..n) \text{ or} \]

\[ (2.3.2) r > s \]

\footnote{For finite propositional theories, unfoundedness reduces to the existence of loops.}
If $P(n + 1) = -\partial_{ap,ntd}$ then

1. $-\Delta q \in P(1..n)$ and
2. $\Delta \sim q \in P(1..n)$ or

(2.1) $\forall r \in R_{sd}[q] \exists a \in A(r) : -\partial_{ap,ntd}a \in P(1..n)$ or
(2.3) $\Delta \sim q \in P(1..n)$ or
(2.4) $\exists s \in R[\sim q]$ such that
    (2.4.1) $\forall a \in A(s) : +\Sigma a \in P(1..n)$ and
    (2.4.2) $r \not\succ s$

So, it is quite obvious that $+\partial_{ap}$ is modified to $+\partial_{ap,ntd}$ in the same way that $+\partial$ was modified to $+\partial_{ntd}$. This observation underscores the orthogonality of the two concepts (team defeat and ambiguity propagation).

In addition, the authors also stated (in the same paper) that all variants stated above satisfy the basic property of coherence, i.e.:

**Theorem 2.10.** There is no defeasible theory $T$ and literal $q$ such that $T \vdash +\delta q$ and $T \vdash -\delta q$, where $\delta$ denotes any of the tags we have presented ($\Delta, \partial, \partial_{ap}, \partial_{ntd}, \partial_{ap,ntd}$).

In [16, 41], the authors have compared the relative strength of different notions of inference within the framework of DLs, as shown below:

**Theorem 2.11 (Inclusion Theorem of DL [41]).** (a) $+\Delta \subseteq +\partial_{ap,ntd} \subseteq +\partial_{ap} \subseteq \partial$
(b) $-\partial \subseteq -\partial_{ap} \subseteq -\partial_{ap,ntd} \subseteq -\Delta$
(c) For each inclusion relationship in (a) and (b) there are defeasible theories $D$ such that inclusion is strict.

An important consequence of this theorem is that DL exhibits different grades of inference, representing different degrees of cautiousness in drawing conclusions. It is interesting that this is achieved without using numerical weights such as probabilities or uncertainty factors.

The inclusions (a) and (b) in Theorem 2.11 look expected. For example, the relation $+\partial_{ap,ntd} \subseteq \partial_{ap}$ appears trivial since the absence of team defeat makes the inference rule weaker. But notice that there is a potential source of complication: when the logic fails to prove a literal $p$ and instead shows its non-provability, then that result may be used by the logic to prove another literal $q$ that could not be proven if $p$ were provable. In fact, while $\partial_{ap,ntd}$ is weaker than $\partial_{ap}$, as stated in Theorem 2.11, $\partial_{ntd}$ is not weaker than defeasible provability with team defeat ($\partial$).

### 2.3 Argumentation Semantics

As our conclusions are not guaranteed to be true, we must countenance the possibility that new information will lead us to change our minds, withdrawing previously adopted beliefs.

*Pollock [175]*
Argumentation systems are yet another way to study non-monotonic reasoning, and recently abstract argumentation frameworks [71, 212] have been developed to support the characterization of non-monotonic reasoning in argumentation-theoretic terms. The basic elements of these frameworks are the notions of arguments and “acceptability” of arguments, i.e., using the construction and comparison of arguments for and against certain conclusions.

As described in [200], an argument can be considered as a support for certain literal. However, this support does not guarantee that the literal will be concluded. We have to take into account values of counterarguments and specificity, which may make it unreasonable to maintain the previous belief, during the reasoning process.

Although DL can be described informally in terms of arguments, the logic has been formalized in a proof-theoretic setting in which arguments play no role. To fill this gap, Governatori and colleagues [101] have put forward a study to establish the connections between DL and other non-monotonic formalism through argumentation. In their framework\(^4\), an argument for a literal \(p\) based on a set of rules \(R\) is a (possibly infinite) proof tree (or non-monotonic derivation) with nodes labeled by literals such that the root is labeled by \(p\) and for every node with label \(h\):

1. If \(b_1, \ldots, b_n\) label the children of \(h\) then there is a rule in \(R\) with body \(b_1, \ldots, b_n\) and head \(h\).
2. If this rule is a defeater then \(h\) is the root of the argument.
3. the arcs in a proof tree are labelled by the rules used to obtain them.

Condition (2) specifies that a defeater may only be used at the top of an argument; in particular, no chaining of defeaters is allowed and defeaters can only be used to block derivations inside the framework.

Given a defeasible theory \(D\), the set of arguments can be generated from \(D\) is denoted by \(\text{Args}_D\).

Any literal labelling a node of an argument \(A\) is called a conclusion of \(A\). However, when referring to the conclusion of an argument, we refer to the literal labelling the root of the argument. A (proper) sub-argument of an argument \(A\) is a (proper subtree) of the proof tree associated to \(A\).

**Proposition 2.12.** Let \(D\) be a defeasible theory and \(p\) be a literal.

1. \(D \vdash +\Delta p\) iff there is a strict supportive argument\(^5\) for \(p\) in \(\text{Args}_D\);
2. \(D \vdash -\Delta p\) iff there is no (finite or infinite) strict argument for \(p\) in \(\text{Args}_D\).

\(^4\)Please be advised that our discussions here will be limited to the aspects are relevant to this thesis. Reader interested to this topic please refer to [101] for details.

\(^5\)A supportive argument is a finite argument in which no defeater is used, a strict argument is an argument in which only strict rules are used, and an argument that is not strict is called defeasible.
At the same time, the connection between the notion of support in DL and the existence of arguments can be characterized as follow.

**Proposition 2.13.** Let $D$ be a defeasible theory and $p$ be a literal.

1. $D \vdash +\Sigma p$ iff there is a supportive argument for $p$ in $\text{Args}_D$;
2. $D \vdash -\Sigma p$ iff there is no (finite or infinite) argument ending with a supportive rule for $p$ in $\text{Args}_D$.

Both propositions follow immediately since strict provability in DL, support in DL, and arguments are monotonic proofs where no conflicting rules, respectively arguments, are considered.

### 2.3.1 Conflicting Arguments: Attack and Undercut

In this subsection, the interaction between defeasible arguments are presented. Obviously it is possible that arguments support contradictory conclusions. For example, consider the following defeasible theory $D$:

\[
\begin{align*}
&\Rightarrow d \quad a, \neg b \Rightarrow c \\
&\Rightarrow e \quad e \rightarrow a \\
&\Rightarrow f \quad f \rightsquigarrow b \\
&d \Rightarrow \neg b
\end{align*}
\]

The arguments $\Rightarrow f \rightsquigarrow b$ and $\Rightarrow d \Rightarrow \neg b$ are conflicting.

**Definition 2.14.** A defeasible argument $A$ is supported by a set of arguments $S$ if every proper sub-argument of $A$ is in $S$.

Note that despite the similarity of name, this concept is not directly related to the support in DL, nor to supportive arguments/proof trees. Essentially the notion of supported argument is meant to indicate when an argument may have an active role in proving or preventing the derivation of a conclusion. The main difference between the above notions is that infinite arguments and arguments ending with defeaters can be supported (thus preventing some conclusions), while supportive proof trees are finite and do not contain defeaters (cf. Proposition 2.13).

This condition determines which arguments can attack or counter-attack, other arguments that are defined in the proof tree. In the proof theory, a defeasible conclusion is shown to have a proof condition consisting of three phases. In the first phase, a supporting rule $r$ for the desired conclusion is provided. In the second phase, all possible attacks provided by conflicting rule(s) against the conclusions are considered and rebutted. In the third phase, counter-attacks by stronger rule(s) are proposed and evaluated.

So in the proof condition, the relation of attack between the first and second phase is somewhat different from the attack between the second and third phase. To reflect this, the notions of attack and undercut (defeat) between arguments are defined as follow.
Definition 2.15. An argument $A$ attacks a defeasible argument $B$ if a conclusion of $A$ is the complement of a conclusion of $B$, and that conclusion of $B$ is not part of a strict sub-argument of $B$. A set of arguments $S$ attacks a defeasible argument $B$ if there is an argument $A$ in $S$ that attacks $B$.

Definition 2.16. A defeasible argument $A$ is undercut by a set of arguments $S$ if $S$ supports an argument $B$ attacking a proper non-strict sub-argument of $A$.

That an argument $A$ is undercut by $S$ means that some premises of $A$ cannot be proved if the arguments in $S$ is accepted.

It is worth noting that the above definition concerns only defeasible arguments and sub-arguments; for strict arguments, it is stipulated that they cannot be undercut or attacked. This is in line with definite provability in DL, where conflicts among rules are disregarded.

2.3.2 The Status of Arguments

Comparing arguments by pairs is not sufficient enough since an attacking argument can in turn be attacked by other arguments. As in many argumentation systems, the heart of argumentation semantics lies in the notion of acceptable argument. Based on this it is possible to define justified arguments and justified conclusions, conclusions that may be drawn even taking conflicts into account. Intuitively, an argument $A$ is acceptable w.r.t. a set of arguments $S$ if, once we accept $S$ as valid arguments, we feel compelled to accept $A$ as valid.

However, the notion of acceptable argument can be defined variously as they can be used to characterise different variants of DL. In Chapter 5, we will characterize the ambiguity blocking variant of DL under distributed environment. For the moment we leave this notion open as a parameter that may be instantiated in different ways, and proceed to define the set of justified and reject arguments.

Definition 2.17. Let $D$ be a defeasible theory. We define $J_i^D$ as follows:

- $J_0^D = \emptyset$;
- $J_{i+1}^D = \{a \in \text{Args}_D \mid a \text{ is acceptable w.r.t. } J_i^D\}$

The set of justified arguments in defeasible theory $D$ is $J\text{Args}^D = \bigcup_{i=1}^{\infty} J_i^D$.

A literal $p$ is justified if it is the conclusion of a supportive argument in $J\text{Args}^D$.

That is an argument $A$ is justified if it can resist every reasonable refutation. However, DL is more expressive since it is able to say when a conclusion is demonstrably non-provable ($-\partial, -\partial p$), indicating that every possible argument not involving defeaters has been refuted. The definition below shows how this notion can be captured in the argumentation framework, by assigning the status rejected to arguments that are refuted.

Roughly speaking, an argument is rejected if it has a rejected sub-argument or it cannot overcome an attack from another argument. Again there are several possible definitions for
the notion of rejected arguments. Similarly we will leave this notion temporarily open and will discuss it more in Chapter 5.

**Definition 2.18.** Let $D$ be a defeasible theory and $T$ be a set of arguments. We define $R^D_i(T)$ as follows:

- $R^D_0(T) = \emptyset$;
- $R^D_{i+1}(T) = \{ a \in \text{Args}_D \mid a \text{ is rejected by } R^D_i(T) \text{ and } T \}$

The set of rejected arguments in defeasible theory $D$ w.r.t. $T$ is $R\text{Args}^D(T) = \bigcup_{i=1}^{\infty} R^D_i(T)$. An argument is rejected if it is rejected w.r.t. $J\text{Args}^D$.

So, a literal $p$ is rejected by $T$ if there is no argument $\text{Args}_D - R\text{Args}^D(T)$ where the top rule is a strict or defeasible rule with head $p$. A literal is rejected if it is rejected by $J\text{Args}^D$.

It is worth noting that a literal $p$ is not necessarily rejected if there is no supportive argument for $p$ in $\text{Args}_D - R\text{Args}^D(T)$ since there may be an infinite argument for $p$ in $\text{Args}_D - R\text{Args}^D(T)$ without defeaters. Thus it is possible for a literal to be neither justified nor rejected. This situation is the similar to DL where we may have both $D \not\vdash +\partial p$ and $D \not\vdash -\partial p$. A sufficient condition that prevents this situation is the acyclicity of the atom dependency graph [101].

In addition to the above definitions, in the paper, the authors also showed how the ambiguity propagation variant of DL can be characterized by the Dung’s grounded semantics [70]; while on the other hand, ambiguity blocking does not corresponding to any existing argumentation semantics in Dung’s framework.

## 2.4 Extending Defeasible Logic with Modal Operators

Modal logics have been put forward to capture many different notions somehow related to the intensional nature of agency as well as many other notions. Usually modal logics are extensions of classical propositional logic with some intensional operators. Thus any modal logic should account for two components: (1) the underlying logical structure of the propositional base; and (2) the logical behavior of the modal operators. As is well-known, classical propositional logic is not well suited to deal with real life scenarios. The main reason is that the descriptions of real-life cases are, very often, partial and somewhat unreliable. In such circumstances, classical propositional logic is doomed to suffer from the same problems.

On the other hand, the logic should specify how modalities can be introduced and manipulated. Some common rules for modalities are, e.g.,

\[
\begin{align*}
\quad \quad & \quad \vdash \phi \\ & \quad \vdash \Box \phi & \quad \text{Necessitation} \\
\quad \vdash \phi \supset \psi \\ & \quad \vdash \Box \phi \supset \Box \psi & \quad \text{RM}
\end{align*}
\]

Both dictate conditions to introduce modalities purely based on the derivability and structure of the antecedent. These rules are related to the problem of logical omniscience [116, 210]: if $\Box$
corresponds to either INT (Intention), BEL (Believe), OBL (Obligation), they put unrealistic assumptions on the capability of an agent. However, if we take a constructive interpretation, i.e., if an agent can build a derivation of $\psi$ then she can build a derivation of $\Box \psi$.

In the light of this, [94] has proposed an extension of DL to capture combinations of mental attributes and deontic concept, as well as multiple (defeasible) consequence relations. In the paper, the authors have proposed a framework that replace the derivability in classical logic with a practical and feasible notion like derivability in DL. Thus, the intuition behind this work is that we are allowed to derive $\Box_i p$ if we can prove $p$ with the mode $\Box_i$ in DL.

The details of the framework are as follow. To extend DL with modal operators we have two options: (1) to use the same inferential mechanism as basic DL and to represent explicitly the modal operators in the conclusion of rules [168]; (2) introduce new types of rules for the modal operators to differentiate between modal and factual rules.

For example, the “deontic” statement “The Purchaser shall follow the Supplier price lists” can be represented as

$$\text{AdvertisedPrice}(X) \Rightarrow O_{\text{purchaser}} \text{Pay}(X)$$

if we follow the first option and

$$\text{AdvertisedPrice}(X) \Rightarrow_{O_{\text{purchaser}}} \text{Pay}(X)$$

according to the second option, where $\Rightarrow_{O_{\text{purchaser}}}$ denotes a new type of defeasible rule relative to the modal operator $O_{\text{purchaser}}$. Here, $O_{\text{purchaser}}$ is the deontic “obligation” operator parametrized to an actor/role/agent, in this case the purchaser.

As stated in [94], the differences between the two approaches, besides the fact that in the first approach there is only one type of rules while the second accounts for factual and modal rules, are two: (i) the first approach has to introduce the definition of $p$-incompatible literals (i.e., a set of literals that cannot be hold when $p$ holds.) for every literal $p$. For example, we can have a modal logic where $\Box p$ and $\neg p$ cannot both be provable at the same time; (ii) the first approach is less flexible than the second: in particular in some situations it must account for rules to derive $\Diamond p$ from $\Box p$; similarly conversions – which permit to use a rule for a certain modality as it were for another modality (cf. Section 2.5.2) – require additional operational rules in a theory, thus the second approach seems to offer a more conceptual tool than the first one. It seems that the second approach can use different proof conditions based on the modal rules to offer a more fine grained control over the modal operators, which allows us to modal interaction over operators.

The language of Modal Defeasible Logic (MDL) consists of a finite set of modal operators $\text{Mod} = \{\Box_1, \ldots, \Box_n\}$ and a (numerable) set of atomic propositions $\text{Prop} = \{p, q, \ldots\}^6$. Besides,  

---

6The language can be extended to deal with other notions. For example to model agents, we have to include a (finite) set of agents, and then the modal operators can be parameterised with the agents. For a logic of action or planning, it might be appropriate to add a set of atomic actions/plans, and so on depending on the intended applications.

28
the definition of literal (an atomic proposition or the negation of it) is supplemented with the following clause:

- If \( l \) is a literal then \( \Box_i l \), and \( \neg \Box_i l \), are literals if \( l \) is different from \( \Box_i m \), and \( \neg \Box_i m \), for some literal \( m \).

which prevents literals from having sequences of modalities where successive occurrences of one and the same modality; however, iterations like \( \Box_i \Box_j \) and \( \Box_i \Box_j \Box_i \) are legal in the language.

Accordingly, a Modal Defeasible Theory is defined as follow.

**Definition 2.19.** A Modal Defeasible Theory \( D \) is a structure \((F, R, \succ)\) where

- \( F \) is a set of facts ( literals or modal literals),
- \( R = R^B \cup \bigcup_{1 \leq i \leq n} R^{\Box_i} \), where \( R^B \) is the set of base ( un-modalised) rules, and each \( R^{\Box_i} \) is the set of rules for \( \Box_i \) and
- \( \succ \subseteq R \times R \) is the superiority relation.

A rule \( r \in R \) is an expression \( A(r) \rightarrow_X C(r) \) such that \( \rightarrow_X \in \{ \rightarrow, \Rightarrow, \sim \} \), \( X \) is \( B \), for a base rule, and a modal operator otherwise, \( A(r) \) the antecedent or body of \( r \) is a (possible empty) set of literals and modal literals, and \( C(r) \), the consequent or head of \( r \) is a literal if \( r \) is a base rule and either a literal or a modal literal \( Yl \) where \( Y \) is a modal operator different from \( X \). As in the sections before, given a set of rules \( R \), we use \( R_s, R_d, R_{sd} \) to denote the set of strict rules, defeasible rules, and strict and defeasible rules respectively; and \( R[q] \) to denote the set of rules in \( R \) whose head is \( q \).

The derivation tags are now indexed with modal operators. Let \( X \) range over \( Mod \). A conclusion can now have the following forms:

\[ +\Delta_X q \] \( q \) is definitely provable with mode \( X \) in \( D \) (i.e., using only facts and strict rules of mode \( X \)).

\[ -\Delta_X q \] \( q \) is definitely rejected with mode \( X \) in \( D \).

\[ +\partial_X q \] \( q \) is defeasibly provable with mode \( X \) in \( D \).

\[ -\partial_X q \] \( q \) is defeasibly rejected with mode \( X \) in \( D \).

That is, if we can prove \( +\partial_{\Box_i} q \), then we can assert \( \Box_i q \).

Similarly to standard DL, the provability of MDL is based on the concept of derivation in \( D \), which corresponds to a finite sequence \( P = (P(1), \ldots, P(n)) \) of tagged (modal) literals satisfying the proof conditions. \( P(1..n) \) denotes the initial part of the sequence \( P \) of length \( n \).

Before introducing the proof conditions for the proof tags for modal \( X \), we start with some auxiliary notions relevant to this thesis.
Definition 2.20. [173] Let \( \# \) be either \( \Delta \) or \( \partial \). Given a proof \( P = (P(1), \ldots, P(n)) \) in \( D \), a (modal) literal \( q \) is \( \Delta \)-provable in \( P \), or simply \( \Delta \)-provable, if there is a line \( P(m) \) of the derivation such that either:

1. if \( q = l \) then
   - \( P(m) = +\#l \) or
   - \( \Box_i l \) is \( \# \)-provable in \( P(1..m-1) \) and \( \Box_i \) is reflexive\(^7\)
2. if \( q = \Box_i l \) then
   - \( P(m) = +\#_i l \) or
   - \( \Box_j \Box_i l \) is \( \# \)-provable in \( P(1..m-1) \), for some \( j \neq i \) such that \( \Box_j \) is reflexive.
3. if \( q = \neg \Box_i l \) then
   - \( P(m) = -\#_i l \) or
   - \( \Box_j \neg \Box_i l \) is \( \# \)-provable in \( P(1..m-1) \), for some \( j \neq i \) such that \( \Box_j \) is reflexive.

In a similar way, a literal defines to be \( \Delta \) and \( \partial \)-rejected by taking, respectively, the definition of \( \Delta \)-provable and \( \partial \)-provable and changing all positive proof tags into negative proof tags, adding a negation in front of the literal when the literal is prefixed by a modal operator \( \Box_j \), and replacing all the \emph{ors} by \emph{ands}. Thus, for example, a literal \( \Box_i l \) is \( \partial \)-rejected if, in a derivation, a line \( \neg \partial_i l \), and the literal \( \neg \Box_i \neg l \) is \( \partial \)-rejected if we have \( +\partial_i \neg l \) and so on.

Definition 2.21. Let \( X \) be a modal operator and \( \# \) is either \( \Delta \) or \( \partial \).

- A literal \( l \) is \( \#_X \)-provable if the modal literal \( Xl \) is \( \# \)-provable;
- A literal \( l \) is \( \#_X \)-rejected if the literal \( Xl \) is \( \# \)-rejected.

Based on the above definition of provable and rejected literals the authors give the conditions to determine whether a rule is applicable or the rule cannot be used to derive a conclusion (i.e., the rule is discarded).

The proof conditions for \( +\Delta \) correspond to monotonic forward chaining of derivations. For a literal \( q \) to be definitely provable with the mode \( \Box \), a strict rule for \( \Box \) with head \( q \) whose antecedents have all been definitely provable previously. And to establish that \( q \) cannot be definitely proven we must establish that for every strict rule for \( \Box \) with head \( q \) there is at least one antecedent which has been shown to be non-provable. Besides, a rule for \( Y \) can be used as a rule for a different modal operator \( X \) in case all literals in the body of the rule are modalised with the modal operator we want to prove. For example, given the rule:

\[
p, q \rightarrow_{\text{BE}L} s
\]

\(^7\)A modal operator \( \Box_i \) is reflexive iff the truth of \( \Box_i \phi \) implies the truth of \( \phi \). In other words \( \Box_i \) is reflexive when we have the modal axiom \( \Box_i \phi \rightarrow \phi \).
we can derive $+\Delta_{\text{INT}}s$ if we have $+\Delta_{\text{INT}}p$, $+\Delta_{\text{INT}}q$ and the conversion $\text{Convert}(\text{BEL}, \text{INT})$ holds in the theory.

Conditions for $\partial_\Box$ are more complicated. A rule for a belief is applicable if all the literals in the antecedent of the rule are provable with the appropriate modalities; while the rule is discarded if at least one of the literals in the antecedent is not provable. As before, for other types of rules we have to take conversions into account. We have thus to determine conditions under which a rule for $Y$ can be used to directly derive a literal $q$ modalised by $X$. Roughly, the condition is that all the antecedents $a$ of the rule are such that $+\partial_\Box a$.

**Definition 2.22.** Let $X$ be a modal operator or $B$. A rule $r \in R$ is $\partial_X$-applicable iff

1. $r \in R^X$ and $\forall a_k \in A(r)$, $a_k$ is $\partial$-provable; or
2. if $X \neq B$ and $r \in R^B$, i.e., $r$ is a base rule, then $\forall a_k$, $a_k$ is $\partial_X$-provable.

**Definition 2.23.** Let $X$ be a modal operator or $B$. A rule $r \in R$ is $\partial_X$-discarded iff

1. $r \in R^X$ and $\exists a_k \in A(r)$, $a_k$ is $\partial$-rejected; or
2. if $X \neq B$ and $r \in R^B$, i.e., $r$ is a base rule, then $\exists a_k$, $a_k$ is $\partial_X$-rejected.

As a corollary of the above definitions, the following definition regarding the support of a literal is established.

**Definition 2.24.** Given a theory $D$, a literal $q$ is supported in $D$ iff there exists a rule $r \in R^X[q]$ s.t. $r$ is applicable, otherwise $q$ is not supported. For $X$ belongs to the set of modal operator in $D$, we use $+\Sigma_X q$ and $-\Sigma_X q$ to indicate that $q$ is supported by rules for $X$.

Defeasible derivations have an argumentation-like structure which is divided into three phases. In the first phase, we have to prove that the conclusion is supported. Then in the second phase, we have to consider all possible (actual and not) reasons against the desired conclusion. Finally, in the last phase, which can be done in two ways: (i) by showing that some of the premises of a counterargument do not hold, or (ii) by showing that the argument is weaker than an an argument in favour of the conclusion. This is formalised by the following (constructive) proof conditions.

$+\partial_X$: If $P(n + 1) = +\partial_X q$ then

1. $+\Delta_X q \in P(1..n)$, or
2. (2.1) $-\Delta_X \sim q \in P(1..n)$, and
   (2.2) $\exists r \in R_{sd}[q]$: $r$ is $\partial_X$-applicable, and
   (2.3) $\forall s \in R[\sim q]$ either $s$ is $\partial_X$-discarded, or
   (2.3.1) $\exists w \in R[q]$: $w$ is $\partial_X$-applicable and $w > s$
\[\neg \partial X: \text{If } P(n + 1) = -\partial_X q \text{ then}
\]
\begin{enumerate}
\item \(-\Delta_X q \in P(1..n)\) and either
\item (2.1) \(+\Delta_X \sim q \in P(1..n)\), or
\item (2.2) \(\forall r \in R_d[q]\), either \(r\) is discarded, or
\item (2.3) \(\exists s \in R[\sim q]: s\) is applicable, and
\begin{enumerate}
\item (2.3.1) \(\forall w \in R[\sim q]\) either \(w\) is discarded, or \(w \not> s\), or
\end{enumerate}
\end{enumerate}

The above condition is, essentially, the usual condition for defeasible derivations in DL, we refer the reader to [94] for more through treatments. The only point we want to highlight here is that base rules can play the role of modal rules when all the literals in the body are \(\partial_{\Box_i}\)-derivable. Thus, from a base rule \(a, b \Rightarrow_B c\) we can derive \(+\partial_{\Box_i} c\) if both \(+\partial_{\Box_i} a\) and \(+\partial_{\Box_i} b\) are derivable while this is not possible using the rule \(a, \Box_i b \Rightarrow_B c\). (see Section 2.5.2).

## 2.5 Modal Defeasible Logic with Interactions

Notice that the proof condition for \(+\partial\) given in Section 2.1.3 and then those for the other proof tags are the same as those of basic DL given in [18]. What we have showed before is essentially to consider \(n + 1\) non-monotonic consequence relation defined in DL and compute them in parallel. In the previous sections, we show that one of the advantages of modal logic is the ability to deal with complex notions composed by several modalities, or by interactions of modal operators. Thus, we have to provide facilities to represent such interactions. In MDL it is possible to distinguish two types of interactions: conflicts and conversions. In the next two subsections, we will show how these properties can be motivated and captured in the framework [94].

### 2.5.1 Conflicts

Consider the following simple inclusion axiom of multi-modal logic relating two modal operators \(\Box_1\) and \(\Box_2\):

\[\Box_1 \phi \rightarrow \Box_2 \phi\]

The meaning of this axiom is that every time we are able to prove \(\Box_1 \phi\), then we are able to prove \(\Box_2 \phi\). Thus, given the intended reading of the modal operators in our approach – a modal operator characterises a derivation using a particular model, it enables us to transform a derivation of \(\Box_1 \phi\) into a derivation of \(\Box_2 \phi\). If the logic is consistent, we also have that \(\Box_1 \phi \rightarrow \Box_2 \phi\) implies that it is not possible to prove \(\Box_2 \neg \phi\) given \(\Box_1 \phi\), i.e., \(\Box_1 \phi \rightarrow \neg \Box_2 \neg \phi\). However, this idea is better illustrated by the classically equivalent formula \(\Box_1 \phi \land \Box_2 \neg \phi \rightarrow \bot\). When the latter is expressed in form of the inference rule

\[
\frac{\Box_1 \phi, \Box_2 \neg \phi}{\bot}
\]
it suggests that it is not possible to obtain $\square_1\phi$ and $\square_2\neg\phi$ together. This does not mean that $\square_1\phi$ implies $\square_2\phi$, but that the modal operators $\square_1$ and $\square_2$ are in conflict with each other. MDL is able to differentiate between the two formulations. For the inclusion version (i.e., $\square_1\phi \rightarrow \square_2\phi$), we have to add the following clause to the proof conditions for $+\partial_{\square_1}$ (and the other proof tags accordingly) with the condition

$$+\partial_{\square_1}q \in P(1..n)$$

For the second case (i.e., $\square_1\phi \land \neg\square_2\phi \rightarrow \bot$), the following preliminary definition is needed.

**Definition 2.25.** Given a modal operator $\square_i$, $F(\square_i)$ is the set of modal operators in conflict with $\square_i$. If the only conflict axiom we have is $\square_1\phi \land \square_2\phi \rightarrow \bot$ then $F(\square_1) = \{\square_2\}$.

With $R^{F(\square_i)}$ we denote the union of rules in all $R^{\square_j}$ where $\square_j \in F(\square_i)$. At this point to implement the proof condition for the conflict all we can replace clause (2.3) of the definition of $+\partial_{\square_i}q$ with the clause

$$(2.3) \forall s \in R^{F(\square_i)}[\neg q] \text{ either}$$

$$s \text{ is } \partial_X\text{-discarded or}$$

$$\exists w \in R[q]: w \text{ is } \partial_X\text{-applicable and } w > s.$$ 

The notion of conflict has been proved useful in the area of cognitive agents, i.e., agent whose rational behaviour is described in terms of mental and motivational attitudes including beliefs, intentions, desires and obligations. Typically, agent types are characterised by stating conflict resolution methods in terms of orders of overruling between rules [47, 61, 62, 92, 94]. For example, an agent is realistic when rules for beliefs override all other components; she is social when obligations are stronger than the other components with the exception of beliefs. Formally, agent types can be characterised by stating that, for any types of rules $X$ and $Y$, for every $r$ and $r'$, $r \in R^X[q]$ and $r' \in R^Y[\neg q]$, we have that $r > r'$.

### 2.5.2 Conversions

Another interesting feature that could be explained using our formalism is that of rule conversion, which allows us to model the interactions between different modal operators [173]. In general, notice that in many formalisms it is possible to convert from one type of conclusion into a different one. For example, the right weakening rule of non-monotonic consequence relations (see, for example [136])

$$
\begin{array}{c}
B \vdash C \quad A \not\vdash B
\end{array}
\Rightarrow
\begin{array}{c}
A \not\vdash C
\end{array}
$$

allows for the combination of non-monotonic and classical consequences.

Suppose that a rule of a specific type is given and all the literals in the antecedent of the rule are provable in one and the same modality. If so, is it possible to argue that the conclusion of the rule inherits the modality of the antecedent? To give an example, suppose we have that $p, q \Rightarrow_{\square_i} r$ and that we obtain $+\partial_{\square_j}p$ and $+\partial_{\square_j}q$. Can we conclude $\square_jr$? In many cases this is a reasonable conclusion to obtain.
For this feature we have to declare which modal operators can be converted and the target of the conversion. Given a modal operator $\Box_i$, with $V(\Box_i)$ we denote the set of modal operators $\Box_j$ that can be converted to $\Box_i$. In addition, we assume that base rules can be converted to all other types of rules. The condition to have a successful conversion of a rule for $\Box_j$ into a rule for $\Box_i$ is that all literals in the antecedent of the rules are provable modalised with $\Box_i$. Formally we have thus to add (disjunctively) in the support phase (clause (2.2)) of the proof condition for $\partial_{\Box_i}$ the following clause

$$(2.2b) \exists r \in R^{V(\Box_i)}[q] \text{ such that } r \text{ is } \partial_{\Box_i}-\text{applicable}$$

The notion of conversion enables us to define new interesting agent types [94].

We conclude this section with a formalisation of the Yale Shooting Problem [113] that illustrates the notion of conversion. Let INT be the modal operator for intention. The Yale Shooting Problem can be described as follows\(^8\)

$$\text{liveAmmo, load, shoot } \Rightarrow_B \text{ kill}$$

This rule encodes the knowledge of an agent that knows that loading the gun with live ammunition, and then shooting will kill her friend. This example clearly shows that the qualification of the conclusions depends on the modalities relative to the individual acts “load” and “shoot”. In particular, if the agent intends to load and shoot the gun ($\text{INT}(\text{load})$, $\text{INT}(\text{shoot})$), then, since she knows that the consequence of these actions is the death of her friend, she intends to kill him ($+\partial_{\text{INT}}\text{kill}$). However, in the case she has the intention to load the gun ($+\partial_{\text{INT}}\text{load}$) and for some reason shoot it ($\text{shoot}$), then the friend is still alive ($-\partial\text{kill}$).

In Chapter 5 we will extend this framework further so as to cope with the situations that we need to handle in the application we described here.

### 2.6 Existing Reasoners for Defeasible Logic

A number of DL reasoners and systems have been proposed in recent years to cover different variants of defeasible reasoning as well as other intuitions of non-monotonic reasoning. Most of the implementations rely on either the transformation showed in Figure 1.1, or on translations to other formalisms or extended logic programming [12, 23, 149]. This section examines previous implementations of both types of defeasible reasoning engines and associated literature. Our focus is on systems that rely on symbolic approaches to deal with inference under uncertainty. Some systems reviewed do offer numerical support, but this is not the primary focus of this thesis, and as such discussion of these features is minimal.

\(^8\)Here we will ignore all temporal aspects and we will assume that the sequence of actions is done in the correct order.
2.6.1 d-Prolog

d-Prolog [59] is a non-monotonic extension of Prolog language based on Nute’s early work in DL [167]. d-Prolog was implemented as a query-answering interpreter within Prolog and was originally designed for solving small and non-recursive inheritance problems. The strict rules in d-Prolog are implemented directly as Prolog rules. So, when there are no disputed rules in the knowledge base, d-Prolog is substantially more efficient than other systems as rules are interpreted directly by the underlying Prolog system. However, in case of theories with disputed inference, d-Prolog performs badly, with time growing exponentially in the problem size [154].

d-Prolog also includes an implicit definition of superiority relation in terms of specificity\(^9\), to adjudicate conflicts between defeasible rules. Essentially a more specific is considered to be more superior than a less specific rule, which are determined by evaluating the bodies of the two conflicting rules to see if either can be derived from the other [48].

Even though the authors of [12] have showed that d-Prolog is capable of handling a relatively large knowledge base (with a maximum of about 20,000 rules, depending on the setting of the underlying rule engines), the results from tests conducted by Mather et al. [154] showed that d-Prolog demonstrates incompleteness inherited from Prolog. To be precise, d-Prolog does not implement loop-checking and inferencing process relies very much on the order in which the rules are given. And this effect becomes more obvious when it loops on circular lines of inference rules.

2.6.2 Deimos and Delores

Maher et al. [154] presented two defeasible logic implementations: one which implements a query-answering algorithm (for single query evaluation) while the other is a system for computing all conclusions of a given theory. The authors treat DL as a “sceptical non-monotonic reasoning system based on rules and a priority relation between rule [that] is used to resolve conflicts among rules, where possible” [154]. That is, instead of using the commonly used specificity to resolve conflicts (as in d-Prolog), priorities (or preferences) between rules was used, as encoded in many domains knowledge.

The query-answering system, Deimos [185], consists of a suite of tools that supports the authors’ ongoing research in this area. The main component of Deimos is a backward-chaining theorem prover for DL based directly on the inference rules. It was implemented using Haskell programming language and much of its code, along with the design strategy, is common to the Phobos querying system for Plausible logic [38], which was developed in parallel with Deimos. Deimos has been designed primarily for flexibility and traceability, and significant effort has been expanded to make the prover efficient. The proof of a conclusion is accomplished with a depth-first search with the memorization of already proved conclusions, and also loop-checking,

\(^9\)[167] Given a defeasible theory \(D = (F, R, >)\), A rule \(r_a \in R\) with antecedent \(A\) is considered to be more specific than another rule \(r_b \in R\) with antecedent \(B\) if we can derive all of \(B\) from \(A\) using only the rules in \(R\), but not vice versa.
which detects when a conclusion occurs twice in a branch of search tree. Both methods are implemented using a balanced binary tree of data.

The one-to-one correspondence between the inference rules and its representation as a Haskell expression ensures that Deimos can precisely represent what is specified by the logical formalism, which provides us not only a mean to verify the correctness of the system, but also facilities the implementations of different extensions of DL. In addition, the proof history provided by Deimos also help users to understand the computational behavior of the logics.

Users can interact with Deimos via a command line interface or through a web interface. The web interface provides full accessibility to the system in a fairly intuitive manner.

The system that computes all conclusions, Delores (DEfeasible LOgic REasoning System) [162] is implemented in C in about 4,000 lines of code. It is based on forward chaining but can only work for deriving positive conclusions. The negative conclusions are derived by a dual process.

Conclusions in Delores are derived through the use of pre-processing transformations as shown in Figure 1.1, that eliminates all uses of defeaters and superiority relations. The transformation is designed to provide incremental transformation of defeasible theories, and systematically uses new atoms and new defeasible rules to simulate the eliminated features. (A full treatment of the transformations, including proofs of correctness and other properties, can be found in [18].) However, as stated in the introduction and [154], even though the transformations can be applied in one pass, they are profligate and can increase the size of the theory by at a factor of 12 at most and the time taken to produce the transformed theory is linear to the size of the input theory.

The algorithm begins by deriving conclusions that can immediately be established: all facts are provable, and all literals with no rules supporting them are unprovable. Then the algorithm proceeds by modifying the rules in the theory. When inferring with positive consequences, the algorithm proceed similarly to unit resolution for definite clauses in classical logic: when an atom is proved, it can be eliminated from the bodies of all other definite clauses. That is, when a literal is established defeasibly, it can be removed from the body of all rules. Similarly, when it is established that a literal cannot be established defeasibly, then those rules which contain that literal in their body cannot be used to prove the head, and therefore they can be removed from the theory. However, when inferring a positive conclusion (+∂) for a literal p, defeasible provability is complicated, in comparison to definite clauses, by the need to consider rules for ∼p.

The algorithm for computing positive conclusions is similar to the bottom-up linear algorithm for determining satisfiability of Horn clauses of Dowling and Gallier [68]. One key difference is in the data structures: the Dowling and Gallier algorithm keeps a count of the number of atoms in the body of a rule involved in the inference process, whereas Delores keeps track of the computed body. Maher and his colleagues stated that their algorithm would result in greater memory usage, but allow them to reconstruct the residue of the computations, which is useful in understanding the behavior of a theory. However, they have not mentioned in their paper on how this residue is utilized within their algorithm or for off-line exploration. A complete analysis
of the system, including complexity analysis, correctness and empirical experiment results, are
documented in [154] for future reference.

2.6.3 DR-Prolog

DR-Prolog [12] is another Prolog based defeasible reasoner. It is a query-answering system which
aims at providing defeasible reasoning on the web. Its focuses have been put on: (1) compliance
with RuleML [189] so that defeasible theory can easily be portable among different systems
on the Internet; (2) integration with Semantic Web [28] technologies, such as RDF (Resource
Description Framework) [181] and OWL (Web Ontology Language) [24].

The core of the system consists of a well-studied translation [19, 13] of defeasible knowledge
into logic programs under Well-Founded Semantics, and invokes the underlying Prolog inference
engine to solve the program (currently utilising XSB Prolog). This declarative translation
distinguishes DR-Prolog from other implementations as the translation of a defeasible theory \( D \)
into a logic programming \( LP(D) \) has the goal of showing that “\( p \) is defeasible provable in \( D \)”
is equivalent to “\( p \) is included in the well-founded model of \( LP(D) \)”.

As stated in the paper, the main reason for the choice of well-founded semantics is due
to its low computational complexity. Besides, the translation of defeasible theories into logic
programs is done using the meta-programs of [19] instead of control literals [13] since the latter
approach would generate a large number of program clauses.

2.6.4 DR-DEVICE

DR-DEVICE [23] is a DL reasoner implemented on the top of CLIPS [58] (C Language In-
tegrated Production System) and R-DEVICE deductive rule system [22], and is capable of
reasoning about RDF metadata and RuleML over multiple Web sources using DL rules, and
support two types of negation (strong, negation-as-failure) and conflicting (mutually exclusive)
literals.

Similar to DR-Prolog, a defeasible theory is imported by the rule loader (or directly from the
user) through a set of rules in DR-DEVICE notation and then translated into a set of CLIPS
production rules. The underlying rule engine then runs the production rules and generates
objects that constitute the results of the defeasible theory, which can be exported to the user
as an RDF/XML document through the RDF extractor. Moreover, the core accepts knowledge
represented in different formats depending on the situations, and can be converted to a set
of CLIPS-like rules, or a set of “object-orientated” deductive rules for further processes. The
defeasible reasoning process is carried out through the compilation of rules into the generic rule
language supported by R-DEVICE. This approach has also been extended to capture modal
defeasible deontic logic [92, 133].

In short, [12] have put forward an empirical study to compare the performance of three
query-based defeasible reasoning systems mentioned above, namely, \textit{DR-Prolog}, \textit{Deimos}, and \textit{d-}
In the study, the authors have employed the DTScale tool of Deimos to create difference sets of theories\textsuperscript{10}, which consist of a large number of facts, rules, and superiority relations, for the evaluation. However, the shortcoming of this study is that the test theories generated do not contain user-defined conflicting literals since this is not a feature supported by Deimos and d-Prolog.

In the study, it was found that the performance of DR-Prolog is proportional to the size of the theory since the defeasible theories are translated into logical programs with the same number of rules before being processed. Besides, the study also shows that DR-Prolog performs better when there are strict rules in the theories since smaller number of rules are being processed.

Compare to the other two systems, DR-Prolog performs a little worse than Deimos in most of the test with undisputed inferences but performs almost the same for cases with disputed inferences; while DR-Prolog performs better than d-Prolog in the cases of complex theories (theories with a large number of rules and priorities) but d-Prolog performs substantially more efficient than DR-Prolog where there are only strict rules in the theories.

Among the implementations mentioned above, Delores is the only system that can compute all conclusions, while others are built as query answering systems. Efficiency is clearly an important element within all implementations. In [154], Maher and colleagues have performed large amount of empirical experiments comparing the performance (in terms of CPU time) of d-Prolog, Deimos and Delores. Even through the results was drawn based on different software platforms and configuration options, there is no doubt that Delores is to be the faster system in almost all experiments on which it could be run.

In the discussion, Maher and colleagues also concluded that the empirical evaluation illustrates that both of their implementations of DL result in linear execution times as expected. It is also apparent from the experiments that the overhead introduced to Delores by the pre-processing transformations varies quite significantly from problem to problem and is sometimes extraordinarily high, which thus support our words in the introduction that more works should be done in addressing the problems of initialization and the pre-processing transformations of the inferencing algorithm.

\subsection*{2.7 Discussion}

As discussed, DL is based on a logic programming-like languages and it is a simple, efficient but flexible non-monotonic formalism able to deal with many intuitions of non-monotonic reasoning, even with incomplete and/or inconsistent information. It is designed to be easily implementable right from the beginning, and currently, several efficient implementations have been developed. In addition, recently, the logic and its extensions have attracted significant interest from various research communities, especially in the area of knowledge representation, distributed reasoning, multi-agents systems and legal reasoning.

\textsuperscript{10}A detailed descriptions of the test theories can be found in Appendix A.
2.7.1 Comparison of DL with other approaches

Antoniou et al. [17] has compared the differences between DL and sceptical Logic Programming without Negation as Failure (LPwNF). Their results have showed that DL has a strictly stronger expressive power than LPwNF as the latter logic fails to capture the idea of teams of rules supporting a specific conclusion.

Defeasible Logic Programming (DeLP) [77] is a formalism that combines results of Logic Programming and Defeasible Argumentation. It provides the possibilities of representing information in the form of defeasible and strict rules in a declarative manner, and a defeasible argumentation inference mechanism for warranting the entailed conclusions. Compared to DL, DeLP does not support the use of defeaters. Moreover, instead of using superiority relation, DeLP deploys dialectical analysis over proof trees of conflicting conclusions, which has imposed detrimental effect on the results of its complexity, especially in the presence of contradictory conclusions. In [54] the authors have evaluated the complexity of DeLP under different conditions and showed that given a logic program composed of a finite set of ground facts, strict and defeasible rules, the complexity of DeLP can vary from P-complete to NP-completed.

Compared to other non-monotonic logic formalisms, such as Courteous logic programs [110, 215], DL distinguishes between strict and defeasible conclusions, and includes the notions of defeater while courteous logic programs do not. In addition, while the atom dependency graph of a courteous logic program must be acyclic, the same does not apply to DL.

Rock and Billington [186] have formalized an extension of DL called Plausible Logic that overcomes its inability to represent or prove disjunctions. Much like DL, a plausible reasoning situation is defined by a plausible description, which consists of a set of facts, plausible rules and defeasible rules. The set of facts describes the known indisputable truths about the context that it represent. The set of plausible rules describe reasonable assumptions, typical truths, rules of thumb, and default knowledge, which may have a few exceptions. The set of defeater rules disallow conclusions which are too risky, without supporting the negation of the conclusion. Lastly, a priority relation, $>$, on the set of rules allows representation of (acyclic) preferences among rules.

It is stated in their paper that their implemented system, Phobos [184], is capable of deducing or proving formulae in conjunctive normal form at three different levels of certainty or confidence (definite, defeasible and support), and can operate with theories consisting of thousands of rules and priorities with a time complexity per sub-goal that is close to linear ($O(N \log N)$ time complexity) with respect to the number of rules in the theory.

2.7.2 Features of DL

2.7.2.1 Provability

As is well-known that there has always been a trade-off between expressiveness and computational complexity. The complexity of the proof theory of DL has been a major obstacle to the
use of the logic. Fortunately, in [150], the authors have explored different intuitions behind the proof theory and concluded that: (i) defeaters do not add to the expressiveness of DL (in terms of conclusions); (ii) the superiority relation does not add to the expressiveness of acyclic DL; and (iii) acyclic defeasible theories are consistent.

Besides, their analysis and results also support the intuition that permitting cyclic defeasible theories does not have practical advantages, which help us to clean up and enhance the understanding of the basic concepts of DL as a tool for logic modeling.

It was Horty [117, 118] who reinvigorated the discussion about the institutions and nature of defeasible reasoning by questioning commonly accepted views about certain non-monotonic reasoning. In particular he argued that floating conclusions (conclusions that are supported by conflicting and equally strong arguments) may not be reasonable patterns in all instances. In response, Prakken [177] shows that: (1) floating conclusions are often be desirable and (2) presented a refinement of his system, allowing the use of default rules to directly block other defaults, which makes it possible to allow, or disallow, floating conclusions, as one wishes. However, Antoniou [11] does not agree on this since, in Horty’s paper, it only says that floating conclusions are not always reasonable, he does not claim them to be always wrong. Besides, Prakken’s proposed approach comes along with a meshing of “knowledge” and “control” (rules stating explicitly that they block other rules), which make the approach less attractive for practical applications. Consequently, Antoniou concludes that we should consider floating conclusions as a feature of defeasible reasoning, instead of a drawback.

On the other hand, Prakken’s paper also pointed out the problem of argument reinstatement in DL, and is still requiring further discussion, as complete rejection of reinstatement may lead to a failure to recognize the zombie arguments\(^{11}\) [177].

DL is neutral about ambiguity blocking and ambiguity propagation. Instead, different variants have been defined to capture both intuitions. It is, in general, possible to justify that both views on ambiguity, and both ambiguity blocking and ambiguity propagation have their own sphere of applicability. There are applications where ambiguity blocking is counterintuitive, as well as applications where ambiguity propagation is counterintuitive, and finally applications that need both. Thus it is expected that a (sceptical) non-monotonic formalism should be able to capture both. To the best of our knowledge, DL is the only formalism able to do that [89].

A recent study [89] has compared DL with the Carneades model of argument [86]. In the paper, the author has studied the formal relations between the inferential aspects of Carneades and DL, and found that, with the exception of scintilla of evidence, the current proof standards in Carneades framework correspond to the ambiguity blocking no team defeat inference mechanism of DL. The interest of this result is twofold: (i) it means that it is possible to use an implementation of DL as the engine for computing acceptability in Carneades; (ii) based on the theoretical results studied in DL, it is possible to shed light on some Carneades features and

\(^{11}\)Zombie arguments: arguments that are not skeptically acceptable, but nevertheless still figure in some extensions and retain a power to counteract other arguments by preventing them from entering into those extensions.
to highlight some possible shortcomings of decision choices behind the current proof standards, and to propose alternative proof standards that can be used in conjunction with other aspects of Carneades.

### 2.7.2.2 Preference Handling

Similar to other rule-based systems, one of the drawbacks with DL is that it is based on a uniform representation of rules where consequences are inferred based on conditionals, whereas in AI and other practical reasoning approaches, other complex structures, such as comparative notions and ordered disjunctions, have been proposed.

To compensate this, while preserving the superiority relation to handle situations related to contradictory or conflicting conclusions, based on the concept of reasoning with *contrary-to-duty* discussed in [91], Dastani and colleagues have established a preference over and within complex conclusions using the operator \( \otimes \) [61, 62]. Intuitively speaking, the reading of a sequence of conclusions like \( a \otimes b \otimes c \) is that \( a \) is preferred, but if \( \neg a \) is the case, then \( b \) is preferred; if \( \neg b \) is the case, given \( \neg a \), then the third choice is \( c \).

**Definition 2.26** (Preference operator). A preference operator \( \otimes \) is a binary operator satisfying the following properties: (1) \( a \otimes (b \otimes c) = (a \otimes b) \otimes c \) (associativity); (2) \( \bigotimes_{i=1}^{n} a_i = (\bigotimes_{i=1}^{k-1} a_i) \otimes (\bigotimes_{i=k+1}^{n} a_i) \) where exists \( j \) such that \( a_j = a_k \) and \( j < k \) (duplication and contraction on the right).

The general idea of degree of preferences and \( \otimes \) formulas over a sequence of conclusions are interpreted as preference formulas as shown in [91], which can be characterized by the following subsumption relation among rules.

**Definition 2.27** (Reparation rule). Let \( r_1 : \Gamma \Rightarrow \bigotimes_{i=1}^{m} a_i \otimes b \) and \( r_2 : \Gamma' \Rightarrow c \) be two goal rules. Then \( r_1 \) subsumes \( r_2 \) iff \( \Gamma \cup \{\neg a_1, \ldots, \neg a_m\} = \Gamma', c = e \) and \( b = (e \otimes (\bigotimes_{k=1}^{n} d_k)), 0 \leq n \).

The following example illustrates subsumption.

**Example 2.3.** \( \text{MakeOrder} \Rightarrow \text{Pay} \otimes \text{PayInterest} \) subsumes \( \text{MakeOrder} \Rightarrow \text{Pay} \). Moreover, \( \text{MakeOrder} \Rightarrow \text{Pay} \otimes \text{PayInterest} \) subsumes \( \text{MakeOrder}, \neg \text{Pay} \Rightarrow \text{PayInterest} \).

In [61, 62], Dastani et al. have further extended this feature to cover all motivational components, and is characterized by the following definition.

**Definition 2.28** (Modalised reparation rule). Let \( r_1 = \Gamma \Rightarrow X \bigotimes_{i=1}^{n} a_i \otimes b \) and \( r_2 = \Gamma' \Rightarrow c \) be two goal rules. Then \( r_1 \) subsumes \( r_2 \) iff \( \Gamma \cup \{\neg a_1, \ldots, \neg a_m\} = \Gamma', c = e \) and \( b = (e \otimes (\bigotimes_{k=1}^{n} d_k)), 0 \leq n \).

\(^{12}\)Note that a similar approach, but with different motivation, has been proposed in the context of logic programming by Brewka et al. in their logic of ordered disjunction [45].
For example, suppose we have a rule for obligation such as $a \Rightarrow_{\text{OBL}} b \otimes c$: if $a$ is given, it says that $b$ is obligatory; but if $\neg b$ is given, then $c$ is obligatory. A similar intuition also applies to other types of rules.

Note however that the above definitions on preference operators and reparation rule can apply only on defeasible rules since, in DL, facts are indisputable statements and strict rules are inferred in classical sense.

Another study related to preference handling with DL was conducted by Governatori et al. [109, 108]. In the two seminal papers, the authors have undergone a systematic investigation on how to modify the superiority relation in a defeasible theory to change the conclusions of the theory itself, i.e., to make a previously provable (respectively non-provable), non provable (respectively provable). The authors have argued that this approach is applicable to legal reasoning where users, in general, cannot change the facts and rules, but can propose their preferences about the relative strength of the rules according to different situations.

### 2.7.2.3 Reasoning algorithm

Query-based algorithms compute answers for one particular argument, whether such answers are yes or not, defence sets or full extensions; while total answer algorithms compute answers for all arguments and defence sets. The decision as to which of the two types is most appropriate depends on the reasoning scenario. As mentioned in [48], total conclusion algorithms may be relevant to software application deployed into small-scale and relatively static environment such that the best decision can be inferred based on the current contextual information; while query-based algorithms are more appropriate to be implemented in a highly dynamic and complex environment as some information may not be known in advance. Nevertheless, one advantage of query-based algorithm over total algorithm is that it can, in principle, take on the task of a total algorithm by simply enumerating all arguments and query each argument as it is enumerated. However, this may impose some inefficiency to the system as some conclusions derived may need to recompute again every time a query is performed.

As discussed in the previous sections, Delores is the only defeasible reasoner implementation that can compute conclusions for all arguments. Existing implementations rely on either the transformations-based approach as shown in Figure 1.1, or on translations to other formalism or extending logic programming, which would result in either introducing a vast amount of unnecessary propositions/rules to the theory, or losing certain representation properties in some variants.

To tackle this gap, in the next chapter, a “hybrid” approach to these problems - using transformations to transform the theory to the regular form and eliminate the defeaters as before, and then derive the total conclusions of the transformed theory using our new inferencing algorithm, will be presented. This hybrid approach marks the introduction of a new notions – superiority rule chain and inferiorly defeated rule, which will also be discussed subsequently. Last but not least, in the next chapter, we will also present the algorithms that can be used
in deriving the conclusions of defeasible theories under ambiguity propagation variant, well-founded semantics, and their combinations.
Chapter 3

Inferencing with Defeasible Theory

Inferencing is the process of deriving logical conclusions from a set of theorems or axioms that is assumed to be true. It is an important skill that people rely on in providing reasoning services. In this chapter we focus upon the problem of derivability of rules in Defeasible Logic.

The next two sections will provide some background information and the current approach about inferencing a defeasible theory. We will briefly describe the theories behind and the reasons of why a new approach is necessary. Then we will introduce the notion of *inferiorly defeated rules* and present the new algorithm that we devise in sections 3.3 and 3.4, respectively. Followed by the algorithms that we devised to inference defeasible theory under ambiguity propagation and well-founded variants.

Since the provability of literals in strict rules are interpreted as in the classical sense, to avoid overly repetitive, the focus of this chapter will be put mainly on the defeasible part.

3.1 Background

According to [174], the derivability of inference rules had first been formulated by K. Ajdukiewicz [5] in 1928, long before the general notion of inference rules was examined. As discussed in [85]:

In Hilbert-style logical systems, derivability of rules corresponds to provability of theorems. In formalisms where exceptions to rules are allowed (for instance, in Reiters default logic [182]), the concept of an *extension* generalizes that of the *set of theorems*. Members of such an extension do not have the force of theorems. They may be seen as tentative candidates for theorems only. Taking into account logical aspects of our approach, it belongs to the family of rule-based formalisms, where formulas play an auxiliary role only.

Designing a computationally efficient algorithm to conduct sceptical reasoning is clearly not a trivial task. In the past two decades, most works on non-monotonic reasoning have been focused on languages that are not tractable. For example, as discussed in [151], sceptical
default reasoning is $\Pi_2^p$-hard, even for very simple classes of default rules, and the same applies for sceptical autoepistemic reasoning and propositional circumscription. The complexity of sceptical inference from LPwNF and the Clark completion, sceptical inference is co-NP-hard [87, 51]. Although such languages are very expressive and have been explored in many different domains, due to their complexities, they have not led to any practical applications in non-monotonic reasoning.

On the other hand, extensive work on inheritance networks with exceptions has led to polynomial time algorithms [205] and applications [165]. Defeasible logic, as a generalization of inheritance networks with exceptions under the directly sceptical reasoning, has replaced the implicit specificity relation by explicit, programmable priority relation, generalizes containment statement between concepts and their complements to rules over literals; and adds the notion of an explicit defeater [151]. In addition, the proof theories described in Chapter 2 also provide the basis for a top-down (backward-chaining) implementation of the logic, which is what many defeasible querying systems are based on.

Furthermore, Maher and Governatori [153] have provided a bottom-up definition of the logic and have formulated the first-order defeasible logic as a recursively enumerable inference problem (cf. section 2.1.4). In [151], Maher proposed an algorithm for computing the conclusions of a defeasible theory and showed that inference in the propositional form of defeasible logic can be performed in linear time. This contrasts markedly to other NP-hardness or even undecidability non-monotonic or monotonic logics [176, 80].

### 3.2 The Linear Algorithm

The core of Maher’s algorithm (a.k.a. the *Delores* algorithm) is a transitions system (for a subset of defeasible logic) that progressively simplifies a defeasible theory while accumulating conclusions that can be inferred from the theory. The computation of the positive conclusions is based on forward chaining; while the negative conclusions are derived by a dual process.

#### 3.2.1 Problems of present approach

However, the transition system is proved correct only for *basic defeasible theories* - defeasible theories that involve no superiority relations and no defeaters, and with duplicated strict rules\(^1\). To bridge this gap, a series of transformations are used to convert the input theories into equivalent theories which have empty superiority relations, and neither defeaters nor facts.

Antoniou et al. [18] described three transformations that can be used to convert any defeasible theory into an equivalent basic defeasible theory (Figure 1.1). The first places the defeasible theory into a regular form. This establishes that every literal is defined either by strict rules, or by one strict rule and other non-strict rules, and no strict rule participates in the superiority

\(^1\)This is the reason why the algorithms presented in the previous sections require no superiority relation in the input theory.
relation $\triangleright$. The second eliminates defeaters, and the third reduces the superiority relation to the empty relation.

Formally, a transformation is a mapping from defeasible theories to defeasible theories. Recall that $D_1 \equiv D_2$ iff $D_1$ and $D_2$ have the same consequences; similarly, $D_1 \equiv_{\Sigma} D_2$ means that $D_1$ and $D_2$ have the same consequence in the language of $\Sigma$. A transformation is correct if the transformed theory has the same meaning as the original theory. That is, in mathematical terms, a transformation $T$ is correct iff for all defeasible theories $D$, $D \equiv_{\Sigma} T(D)$, where $\Sigma$ is the language of $D$.

**Definition 3.1.** A transformation $T$ is incremental iff for all defeasible theories $D_1$ and $D_2$, $T(D_1 \cup D_2) \equiv_{\Sigma} T(D_1) \cup T(D_2)$, where $\Sigma$ is the union of language of $D_1$ and $D_2$.

**Definition 3.2.** A transformation $T$ is modular iff for all defeasible theories $D_1$ and $D_2$, $D_1 \cup D_2 \equiv_{\Sigma} D_1 \cup T(D_2)$, where $\Sigma$ is the union of the language of $D_1$ and $D_2$.

Essentially modularity says that, independent of its contexts, a transformation can be applied to each unit of information in the theory without modifying the meaning of the theory as a whole. And incrementality says that transformation can be performed on a bit-by-bit basis, and an update in the original theory should have cost proportional to the change, without the need to transform the entire updated theory anew.

**Proposition 3.3.** If a transformation is modular then it is correct and incremental.

*Proof.* Taking $D_2 = \emptyset$ in the definition of modularity, we have $D_1 \equiv_{\Sigma} T(D_1)$ which expresses correctness. By modularity $T(D_1) \cup T(D_2) \equiv_{\Sigma} D_1 \cup T(D_2)$, again by modularity $D_1 \cup T(D_2) \equiv_{\Sigma} D_1 \cup D_2$, then by correctness $D_1 \cup D_2 \equiv_{\Sigma} T(D_1 \cup D_2)$; therefore $T(D_1) \cup T(D_2) \equiv_{\Sigma} T(D_1 \cup D_2)$ which expresses incrementality. 

However, in general, the inverse of Proposition 3.3 does not hold. That is, there are correct and incremental transformations that are not modular.

It is instructive to compare the concepts of correctness and modularity. Correctness means that the original and transformed theories have the same conclusions in a particular viewpoint. It is expected that the given knowledge is considered independently from its context or current changes. While, on the other hand, modularity requires equivalent behavior in any context the theories may be placed in, which is more in line with defeasible, or non-monotonic nature of defeasible logic. And obviously modularity is more stricter condition than correctness, as Proposition 3.3 states.

### 3.2.2 Transformations

Below are the definitions of the transformations (depicted from [18])\(^2\) that are used to transform the theory into regular form, and eliminate all uses of superiority relations and defeaters.

\(^2\)It is beyond the scope of this thesis to present the correctness of the transformations. Readers interested please refer to [18] for details.
Definition 3.4 (Regular form transformation). Consider a defeasible theory \( D = (F, R, >) \), and let \( \Sigma \) be the language of \( D \). We define regular\( (D) = (\emptyset, R', >) \), where \( R' \) is defined below.

Let \( ' \) be a function which maps propositions to new (previously unused) propositions, and rule names to new rule names. We extend this, in the obvious way, to literals and conjunctions of literals.

\[
R' = R_d \cup R_{dft} \cup \\
\{ \rightarrow f' \mid f \in F \} \cup \\
\{ r' : A' \rightarrow C' \mid r : A \rightarrow C \text{ is a strict rule in } R \} \cup \\
\{ r : A \Rightarrow C \mid r : A \rightarrow C \text{ is a strict rule in } R \} \cup \\
\{ p' \rightarrow p \mid A \rightarrow p \in R \text{ or } p \in F \}.
\]

The rules derived from \( F \) and rules \( p' \rightarrow p \) are given distinct new names.

Definition 3.5 (Defeaters elimination). Let \( D = (F, R, >) \) be a defeasible theory, and let \( \Sigma \) be the language of \( D \). Define \( \text{elim}_{dft}(D) = (F, R', >') \) where

\[
R' = \bigcup_{r \in R} \text{elim}_{dft}(r)
\]

and

\[
\text{elim}_{dft}(r) = \left\{ \begin{array}{ll}
    r^+: \{ A(r) \rightarrow p^+, r^- : A(r) \rightarrow \neg p^-, r : p^+ \rightarrow p \} & r \in R_s[p], \\
    r^-: \{ A(r) \rightarrow p^-, r^+: A(r) \rightarrow \neg p^+, r : p^- \rightarrow \neg p \} & r \in R_s[\neg p], \\
    r^+: \{ A(r) \Rightarrow p^+, r^- : A(r) \Rightarrow \neg p^-, r : p^+ \Rightarrow p \} & r \in R_d[p], \\
    r^-: \{ A(r) \Rightarrow p^-, r^+: A(r) \Rightarrow \neg p^+, r : p^- \Rightarrow \neg p \} & r \in R_d[\neg p], \\
    r : \{ A(r) \Rightarrow \neg p^\} & r \in R_{dft}[p], \\
    r : \{ A(r) \Rightarrow \neg p^\} & r \in R_{dft}[\neg p].
\end{array} \right.
\]

The superiority relation \( >' \) is defined by the following condition:

\[
\forall r', s' \in R'(r' >' s' \iff \exists r, s \in R, r' \in \text{elim}_{dft}(r), s' \in \text{elim}_{dft}(s), r > s
\]

where \( r \) and \( s \) are conflicting.

Definition 3.6 (Superiority relations elimination). Let \( D = (\emptyset, R, >) \) be a regular defeasible theory. Let \( \Sigma \) be the language of \( D \). Define \( \text{elim}_{sup}(D) = (\emptyset, R', \emptyset) \), where

\[
R' = R_s \cup \{ s^+: \neg \text{inf}^+(r_1) \Rightarrow \text{inf}^+(r_2), \\
                   \neg \text{inf}^+(r_1) \Rightarrow \text{inf}^-(r_2) \mid r_1 > r_2 \} \cup \\
    \{ r_a : A(r) \Rightarrow \neg \text{inf}^+(r), \\
                   \neg \text{inf}^+(r) \Rightarrow p \mid r \in R_d[p] \} \cup \\
    \{ r_a : A(r) \Rightarrow \neg \text{inf}^-(r), \\
                   \neg \text{inf}^-(r) \Rightarrow p \mid r \in R_{dft}[p] \}.
\]

It is clear that these transformations are designed to provide incremental transformation to
the input theories, and systematically introduce new literals and rules to emulate the features removed. Below are some important properties of the transformation, depicted from [18].

**Proposition 3.7.** The transformation regular is incremental, but not modular.

**Proof.** It is immediate to see that for every pair of defeasible theories $D_1, D_2$, $\text{regular}(D_1) \cup \text{regular}(D_2) = \text{regular}(D_1 \cup D_2)$. Consequently regular is incremental.

To see that regular is not modular, consider $D_1 = \{a \rightarrow b\}$ and $D_2 = \{\rightarrow a\}$. Then $\text{regular}(D_1) = \{a \Rightarrow b, a' \rightarrow b', b' \rightarrow b\}$. Clearly $D_1 \cup D_2 \vdash +\Delta b$. However, $\text{regular}(D_1) \cup D_2 \vdash -\Delta b$, since there is no fact $a'$.

**Proposition 3.8.** The transformation elim_dft is correct for well-formed regular theories.

**Proposition 3.9.** The transformation elim_dft is incremental, but not modular.

**Theorem 3.10.** The transformation elim_sup is modular for separated well-formed regular defeasible theories. That is, for such theories $D_1$ and $D_2$, $D_1 \cup D_2 \equiv \Sigma D_1 \cup \text{elim sup}(D_2)$, where $\Sigma$ is the union of the languages of $D_1$ and $D_2$.

**Proposition 3.11.** The transformation elim_sup is correct for well-formed regular theories.

**Example 3.1.** Consider the following defeasible theory:

- $r_1 : \Rightarrow a$
- $r_2 : \Rightarrow \neg a$
- $r_2 > r_1$

The transformed theory looks as follows:

- $r_1.a : \Rightarrow \neg \text{inf}^+(r_1)$
- $r_2.a : a \Rightarrow \neg \text{inf}^+(r_1)$
- $r_1.c : \neg \text{inf}^+(r_1) \Rightarrow a$
- $r_1'.c : \neg \text{inf}^+(r_1') \Rightarrow a$
- $r_2.a : \Rightarrow \neg \text{inf}^+(r_2)$
- $r_2'.a : \Rightarrow \neg \text{inf}^+(r_2')$
- $r_2.c : \neg \text{inf}^+(r_2) \Rightarrow \neg a$
- $r_2'.c : \neg \text{inf}^+(r_2') \Rightarrow \neg a$

and

- $s_1^+ : \neg \text{inf}^+(r_2) \Rightarrow \text{inf}^+(r_1)$
- $s_1^- : \neg \text{inf}^+(r_2) \Rightarrow \text{inf}^-(r_1)$

It is obvious from the example above, and also discussed in [151], that even though such transformations can be applied in one pass, they are profligate in their introduction of new proposition and generation of rules, which would poorly affect the performance of the inference process by a maximum factor of 12.

Moreover, despite the fact that the transformation works absolutely fine in the ambiguity blocking variant, it is not the case in other variants. For instance, consider again the theory as showed in example 3.1. Under ambiguity propagation variant, the conclusions derived should
be \( \{ -\partial_a p, +\partial_a \neg a, -\partial_c p, +\partial_c \neg c \} \). However, in the transformed theory, as the superiority relation is removed, the support of \( a \) in \( r_1.c \) cannot be blocked, which subsequently propagates and supports the conclusions of \( r_1.a \) and \( r_1.c \). Hence the conclusions derived in the transformed theory become \( \{ -\partial_a p, +\partial_a \neg a, -\partial_c p, -\partial_c \neg c \} \), meaning that not all representation properties of the defeasible theory can be preserved during the transformations. What makes things worse is that the example showed is not a special case. This kind of improper support propagation will appear in every transformed theory if there exists a superiority relation in their original theory.

**Theorem 3.12.** [18] Let \( D = (F, R, >) \) be a defeasible theory. Then in general there is no modular transformation \( T \) such that \( T(D) = (F', R', \emptyset) \).

**Proof.** Let us consider the defeasible theory \( D \) consisting of

\[
\begin{align*}
  r_1 : & \Rightarrow p \\
  r_2 : & \Rightarrow \neg p \\
  r_1 : & > r_2
\end{align*}
\]

We partition \( D \) into \( D_1 = \{ r_1 : \Rightarrow p, r_2 : \Rightarrow \neg p \} \) and \( D_2 = \{ r_1 : > r_2 \} \). Let us suppose that a modular transformation \( T \) removing the superiority relation exists. According to the definition of modularity we have \( D \equiv \Sigma D_1 \cup T(D_2) \). It is easy to see that \( D \vdash +\partial p \). Since \( D_1 \cup T(D_2) \) contains an applicable rule for \( \neg p \) (i.e., \( r_2 \)) and the superiority relation is empty, \( D_1 \cup T(D_2) \vdash -\partial p \). But then \( D \not\equiv \Sigma D_1 \cup T(D_2) \), which contradicts our assumption.

So, regrettably, not all the transformations are inside a great way: some of the transformations are not modular and can only be applied in a limited context. Theorem 3.12 states that it is, in general, impossible to find a modular transformations such that any defeasible theories with superiority relations in the content can be transformed into another defeasible theories without superiority relations.

With above (Proposition 3.7 to 3.9 and Theorem 3.12), it is important to point out that the correctness of transformations presented in [18] can be guaranteed only under the “standard” semantics of defeasible logic, i.e., the ambiguity blocking variant of defeasible logic, which subsequently leads to a situation that the inference algorithm devised by Maher cannot be used to inference defeasible logic in general. To recap, the theory presented in Example 3.1 already captured the inability of Maher’s algorithm in deriving the conclusions under ambiguity propagation variant of DL. Thus, a new approach that is able to handle the inferencing process of different variants of defeasible theory is need.

In logic programming terms, there are two ways to resolve this problem. One approach is to devise a new transformations system and utilize the Maher’s inference algorithm as before. However, we may suffer from the same problem (as showed in Example 3.1) as finding a modular...
transformation to eliminate superiority relations from a defeasible theory is, in general, not possible, and may result in losing some of the theories’ representation properties which would subsequently disallow us to block the support of literals due to superiority relations. Another approach is to devise/modify the inference algorithm such that it is capable of handling all representational properties of DL even without performing any transformation to the theory. However, doing this may result in making the inferencing algorithm more complex, which may impose negative impact to the complexity.

To this end, by introducing the notion of Inferiorly Defeated Rule below, we devised a “hybrid” approach to resolve the problem by incorporating the superiority relations in the inference process. Compared to the algorithm proposed by Maher, our algorithm requires only transforming original defeasible theory into an equivalent theory without defeaters, which significantly reduce the number of proposition introduced and rules generated.

In addition, our approach preserves all representation properties in all variants of DLs. For instance, as discussed before, the problem discussed in Example 3.1 is due to the fact that the support of literal \( a \) cannot be blocked after the removal of the superiority relation \( (r_2 > r_1) \). With our approach, as the superiority relation is in place during the inference process, the support of literals in the inferior rule can be blocked accordingly.

### 3.3 Inferiorly defeated rules

As we mentioned, in DL, the superiority relation is used to define the preference, or relative strength, of rules, i.e., it provides information about which rules can overrule other rules. Furthermore, in some variants, such as the ambiguity propagation variant, the superiority relation also provides information on whether conclusions of rules are supported or not. Based on these, we introduce the notion of superiority chain.

**Definition 3.13.** A superiority chain is a superiority relation hierarchy such that, apart from the first and last element of the chain, there exists a superiority relation between rules \( r_k \) and \( r_{k+1} \):

\[
r_1 > r_2 > \cdots > r_n
\]

where \( n \) is the length of the chain, and \( C(r_k) = \neg C(r_{k+1}) \), \( \forall 1 \leq k < n \).

One point to note here is that the superiority relation, in general, is not transitive, meaning that, unless otherwise specified, there exists no superiority relation between a rule \( r \) and another rules in the chain. For instance, consider the theory in example 3.2, \( r_1 \) and \( r_4 \) are in the same superiority chain but \( r_1 \) is not superior to \( r_4 \), so the consequence of \( r_1 \) cannot be used to overrule the consequence of \( r_4 \).

**Definition 3.14.** Let \( D = (\emptyset, R, >) \) be a defeasible theory (in regular form) over a language
Then, a rule $r \in R_{sd}[q]$ is inferiorly defeated if:

$$\exists s \in R_{sd}[-q] \text{ s.t. } A(s) = \emptyset \text{ and } s > r.$$ 

This means that if $r$ is a rule in $R_{sd}[q]$ and there exists another rule $s$ in $R_{sd}[-q]$ such that $A(s)$ is empty and $s > r$, then irrespective to whether $r$ is derivable or not, the conclusion of $r$ is going to be overruled by $s$. For illustration, consider again the theory in Example 3.1. Since $r_1$ and $r_2$ are both derivable and $r_2$ is superior than $r_1$, $r_1$ is inferiorly defeated and the conclusion of $r_1$ is overridden by $r_2$. In other words, under this situation, $r_1$ is redundant and cannot be used to derive any positive conclusion. So removing it from the theory and falsifying its conclusion does not affect the conclusions derived. And the same applies even when $A(r_1) \neq \emptyset$.

However, the example above is just oversimplified. Consider the example below:

**Example 3.2.** Let us consider the defeasible theory $(D)^4$:

\[
\begin{align*}
  r_1 : & \Rightarrow a \\
  \lor \\
  r_2 : & \Rightarrow \neg a \\
  \lor \\
  r_3 : & \Rightarrow a \\
  \lor \\
  r_4 : & \Rightarrow \neg a
\end{align*}
\]

All rules above are applicable but $r_2$, $r_3$ and $r_4$ are inferiorly defeated. So, $r_1$ is the only rule that can be used to derive positive conclusion and the conclusions inferred should be $+\partial a, -\partial \neg a$. However, if we remove an inferiorly defeated rule arbitrary, say $r_3$, then the theory will become $(D')$:

\[
\begin{align*}
  r_1 : & \Rightarrow a \\
  \lor \\
  r_2 : & \Rightarrow \neg a \quad \text{and} \quad r_4 \Rightarrow \neg a
\end{align*}
\]

So, in this case, only $r_2$ is inferiorly defeated and the conclusions derived will become $-\partial a, -\partial \neg a$, meaning that $D \neq D'$. Hence that a rule is inferiorly defeated is not an adequate condition to enable it to be removed from the theory without changing the conclusions. If we take into account a line of superiority chain, then additional conditions are needed.

### 3.4 New Inference algorithm

To recap, our main focus lies in characterizing a set of inferiorly defeated rules that cannot be used to derive positive conclusions and can be removed from the theory without changing its conclusions. Belows are the intuitions behind our algorithm:

---

4Here, for better graphical representation and easier understanding, we overloaded the symbol $\lor$ to denote the superiority relation between two rules. That is, in the first part of the example, we have: $r_1 > r_2 > r_3 > r_4$. 

---
If a rule \( r \in R_{sd}[q] \) is inferiorly defeated, according to the definition, there exists a rule \( s \in R_{sd}[-q] \) such that \( A(s) = \emptyset \) and \( s > r \). So, the conclusion of \( r \) will be overruled by \( s \) irrespective to whether the conclusion of \( r \) is derivable or not. In addition, if \( \exists t \in R_{sd}[-q] \) so that \( r > t \), i.e., there exists no rule such that \( r \) is superior to, then by the irreflexive nature of superiority relation, \( r \) is redundant and can be removed from the theory without affecting its conclusions.

However, before we go into the details of our theorem, based on the previous section, we need to introduce the following definition.

**Definition 3.15.** Let \( D = (\emptyset, R, >) \) be a defeasible theory (in regular form) over a language \( L_D \). Then, \( R_{infd} \subset R \) is the set of inferiorly defeated rules in \( D \) such that the number of weaker rules is equal to zero. That is, \( \forall r \in R_{infd}[q], \exists s \in R[-q] \text{ s.t. } r > s \)

**Theorem 3.16.** Let \( D = (\emptyset, R, >) \) be a defeasible theory (in regular form), and \( r \in R_{infd}[q] \) with number of weaker rules equals to zero. Let \( D' = (\emptyset, R \setminus \{r\}, >') \) be the reduct of \( D \) with response to \( r \), denoted by \( \text{reduct}(D) \), where \( >' \) is defined as:

\[
> \setminus \{s > r\} : \exists s \in R_{sd}[-q], \ A(s) = \emptyset \text{ and } s > r
\]

Then \( D \equiv D' \).

In addition, \( -\partial q \) can be derived if \( R[q] = \emptyset \) after the removal of \( r \).

**Proof.** (sketch, details of the proof can be found in Appendix B)

We prove the theorem by induction on the length of derivations \( P \) in \( D \).

The key point of the inductive step is to consider all possible cases, thus if the rule we removed is for literal \( q \) we have to consider the cases for \(+\partial q\), \(+\partial \neg q\), \(-\partial q\) and \(-\partial \neg q\).

For \( \Rightarrow \) we have to show that the introduction of defeated rules do not alter the conclusions, while for \( \Leftarrow \) we have to show that the removed rules are redundant and we can obtain the same results without them. In both cases we can show this reasoning by contradiction.

This theorem looks simple but it lies the condition on differentiating the set of necessity rules that are required in the inference process from the set of rules that are redundant and can be removed from the theory. From the theorem, we can conclude that if an inferiorly defeated rule is the weakest rule along the line of superiority rule chain, i.e., there exists no rule weaker than it, and removing it from the theory will not cause any undesired side effects to the inference process.

**Example 3.2** (continuing from p. 52). By applying Theorem 3.16 recursively, the rules \( r_4, r_3 \) and \( r_2 \) will be removed subsequently leaving \( r_1 \) as the only rule in the superiority rule chain. So the conclusions derived become \(+\partial a, -\partial \neg a\) as expected.
Algorithm 3.1 (ComputeDefeasible) below shows the algorithm for deriving conclusions from a defeasible theory. In the algorithm we assume that the input theory \( D \) is in regular form and with no defeaters. These assumption involve no loss of generality; they can be obtained by the preprocessing transformations that we have discussed in section 3.2.2. We can generate all positive definite consequences (\( +\Delta \)) of strict rules and facts by straightforward rule application (essentially unit resolution). The negative consequences (\( -\Delta \)) then can be derived as the complement of the positive consequences.

The algorithm has a form similar to the inference algorithm proposed in [151] in a sense that it is based on a series of transformations which (1) to assert whether a literal is provable or not; and (2) to progressively reduce and simplify the theory; but with two major differences. The first is that, by incorporating Theorem 3.16 into the algorithm, we choose to treat the superiority relation explicitly; in [151] it is eliminated in a pre-processing step. As discussed before, we do this because that preprocessing step is not valid for all variants, but also to avoid the overhead of pre-processing and the resulting bloating of the theory.

The second difference is that we can compute \( \partial \)-unfounded set, which is a crucial and essential component of computing the consequences of the well-founded variants of DL, and will be discussed further in the later section (cf. Section 3.5.2) of this chapter.

Algorithm 3.1: Inference Algorithm for \( +\partial \) and \( -\partial \)

**Algorithm:** ComputeDefeasible(\( D \))

**Data:** \( HB \): Herbrand base of \( D \), where \( D \) is a defeasible theory in regular form and with no defeaters

**Data:** \( r_{sup} = \{ s : s > r \} \), \( r_{inf} = \{ s : r > s \} \)

1. repeat
2. \( \partial^+ = \emptyset \), \( \partial^- = \emptyset \)
3. foreach \( q \in HB \) do
4.   if \( R[q] = \emptyset \) then
5.     \( \partial^- = \partial^- \cup \{ q \} \)
6.     \( R = R \setminus \{ r \in R : q \in A(r) \} \)
7.     \( \{ > \} = \{ > \} \setminus \{(r,s),(s,r) \in > : q \in A(r)\} \)
8.     if \( \exists r \in R[q] : A(r) = \emptyset \) then
9.       if \( r_{sup} = \emptyset \) then
10.      \( \partial^- = \partial^- \cup \{ \neg q \} \)
11.     \( R = R \setminus \{ r \in R : \neg q \in A(r) \} \)
12.     \( \{ > \} = \{ > \} \setminus \{(r,s),(s,r) \in > : \neg q \in A(r)\} \)
13.     if \( r \in R_{sd}[q] \) and \( R[\neg q] \setminus R_{infd} \subseteq r_{inf} \) then
14.       \( \partial^+ = \partial^+ \cup \{ q \} \)
15.       \( R = \{ A(s) \setminus \{ q \} \Rightarrow C(s) : A(s) \Rightarrow C(s) \in R \} \)
16.     else
17.       \( R_{infd} = R_{infd} \cup r_{inf} \)
18.     end
19.   end
20. until (\( \partial^+ = \emptyset \) and \( \partial^- = \emptyset \)) or \( R = \emptyset \)
In the algorithm, two global variables, $+\partial$ and $-\partial$, hold set of literals representing the conclusions inferred so far and are initialized to $+\Delta$ and $\{l : \sim l \in +\Delta\}$ respectively prior to the start of the process. Two other global variables, $\partial^+$ and $\partial^-$, hold the most recent inferences of the process.

To be able to handle the superiority relations explicitly, we need to know the relations between rules. For each rule $r$ with conclusion $q$, we record the set $r_{sup}$ of rules for $\sim q$ that are superior to $r$, and the set $r_{inf}$ of rules that are inferior to $r$. The key concept is that an inferiorly defeated rule: rules for which there is superior rule with empty body, cannot be used to infer a positive conclusion since the superior rule will override it, but they should not simply be removed. The variable $R_{infd}$ accumulates the set of inferiorly defeated rules.

There are several circumstances in which we can infer a new conclusion and/or simplify the theory:

- If there are no rules for a literal $q$ then we can conclude $-\partial q$ (lines 4–7). Consequently, we can safely remove all rules containing $q$ in the body since all rules containing $q$ in the body can never be used to make a positive inference (because of the presence of $q$). When removing rules, we can also remove them from the sets $r_{sup}$ and $r_{inf}$.

- If there is a rule $r$ for $q$ with an empty body and with $r_{sup} = \emptyset$ then we can conclude $-\partial \sim q$ because there is no rule that can override $r$ (lines 8–17). Consequently we remove all rules with $\sim q$ in the body. If, in addition, all rules for $\sim q$ are either inferiorly defeated or inferior to $r$ then we can conclude $+\partial q$. We then remove the literal $q$ from all rules.

**Performance of the algorithm**

The simplifications may spawn more inferences and simplifications, and so on. When these are all exhausted, ComputeDefeasible terminates. With an appropriate choice of data structure (cf. Section 4.1.1.2), the deletion/update of literals, rules and superiority relations (including the numbers of superior rules for each defeasible rules) can be done in time proportional to the number of literals deleted. Besides, we can detect in constant time whether the literal deleted was the only literal in the rule, and whether a rule deleted was the only rule with conclusion $q$ in the theory. Furthermore, each literal occurrence is deleted at most once, and the test of rule with empty body is made at most once per deletion. Similarly, each rule is deleted at most once, and the test for no more rules for literals is made at most once per deletion. As all checking/updating relies on information that is immediately available, the computational cost of the main part of the algorithm is $O(N)$, where $N$ is the number of removals occurrences in $D$.

Here it is worth knowing that the structure of the inference algorithm is similar for all tags, so the algorithm provided here can be easily modified for the other proof tags. In the case of ambiguity propagation or other variants of DL, since the superiority relation is in place during
the inference process, the support of literals can be blocked if the rules are inferiorly defeated and removed from theory.

3.5 Inferencing Defeasible Theory under different variants

In the following sections, we will shift our attention to algorithms that we devised for inferencing defeasible theory with two different variants, namely: ambiguity propagation and well-founded variants.

3.5.1 Ambiguity Propagation

Ambiguity is ubiquitous. It plays a crucial role in the field of KR and automated reasoning, and is a main issue of many non-monotonic formalisms such as default logic, defeasible reasoning, inheritance networks and belief revision [213]. As mentioned in Section 2.2.2, a literal is ambiguous iff there exist two chains of reasoning with one supporting the conclusion \( p \) is true whereas the other supports the conclusion \( \neg p \) is true, and the superiority relation does not resolve this conflict. Whenever such situation exists, due to the sceptical nature of DL, neither \( p \) nor \( \neg p \) can be concluded.

Example 3.3. Consider the following defeasible theory (adapted from [205]) and its inheritance network in Figure 3.1.

\[
\Rightarrow a \quad a \Rightarrow b \quad b \Rightarrow d \quad d \Rightarrow f \quad f \Rightarrow h \quad h \Rightarrow j \\
\quad f \Rightarrow i \quad i \Rightarrow \neg j \\
\quad d \Rightarrow g \quad g \Rightarrow \neg i \\
\quad b \Rightarrow e \quad e \Rightarrow \neg g \\
\quad a \Rightarrow c \quad c \Rightarrow \neg e \\
\]

The superiority relation is empty.

In the theory, it is clear that literal \( a \) determines that literal \( e \) is ambiguous w.r.t. \( a \) (since \( a \Rightarrow b \Rightarrow e \) and \( a \Rightarrow c \Rightarrow \neg e \)). So, in the inheritance network, all edges to and from \( e \) are eliminated as all rules with \( e \) in their bodies become inapplicable. In particular, the edge from \( e \) to \( \neg g \) is eliminated, making \( g \) unambiguous w.r.t. \( a \). This is certainly one possibility. But it is also possible that \( a \) implies \( \neg e \); and if \( a \) implies \( \neg e \), it is unclear that whether \( a \) implies \( g \), i.e., \( a \) might not imply \( g \). So it is, to certain extend, not safe to assume from the ambiguity at \( e \) that the path \( a, b, d, g \) is always true (Figure 3.2a).

Moreover, a more severe anomaly following this one is that ambiguity blocking inheritance computes a kind of “parity” on the number of ambiguities in a path [205]. It is unclear to whether the network in Figure 3.1 is sceptical as to whether \( a \) is-a \( e \) or an \( i \) but supports the conclusions that \( a \) is-a \( g \) and a \( j \). Similar, this net is sceptical about whether \( b \) or \( f \) is-a \( j \) but
allows the path from $a$ and $d$ to $j$, which results in calling into question the intuitiveness of ambiguity blocking inheritance. Nevertheless, it is the default method that DL uses on handling ambiguity.

The work of Horty et al. [119, 120], and Haugh [114], was of great importance for laying the groundwork for the establishment of sceptical inheritance in the field of non-monotonic reasoning. In their papers, Horty and his colleagues argue that an ambiguous line of reasoning should not be allowed to interfere with other potential conclusions, and the line of reasoning should be interrupted as soon as an ambiguity has been reached. Haugh calls it *ambiguity blocking* inheritance.

*Ambiguity propagation*, on the other hand, is used to generate a single extension whose conclusions are consistent. It allows the ambiguous line of reasoning to proceed and conclusions can only be drawn when there are *no* counterarguments; while ambiguity blocking considers only *unambiguous* counterarguments [204]. For example, the cascading ambiguities of Example 3.3 which give ambiguity blocking difficulty, present no problem for ambiguity propagating inheritance (Figure 3.2b). It yields a stricter form of skepticism that results in fewer conclusions being drawn, which makes it preferable when the cost of handling incorrect conclusions is high. See [118] for a discussion of ambiguity propagation.

The ambiguity propagation variant of DL was first introduced in [16], to cater for defeasible reasoning under different situations. To achieve the goal, a new level of provability, called *support* and denoted by $\Sigma$, is introduced (cf. Section 2.2.2). Its aims is to separate the invalidation of a counterargument from the derivation of $\partial$ tagged literals.
Figure 3.2: Results of theory in Example 3.3 under different variants

**Definition 3.17.** A rule $r \in R[q]$ is applicable at step $i+1$ in a derivation $P$ iff $\forall a \in A(r), +\partial a \in P[1..i]$.

**Definition 3.18.** A rule $r \in R$ is discarded in a derivation $P$ iff $\exists a \in A(r), -\partial a \in P[1..i]$.

**Definition 3.19.** A rule $r \in R[q]$ is supported at step $i+1$ in a derivation $P$ iff $\forall a \in A(r), +\Sigma a \in P[1..i]$.

**Definition 3.20.** A rule $r \in R$ is unsupported at step $i+1$ in a derivation $P$ iff $\exists a \in A(r), \Sigma a \in P[1..i]$.

The above definitions [41] instantiate the abstract definition of the proof theory described in Section 2.2.2 to an ambiguity propagation variant of DL by defining what it means for a rule to be applicable, discarded, supported and unsupported. Simply speaking, a rule is applicable iff every literals in the antecedent is defeasibly provable; while a rule is discarded if at least one of the literals in the premises of the rule is not defeasibly provable. On the other hand, a rule is supported iff there is a chain of support for every literals in the antecedent of the rule; while a rule is unsupported iff at least one of the literals in the premises of the rule is not supported.

So, when comparing these to the ambiguity blocking variant of DL, where the notions of applicable and supported coincides (and also the notions of discarded and unsupported, being the negation of the previous notions), in ambiguity propagation these notions are distinct [41]. That is, in ambiguity blocking, a literal $p$ is supported iff it is (definitely/defeasibly) provable. However, in ambiguity propagation, as is reflected in the proof theory (cf. Section 2.2.2), a literal $p$ is supported iff there exists a line of reasoning that would lead us to conclude $p$, irrespective to
whether the literal is provable or not (Proposition 2.9). Clearly, this is an important intuition to elicit as it implies that: when inferring conclusions with ambiguity propagation, we have to consider the support and provability of a literal separately; which may imposes detrimental effects on the computational complexity of our algorithm.

Example 2.1 (continuing from p. 18). As mentioned before, given both quaker and republican as facts, all literals (both positive and negated) are supported. As before, we conclude $-\partial\text{pacificist}$ and $-\partial\neg\text{pacificist}$ since $r_1$ and $r_2$ overrule each other. However, in case of ambiguity propagation we conclude $-\partial ap\neg\text{antimilitary}$, since rule $r_4$ is discarded but not unsupported. Thus a positive proof of $+\Sigma\neg\text{antimilitary}$ is not possible. So overall we have both $-\partial ap\neg\text{antimilitary}$ and $-\partial ap\text{antimilitary}$ (Figure 2.2b).

Last but not least, before moving on to the computation section, it is worth to note that a preference for ambiguity blocking and ambiguity propagation behavior is one of the properties of non-monotonic inheritance nets over which intuitions can clash [207]. Stein [205] has argued that ambiguity blocking, under certain situations, would result in strange pattern of conclusions. However, in a recent study on the relation between Carneades and DL, Governatori [89] has argued that it is possible to justify both views on ambiguity and that both ambiguity blocking and ambiguity propagation have their own sphere of applicability. There are situations where ambiguity blocking is counterintuitive as well as situations where ambiguity propagation is counterintuitive, and finally situations where we need both. Besides, the author has further pointed out that a (sceptical) non-monotonic formalism should be able to accommodate both and it seems that, at this moment, DL is the only formalism that is able to do this.

**Computing Consequences in DL with Ambiguity Propagation**

We can now describe ComputeDefeasibleAP (Algorithm 3.2), the algorithm for inferring conclusions for the ambiguity propagation variant of DL. ComputeDefeasibleAP is based on the algorithm that we presented in computing the ambiguity blocking of DL (Algorithm 3.1). As the algorithm proceed, the theory $D$ is simplified progressively and new conclusions are accumulated, while the superiority relations still in place.

As mentioned before, when inferring conclusions with ambiguity propagation, we have to consider separately the support and provability of the literals. To be able to do this, we duplicate the set of rules in $D$ (where $D$ is a regular form defeasible theory and with no defeaters), and denotes the set for computing support by $R^\sigma$ and the set for computing defeasibly provable conclusions by $R^\delta$.

In addition to the four global variables ($+\partial, -\partial, \partial^+, \partial^-$) that we used to hold the set of conclusions inferred and the conclusions that we derived in each iterative steps, two global variables, $+\Sigma$ and $-\Sigma$, are introduced to hold the sets of literals representing the support (and respectively unsupported) set inferred so far, and are initialized as empty sets prior to the inference process.
Algorithm 3.2: Inference Algorithm for $+\partial_{ap}$ and $-\partial_{ap}$

**Algorithm:** ComputeDefeasibleAP($D$)

**Data:** $HB$: Herbrand base of $D$, where $D$ is a defeasible theory in regular form and with no defeaters

**Data:** $r_{sup} = \{ s : s > r \}$, $r_{inf} = \{ s : r > s \}$

1. repeat
   2. $\partial^+ = \emptyset$, $\partial^- = \emptyset$
   3. foreach $q \in HB$ do
      4. if $R^\sigma[q] = \emptyset$ then
         5. $-\Sigma = -\Sigma \cup \{ q \}$
         6. $R^\sigma = R^\sigma \setminus \{ s \in R^\sigma : q \in A(s) \}$
      7. if $R^\delta[q] = \emptyset$ then
         8. $\partial^- = \partial^- \cup \{ q \}$
         9. $R^\delta = R^\delta \setminus \{ s \in R^\delta : q \in A(s) \}$
        10. $\{ > \} = \{ > \} \setminus \{ (r,s),(s,r) \in > : q \in A(r) \}$
      11. if $\exists r \in R^\sigma[q] : A(r) = \emptyset$ and $r_{sup} = \emptyset$ then
         12. $\partial^- = \partial^- \cup \{ \neg q \}$
         13. $+\Sigma = +\Sigma \cup \{ q \}$
         14. $R^\delta = R^\delta \setminus \{ s \in R^\delta : \neg q \in A(s) \}$
         15. $R^\sigma = \{ A(s) \setminus \{ q \} \Rightarrow A(s) : A(s) \Rightarrow C(s) \in R^\sigma \}$
      16. if $\exists r \in R^\delta[q] : A(r) = \emptyset$ then
         17. $R_{inf} = R_{inf} \cup r_{inf}$
         18. if $r \in R_{sd}[q]$, $r_{sup} = \emptyset$ and $R^\sigma[\neg q] \setminus R_{inf} = \emptyset$ then
            19. $\partial^+ = \partial^+ \cup \{ q \}$
            20. $\partial^- = \partial^- \cup \{ \neg q \}$
            21. $-\Sigma = -\Sigma \cup \{ \neg q \}$
            22. $R^\delta = \{ A(s) \setminus \{ q \} \Rightarrow A(s) : A(s) \Rightarrow C(s) \in R^\delta \}$
            23. $R^\delta = R^\delta \setminus \{ s \in R^\delta : \neg q \in A(s) \}$
            24. $R^\sigma = R^\sigma \setminus \{ s \in R^\sigma : \neg q \in A(s) \}$
            25. $\{ > \} = \{ > \} \setminus \{ (r,s),(s,r) \in > : \neg q \in A(r) \}$
            26. $+\partial_{ap} = +\partial_{ap} \cup \partial^+$, $-\partial_{ap} = -\partial_{ap} \cup \partial^-$
      27. until $(\partial^+ = \emptyset \text{ and } \partial^- = \emptyset)$ or $(R^\delta = \emptyset \text{ and } R^\sigma = \emptyset)$
The theory simplification and reduction processes in ComputeDefeasibleAP are essentially the same as what we have presented in Algorithm 3.1, except that different inference conditions are used and additional steps have been added to cater the derivation of the support and unsupport sets.

The case for inferencing negative conclusions is relatively simple. From the proof theory and the inclusion theorem [41] we know that a literal $q$ is not supported or defeasibly provable if there are no rules for $q$ and we can conclude $-\Sigma q$ and $-\partial q$ accordingly (Lines 4–10). Consequently those rules with $q$ in their antecedent become inapplicable and can safely be removed since all rules containing $q$ can never be used to make a positive inference.

However, the case for positive provability is a bit different. For a literal $q$, before inserting $+\partial q$ to the conclusions set ($\partial^+$) we have to verify whether: (1) the conclusion is a conclusion inferred by an inferiorly defeated rule which will be overruled by some superior rules; or (2) its counterpart is supported (Lines 16–25). In either case, we should not conclude $+\partial q$ and a negative conclusion ($-\partial q$) should be added to the negative conclusion set ($\partial^-$) in the next iteration if there exists no rules for $q$ anymore. Otherwise, we should add $+\partial q$ to the conclusions set and simplify the theory accordingly.

Lines 11–15 in ComputeDefeasibleAP is used to infer literals that may appear in the positive support set, which should be checked independently with the (positive) provability that we discussed above.

Performance of the algorithm

The discussion above shows how defeasible conclusions can be derived from a defeasible theory, which is a bit tedious. However, when compare to the algorithms proposed in [143] which separate the process of computing the support/unsupport-set ($+\Sigma$, $-\Sigma$) with the sets of ($+\partial$, $-\partial$), the algorithm presented here updates all sets values simultaneously.

As can be seen, each operations (including deletion/update of literals, rules and superiority relation) presented here is of very similar to the one presented in ComputeDefeasible, which relies on information that is immediately available. The major difference between the two is that for each literal occurrences in ComputeDefeasibleAP, instead of performing the update/deletion operations in at most once per each iterations, we have to do it twice: one for processing information in the support set and another for processing information in the provability set. So, in terms of computational complexity, it is at most doubling the amount of that in ComputeDefeasible to $O(2N)$, where $N$ is the number of removals occurrences in $D$.

3.5.2 Well-Founded Semantics

Introduced by [209] the well-founded semantics is a fixpoint semantics which was originally developed to provide reasonable interpretation of logic programming with negation, but has since been applied to extended logic programs such as non-monotonic reasoning. It is a sceptical approximation of answer set semantics such that every well-founded consequences of a logic
program $P$ is contained in every answer set of $P$. Whilst some programs are not consistent under answer set semantics, well-founded semantics assigns a coherent meaning to all programs.

**Example 3.4.** Consider the following defeasible theory:

$$
\begin{align*}
 r_1 & : \Rightarrow \text{doResearch}(\text{John}) \\
 r_2 & : \text{doResearch}(X) \Rightarrow \text{publishPapers}(X) \\
 r_3 & : \text{publishPapers}(X), \text{teachAtUni}(X) \Rightarrow \text{professor}(X) \\
 r_4 & : \text{professor}(X) \Rightarrow \text{doResearch}(X) \\
 r_5 & : \text{professor}(X) \Rightarrow \text{teachAtUni}(X) \\
 r_6 & : \text{teachAtUni}(X) \Rightarrow \text{highReputation}(X) \\
 r_7 & : \Rightarrow \neg \text{highReputation}(X)
\end{align*}
$$

Given that a person John who does research at university, we would like to ask if John is a professor. Now, in order to derive $\text{professor}(\text{John})$ we must derive $\text{publishPapers}(\text{John})$ and $\text{teachAtUni}(\text{John})$. When trying to derive $\text{teachAtUni}(\text{John})$ we need to check $\text{professor}(\text{John})$. And we enter in an infinite loop. Consequently neither could we show $\text{highReputation}(\text{John})$.

The reason of why DL fails to derive any useful conclusions from the above theory is because it does not detect that literals ($\text{professor}(X)$ and $\text{teachAtUni}(X)$) in rules $r_3$ and $r_5$ are relying on each other and can never be applied. However, as commented in [153], it is desirable for a logic to recognize such “loops” and derive conclusions based on some intuitions.

As mentioned in Section 2.2.3, a well-founded defeasible logic (WFDL) was introduced to handle such cases. Unlike the standard DL in which inference is determined, not only by the inference rules, but also using unfoundedness as the notion of failure. That is, the inferences rules for provability in WFDL are exactly the same as in DL; while the inferences rules for unprovability, however, extend those of DL by the identification of unfounded sets, representing sets of literals that are either directly unprovable in the sense of DL, or unprovable but cyclically dependent on each other. In the latter case WFDL detects unprovability that is not detected by DL. Thus the central definitions in WFDL are those of a $d$-unfounded set, where $d \in \{\Delta, \partial\}$.

Consider a defeasible theory $D = (F, R, >)$, the following are the definitions of $\Delta$-unfounded and $\partial$-unfounded [153].

**Definition 3.21 ($\Delta$-unfounded).** A set $S$ of literals is $\Delta$-unfounded w.r.t. an extension $E$ and definite inference iff: For every $s$ in $S$, and for every strict rule $B \rightarrow s$ either

- $B \cap -\Delta_E \neq \emptyset$, or
- $B \cap S \neq \emptyset$.

The above definition is very similar to the definition of unfounded set in [209]. The main differences are that the basic element of $S$ are literals (and “negation” is classical negation) and “negation as failure” is not present in the bodies of rules.

The corresponding definition of defeasible provability is more complex, more factors are involved during the inference process. Nevertheless, the basic idea is the same.
Definition 3.22 (\(\partial\)-unfounded). A set of literals \(S\) is \(\partial\)-unfounded w.r.t. an extension \(E\) and defeasible inference iff: For every literal \(s\) in \(S\), and every strict or defeasible rule \(r : A(r) \rightarrow s\) in \(D\), where \(\rightarrow\in\{\rightarrow, \Rightarrow\}\), either

- \(A(r) \cup -\partial_E \neq \emptyset\), or
- \(A(r) \cup S \neq \emptyset\), or
- \(\exists s \in R[q] \text{ s.t. } A(s) \subseteq +\partial_E \text{ and } \forall t \in R[q] \text{ either}\)
  - \(A(t) \cup -\partial_E \neq \emptyset\), or
  - \(t \not\approx s\)

Again notice the close relationship between the definition of \(\partial\)-unfounded set and the inference condition for \(\neg\partial\) in DL. The second disjunct has been added to the inference rule to capture cyclic dependency, stating that any rule \(r\) which contains a literal, or its negation, in the unfounded set is unfounded. Gelder et al. [208] called this a witness of unusability for the rule \(r\). However, the main difference between the definitions here to that of [209] are that the basic elements of the unfounded set \(U\) are literals (and negation is classical negation) and “negation as failure” is not present in the bodies of rules [153].

Example 3.4 (continuing from p. 62). Based on the definitions above, the set \(\{\text{teachAtUni}(John), \text{professor}(John), \text{highReputation}(John), \neg\text{highReputation}(John)\}\) is an unfounded set with respect to the defeasible theory. However, either \(\text{highReputation}(John)\) or \(\neg\text{highReputation}(John)\) can be derived if we can remove the loop caused by \(\text{professor}(John)(r_3)\) and \(\text{teacherAtUni}(John)(r_5)\). Thus only \(\text{professor}(John)\) and \(\text{teachAtUni}(John)\) constitute an unfounded set under DL.

Clearly the class of \(\Delta\)-unfounded and \(\partial\)-unfounded sets are both closed under the union. Hence for any defeasible theory \(D\) and a partial interpretation \(I\), there exists a greatest unfounded set \(U_D(I)\).

Definition 3.23. [153] The greatest unfounded set \(U_D^\nabla(I)\) of a theory \(D\) with respect to a partial interpretation \(I\) is the union of all the unfounded sets with respect to \(I\), i.e.:

\[
U_D^\nabla(I) = \bigcup_{i} U_i^\nabla(I) \mid U_i^\nabla(I) \text{ is an unfounded set of } D \text{ w.r.t. } I
\]

where \(\nabla \in \{\Delta, \partial\}\)

Here \(U_D\) can be viewed as producing the set of literals that are unfounded with respect to the interpretation \(I\); while \(T_D\) are the set of literals that are well-founded with respect to \(I\). It is straightforward to show that \(W_D\) is monotonic and so has a least fix point. We thus denote these least fix point by \(\Pi_D\) as the well-founded model of \(D^5\).

\[^5\text{In this thesis, our focus is on the computational aspect of the well-founded semantics for DL. For the theoretical aspect of this subject, please, refer to [16, 155] for details.}\]
Computing Consequences in DL with Well-Founded Semantic

To nullify evidence DL has to be able to disprove rules [37]. This means that the proof system should be able to demonstrate in a finite number of steps that there is no proof of the rule and thus remove them from the theory. As conclusions cannot be derived using circular arguments, *loops detection* plays a crucial role in deriving conclusions under well-founded semantics. *Failure-by-looping* provides a mechanism for falsifying a literal when it is within a loop with no external support. It helps to simplify a theory by removing inapplicable rules and makes theory becomes decisive, i.e., all rules in the theory are either provable or unprovable [36].

**Definition 3.24.** *Given a theory D and a set of literals* \( L \subseteq HB(D) \). *Then* \( L \) *is called a loop of D if (i) \( L \) is non-empty; and (ii) for every pair of literals* \( p_1, p_2 \in L \) *there exists a path from* \( p_1 \) *to* \( p_2 \) *in the literals dependency graph of D s.t. all vertices in this path belong to* \( L \).*

In other words, a non-empty set \( L \) of literals is a loop of \( D \) iff all literals in \( L \) are not externally supported and the subgraph of the literals dependency graph of \( D \) induced by \( L \) is strongly connected [8]. From the definition of the unfounded set any rule whose head belongs to an unfounded set, or there exists an unfounded literal in its body, is inapplicable. Since unfounded sets are finite, we have the following consequence.

**Proposition 3.25.** [8] *Given a theory D, a partial interpretation I, and unfounded set* \( U_D \) *w.r.t. I. If* \( U_D \neq \emptyset \), *we have* \( L \subseteq U_D \) *for some loop* \( L \) *in D that is unfounded w.r.t. I.*

The above proposition states that any non-empty unfounded set is a super set of some loop that is itself unfounded.

As a result, algorithm 3.3 (ComputeWFDL) shows the top level of the algorithm for computing the WFDL. Similar to the case of inferencing with DL, two global variables, \(+\partial\) and \(-\partial\), hold set of literals representing the conclusions inferred so far. They are initialized to empty set at the beginning of the pre-processing. The algorithm then repeatedly makes simple inferences until exhaustion (ComputeDefeasible) (Algorithm 3.1) and inference from unfounded sets (ComputeUnfounded) until no further conclusions can be inferred. Two other global variables, \( \partial^+ \) and \( \partial^- \), hold the most recent inferences of the process.

So for each iteration, let \( E = +\partial \cup -\partial \) be the conclusion established so far. The effects of all these conclusions have already been made by the simplification of the theory in ComputeDefeasible. As a result, it is sufficient to compute a \( \partial \)-unfounded set of the current theory w.r.t. \( \emptyset \), instead of computing a \( \partial \)-unfounded set of the original theory w.r.t. \( E \). Furthermore, the condition for an unfounded set after the simplification reduces to: a set of literals \( S \) is \( \partial \)-unfounded iff

\[
\forall s \in S, \forall r \in R[s], \ A(r) \cap S \neq \emptyset
\]

\(^6\text{HB}(D): \text{Herbrand base of} \ D\)
Algorithm 3.3: Inference Algorithm for Well-Founded Defeasible Logic (\( \partial \))

**Algorithm:** ComputeWFDL\((D)\)

**Data:** \( D = (0, R, >) \): a defeasible theory in regular form and with no defeaters

1. \(+\partial = +\Delta\)
2. \(-\partial = \{ \neg l : l \in +\Delta \}\)
3. repeat
   4. \(\partial^+ = \emptyset\)
   5. \(\partial^- = \emptyset\)
   6. ComputeDefeasible\((D)\)
   7. ComputeUnfounded\((D)\)
8. until \(\partial^+ = \emptyset\) and \(\partial^- = \emptyset\)

All rules such that \(A(r) \cap -\partial_E\) have been removed from the theory, and the literals for which the fourth condition of the original definition have already been included in the extension (lines 8–12 of ComputeDefeasible).

Algorithm 3.4: Computation of \(\partial\)-unfounded set

**Algorithm:** ComputeUnfounded\((D)\)

**Data:** \( D = (0, R, >) \): a defeasible theory in regular form and with no defeaters

Data: \( HB \): Herbrand base of \( D \)

Data: \( SCC \): the set of strongly connected components of \( D \)

Data: \( \partial^- \): the set of unfounded literals of \( D \)

1. ComputeSCL\((l, D)\), where \( l \in HB \)
2. foreach \( C \in SCC \) do
   3. if \( \forall l \in C, \forall r \in R[l], \exists m \in A(r) \cap C \) then
      4. \( \partial^- = \partial^- \cup C \)
      5. \( R = R - \{ r \in R : A(r) \cap C \neq \emptyset \} \)
      6. \( \{ > \} = \{ > \} - \{ (s, r), (r, s) \in > : A(r) \cap C \neq \emptyset \} \)
   7. \(-\partial = -\partial \cup \partial^-\)

Algorithm 3.4 uses a literal dependency graph, where nodes are literals and there is an edge from \( p \) to \( q \) if \( p \) appears in the body of a rule for \( q \). The computation of the greatest \( \partial\)-unfounded set (ComputeSCL, Algorithm 3.5) uses the insight of [8] (Proposition 3.25) that unfounded set are bounded above by the strongly connected components (SCC).

It is worth knowing that Algorithm 3.5 is based on Gabow’s algorithm [76] for computing SCCs, with adjustments made for the fact that the rules define a graph on literals. In the algorithm, \( D \) is the simplified input defeasible theory such that \( R_{sd} \neq \{ \emptyset \} \) but with no further simplification can be made; and \( L \) is the set of literals that appear in the Herbrand base of \( D \). \( S \) is a stack used to keep track on the set of processed literals; while \( P \), another stack, containing literals on the search path, is used to decide when to pop the set of SCC from \( S \). At the end of the process, every literals will be assigned with two values: (1) \( pre \) which indicates the permutation of literal defined by the access order numbering; and (2) \( groupId \) which indicates the group id of the SCC that the literal associated with. The variables \( cnt \) and \( gcnt \) are used...
Algorithm 3.5: Strongly Connected Literals computation

Algorithm: ComputeSCL(l, D)

Data: l: a literal in the Herbrand base of D
Data: D = (0, R, >): a defeasible theory in regular form and with no defeaters
Data: SCC: set of strongly connected literals in D

1. l.pre = cnt++
2. S.push(l)
3. P.push(l)
4. foreach r ∈ Rs : l ∈ A(r) do
   5. c = C(r)
   6. if c.pre = −1 then
      7. L = L \ c
      8. ComputeSCL(c, D)
   9. else if c.groupId = −1 then
      10. while P is not empty and P.top().pre > c.pre do
         11. P.pop()
   12. while P ≠ ∅ and P.top().pre > min(l.pre, ¬l.pre) do
      13. P.pop()
   14. if P.top().pre = l.pre then
      15. P.pop()
   16. else
      17. return
   18. repeat
      19. t = S.pop()
      20. t.groupId = gcnt
      21. SCC = SCC ∪ {t}
      22. until S is empty or t = l
      23. gcnt++

The algorithm works based on two observations: (1) when we reach the end of the recursive function, we know that we will not encounter any more literals in the same strongly connected set since all literals that can be processed have already been passed (line 4-11); and (2) the back links in the literal dependency graph provide a second path from one literal to another and bind together the SCC (line 12-21).

The recursive function first finds the highest literal reachable based on the literal dependency graph. It also utilizes a literal-indexed vector to keep track to the SCC and a stack to keep track of the current search path. A literal will be pushed onto the stack on entry to the recursive function; and pops them (with assigned SCC id) after visited the final member of each SCC. So at the end of the recursive procedure it will return to us all literals encountered since entry that belong to the same SCC, i.e., the set of literals that are unfounded. The algorithm extends the
unfounded set through each iteration. That is, to calculate the greatest unfounded set, we have to iterate \(\text{ComputeSCL}\) through the set of literals that appear in the defeasible theory \(D\).

However, not all SCC computed using \(\text{ComputeSCL}\) are \(\partial\)-unfounded sets. We have to prune out the components having a node with an incoming arc from a node not in the component itself (notice the correspondence of the revised condition for a set to be a \(\partial\)-unfounded set, and the condition to keep a SCC of \(D\), line 3 of \(\text{ComputeUnfounded}\)).

As discussed before, failure-by-looping provides a mechanism to remove literals in loops with no external support. That is, all literals in the unfounded sets are inferred to be failed (\(\neg\partial\)) and all rules containing such a literal in the body are removed, which subsequently enable us to derive the well-founded model of defeasible theory.

**Performance of the algorithms**

We now give bounds on the running time of \(\text{ComputeWFDL}\) for a defeasible theory \(D = (0, R, >)\).

Let \(N_l, N_R\) are the number of literals (both positive and negative) and rules that appear in \(D\) respectively. As computing the greatest unfounded set requires \(\text{ComputeSCL}\) to iterate through the set of literals that appear in \(D\), it runs in \(O(N_l)\) time. However, the prune out process in \(\text{ComputeUnfounded}\) requires a verification of literals such that literals with an incoming arc will be removed from the unfounded set. So, for each literals, we have to iterate through the set of rules that it appears as head and check for if there are any literals in the body that do not appear in the unfounded set, which thus subsequently makes \(\text{ComputeUnfounded}\) runs in \(O(N_lN_R)\) time.

Since \(\text{ComputeDefeasible}\) runs in \(O(N)\) time, where \(N\) is the number of removal occurrences in \(D\), therefore the computational complexity of the main part of \(\text{ComputeWFDL}\) is:

\[
O(M(N + N_lN_R))
\]

where \(M\) is the number of iteration that appears in the computation. However, in realistic knowledge base where loops appear only occasionally, the complexity of \(\text{ComputeWFDL}\) will of course approach the value of \(O(N + N_lN_R)\).

### 3.6 Summary

In summary, we have presented a theorem that allows us to reason on a defeasible theory without removing the superiority relations. The essence of this lies in the ability in identifying inferiorly defeated rules that are redundant and can be removed from the theory, which helps in preserving the representation properties of defeasible logic across different variants and can simplify the work in the subsequent processes.

Besides, we also presented algorithms for computing the extensions of defeasible theory under ambiguity propagation variant and well-founded semantics of DL. These contribute to
the computational aspect of the two variants in practical approach such that consequences of both variants can be computed with relatively low complexity when compared to other non-monotonic formalism, which make DL a possible candidate for some computational demanding jobs, or tasks that require immediate response, such as reasoning on the Semantic web [28].

In addition, Algorithm 3.3 can easily be modified to support the derivation of ambiguity propagation variant of DL with well-founded semantics (by changing $\text{ComputeDefeasible}(D)$ to $\text{ComputeDefeasibleAP}(D)$ in line 6 of the algorithm), which gives us lots of flexibilities in terms of implementations.

In the next chapter, we will discuss the implementation aspect of these algorithms and show that our approach outperforms the traditional approach by several orders of amplitude.
Chapter 4

SPINdle Defeasible Reasoner

Following the theoretical principles and algorithms depicted in the previous chapter, we have designed and implemented a defeasible reasoning engine, SPINdle, for computing consequences of defeasible theories. It is a bottom-up approach that can be used to compute conclusions for all arguments and is capable to perform efficient and scalable reasoning with defeasible theories of over 1 million rules [142]. It is a tool that (i) support our ongoing research in defeasible reasoning, and (ii) is provided for public use to apply defeasible reasoning to practical problems or for education purposes.

SPINdle, in accordance with the general philosophy of logic programming, is designed to compute the total conclusions of a defeasible theory. It is written in Java and is by no means the only implementation to inference with defeasible theory. Nute [166], Maher [151], Bassiliades et al. [23], Cecchi et al. [54], Antoniou and Bikakis [12], and others have developed similar programs (cf. Section 2.6). However, as it is argued by Bryant and colleagues [49, 50], for a real world software applications to be assisted by defeasible reasoning, access to a general purpose non-monotonic reasoning component is needed. Even though there are several prototypes of standalone non-monotonic and argumentation-based applications documented in academic literature, many of the implementations are either proprietary or have been built using esoteric or inefficient languages not suitable for large-scale deployment [48]. Compared to these efforts, SPINdle is a modest prototype of efficient and flexible defeasible reasoning system that offers users with limited constraints.

SPINdle can be used as a standalone theory prover or can be embedded into any applications as defeasible reasoning engine. It allows users or agents to issue queries on a given knowledge base, and allows theories to be generated on the fly by other applications and perform reasoning on it. Its built in data structures allow literals and rules in memory to be loaded or manipulated in a constant time.

This chapter focuses on the implementation aspect of the latest version of SPINdle\textsuperscript{1}. It should be noted that SPINdle is under active development and the source code may differs in minor ways from the description here. The followings are some features of SPINdle.

\textsuperscript{1}The latest version of SPINdle at the time of writing this thesis is version 2.1.0.
• It supports all rule types of DL, namely: fact, strict rules, defeasible rules, defeaters and superiority relations.

• It supports negation and conflicting (mutually exclusive) literals.

• It supports conclusion reparation with preference operator (Definition 5.1 and 2.27) when inferring conclusions using defeasible rules (cf. Section 4.1.2.1)

• It provides a flexible mechanisms that allows users to capture different intuitions of a defeasible theory. To be precise, SPINdle supports defeasible reasoning with different variants, which includes ambiguity blocking, ambiguity propagation, well-founded semantics, and their combinations.

• For each literals, two kinds of conclusions, definite provability and defeasible provability, are inferred and can be retrieved from the conclusions sets in accordance with the strength of proof that users interested in.

• It supports Modal Defeasible Logic (MDL) with modal operator conversions and conflict resolutions (cf. Section 2.4 and [94]).

• It is feasible for implementation in embedded platforms.

• It is extensible through additional elements to take advantages of the underlying reasoning facilities.

Besides, SPINdle also features a visual theory editor\(^2\) (Figure 4.1) for authoring/editing defeasible theories; and a console application, which provides the same functions as the theory editor, was implemented to support efficient theories testings and demonstrations.

\(^2\)http://spin.nicta.org.au/spindle/tools.html
The design process has been conducted using an iterative model, highly linked with the implementation activity, during which various algorithms and features have been proposed, evolved, and tested, thus validating both the theoretical model and the architecture.

The following sections illustrate different aspects of the SPINdle architecture.

4.1 SPINdle Architecture

Building and deploying monolithic applications is a thing in the past [214]. As the complexity of applications increases, decomposing a large system into several smaller and well-defined collaborating pieces, and hiding design and implementation details behind a stable application programming interface (API) is a much better way to go. It helps reducing the complexity of application development and improves the testability and maintainability of the whole application through enforcing the logical module boundaries.

SPINdle is an expandable system designed using plug-in architecture such that modules (or components) are loaded dynamically into the application when they are necessary, which subsequently helps reducing the footprint and memory consumption of the whole system; and can be extended easily according to the future needs.

Figure 4.2 depicts the overall system architecture of SPINdle. The internal architecture of SPINdle is designed as a composition of different functional components, which can be grouped into two sets:

- **Base components**: they provide the very basic operations for theory modelings, data handling, I/O operations and system configurations.

- **Reasoning components**: they provide the theory transformations and reasoning capabilities, and becoming active only when a transformation and/or reasoning task is received from users (or applications).

That is, base components contribute to the creation of a reasoning context space; while reasoning components embody the reasoning that underlies in each defeasible theories.
Regarding the reasoning components, SPINdle is intrinsically equipped with various theory normalizers and reasoning engines but, since there are different types of theory and reasoning capabilities requirements, only the theory normalizer and/or reasoning engine with suitable theory types and appropriate reasoning capabilities will be loaded onto the system space when required, and will be removed from the system space when a theory with different theory type is imported or different reasoning capabilities is needed (cf. Section 4.1.2).

This flexible scheme makes SPINdle able to cope with different platforms and capture different intuitions to a broad range of scenarios, trying always to obtain the most from the existing system environment. In some sense, SPINdle is designed as a black-box, trying to hide every implementations details and allows users to carry out defeasible reasoning in their applications seamlessly.

SPINdle works in the following way: The user, or application, imports defeasible theories, either using the syntax of defeasible logic (called DFL) or in the RuleML-like syntax, which will be described in the sections below. The theories are then checked by the associated theory parsers (which depends on the syntax used), and if they are syntactically correct, they are then passed to the theory normalizer for transforming the theories into an equivalent theory (in regular form) without defeaters (and also without superiority relations if Maher’s approach [151] is used) (cf. Section 3.2.2). Subsequently, the inference engine evaluates the transformed theories and derive the conclusions that corresponds to the intuitions that the user selected (ambiguity blocking, ambiguity propagation, well-founded semantics, or their combination). The theory outputters (in Figure 4.2) is used to output the current theory in the system or the conclusions derived to the user, according to the syntax required.

4.1.1 Base Components

The goal of base components is to contribute to the creation of a common reasoning context among different components. They can also provide the means for the reasoning components to be controlled or configured by users (or applications) in a more convenience ways. In order to do so, they expose several services through their interfaces:

- **Theory retrieval**: it provides mechanisms of retrieving and parsing defeasible theories that are stored locally in the file system or anywhere else that can be accessed through the Internet.

- **Theory modeling**: it provides a theory model that facilitates the inference process as proposed in Chapter 3.

- **Context management**: it provides management/configuration operations over the reasoning components and other system-wise configurations.

The context management interface is aimed at influencing the behavior of SPINdle such that users/applications can select which intuition(s) to be captured by the reasoning components,
which algorithms\textsuperscript{3} to be used in the inference process, the location of theories/conclusions to be stored and in what format, progress information display, etc. However, since all this configurations information can be captured easily in Java (through implementing a proxy using static class morphing), the following sections will focus on the implementations aspects of the other two services: theory retrieval and theory modeling.

4.1.1.1 I/O Manager

The I/O Manager manages the communications between SPINdle and users and/or applications. It provides users an interface to interact with SPINdle in loading and/or storing defeasible theories and conclusions after computation. It is composed mainly by two groups of functional elements: theory parsers and theory outputters, that carry out the required theories import and export processes, and can work independently from the SPINdle reasoning engine.

To be able to accept defeasible theories that are stored in local file system or anywhere else that can be accessed through the Internet, SPINdle uses URI to represent the location of the defeasible theories. The biggest advantage of this method is that URI are independent of the system locale, meaning that no matter which locale SPINdle are running, the same defeasible theory files can be retrieved using the same URIs.

After receiving an URI, the I/O Manager then fetches the document from the source location and performs an initial analysis on the theory to decide which theory parsers should be employed to parse the document.

Note that defeasible theories are entered into components of SPINdle either in textual form or in the RuleML-like syntax. The defeasible theories language used by SPINdle, known as DFL, is summarized in Appendix C. It is very similar to what we use in describing logic theories in the literature, and is very simple and easy to use. Example 4.1 shows an example of using DFL language to represent a defeasible theory in SPINdle.

**Example 4.1.** The following defeasible theory:

\[
\begin{align*}
r_1 & : \ a, b \rightarrow c \\
r_2 & : \ a, b \Rightarrow \neg c \\
r_1 & > r_2
\end{align*}
\]

and the following facts:

\[
\begin{align*}
a & \quad b
\end{align*}
\]

can be represented in DFL\textsuperscript{4}, as follows.

\textsuperscript{3}Currently two algorithms: Maher’s algorithm [151] and the algorithms proposed in Chapter 3, are available.

\textsuperscript{4}As a quick reference, “\textgreater\textgreater” denotes fact, “\textasciitilde\textrightarrow, \textggtr” are used to denote strict and defeasible rules respectively. Please refers to Appendix C for details.
As can be seen here the DFL language is simple, and is appropriate for human beings to read and analyze a defeasible theory, it is not typically the case for communications between machines over the Internet.

To compensate this shortcoming, SPINdle also supports defeasible theories represented using XML, which is a common scenario in the Semantic Web or agents communication. The XML schema (Appendix D) that SPINdle utilizes is very similar to that of RuleML [189]. In most cases, due to some historical reason that appeared when we were implementing the XML theory parser, the two are only different by the tag names used. For instance, rules in RuleML are encoded in an implies element, while in SPINdle, rule is used. However, they all have a rule label (which act as a rule identifier) and a rule type (indicate the strength of the rule) attribute associated with it. On the top of this, following the approaches proposed in [173, 12], the elements that we add/modify to support defeasible theories are:

- The element FACT, which consists of a literal element, refers to the fact assertions.

- The attributes label and strength to element rule, which refer to the rule label and rule types (“STRICT”, “DEFEASIBLE” or “DEFEATER”) of the rule respectively.

- The child element head to element rule, which consists of either a single literal or a list of literals, used to replace the element then in RuleML and refers to the (list of) consequences of the rule.

- The sup empty element, which accepts the rule label of two rules as its attributes (superior and inferior) and refers to the superiority relation between these two rules.

Example 4.1 (continuing from p. 73). Based on the SPINdle XML schema, the same defeasible theory is represented using XML, as shown in Figure 4.3.

So after fetching and parsing the theory document, the theory parsers then will translate the content into a theory model (cf. Section 4.1.1.2) that can be processed and manipulated by the reasoning components. Its nature is rather similar to the Logic Program Loader modular of the DR-Device family of applications developed by [23]. However, the logic program loader compiles the defeasible logic rules into a set of CLIPS production rules that can be used by the underlying reasoning engine; while the theory parsers in SPINdle will generate the theory model that can be used by other parts of the engine directly.

Besides, as said, the theory parsers can also be used separately as a test program that exercise lexer and parsers required to parse a defeasible theory, and can be run directly using the Java console with the following command:
Figure 4.3: Sample theory of Example 4.1 in XML format
where `url` is the location of the defeasible theory file location and `logger` is the application
logger (which can be `null`). A defeasible theory model will be returned if the file can be parsed
without any problems, or a parser exception will throw.

The `theory outputters`, on the other hand, is used to stored, or export, the modified theories
or conclusions derived. In a similar vein, the theory outputters can export the content according
to the formalism specified by users. That is, at this moment, the content can be exported as
a DFL file, or an XML document. Contrary to what the theory parsers can do, instead of
exporting the content to a particular location specified by a URI, they can only be exported to
the local file system.

4.1.1.2 Theory modeling

The key to an efficient implementation of the algorithms proposed in Chapter 3 is the data
structure used to model a defeasible theory, especially the data structure used to represent rules
such that rules in association with a particular literal (and its negation) and/or superiority
relations can be retrieved from the theory with a limited amount of time. Here we have extended
the data structure proposed by Maher [151] to handle modal literals and inferiorly defeated rules,
and is exemplified (albeit incompletely) in Figure 4.4 for the theory below:

\[
\begin{align*}
r_1 & : \quad b, [\text{INT}]c \Rightarrow a \\
r_2 & : \quad b, d \Rightarrow a \\
r_3 & : \quad \neg b, [\text{INT}]c, [\text{OBL}]\neg d \Rightarrow \neg a \\
r_4 & : \quad b, [\text{INT}]c \Rightarrow [\text{OBL}]\neg d \\
r_2 & > r_3
\end{align*}
\]

Each rule body is represented as a set in the memory. For each literal `p` there are linked
lists (the solid line) of the occurrences of `p` in the rules can be created during the theory parsing
phase. Each literal occurrence has a link to the rules it occurs in (the dashed arrow). Using
this structure, whenever a literal update occurs, the list of rules relevant to that literal can
be retrieved easily. Thus instead of scanning through all the rules for empty head, only rules
relevant to that particular literal will be scanned. Besides, this structure also help in retrieving
information of the conflicting literal(s) that may appear in the theory, which thus further helps
in speeding up the inference process.

Moreover, this structure also allows the deletion of literals and rules in time proportional
to the number of literals deleted as it facilitates the process by detecting in constant time on
whether a literal deleted was the only literal in that rule.

To facilitate the operations induced by Theorem 3.16, an efficient approach is needed in
detecting whether a rule is in association with any superiority relations. Taking this into con-
sideration, two numbers \((r_{sup}, r_{inf})\), sitting next to the rule label, representing the number of
superior rules and inferior rules a rules currently associated with respectively, were used. For
example, as can be seen from the theory, $r_2 > r_3$. So in the figure, $r_2$ has one inferior rule while $r_3$ has one superior rule.

So, during the inference process, these two numbers will be updated accordingly while rules are removed from the theory. Testing whether a rule is an inferiorly defeated rule with no weaker rule appear in the theory becomes a simple task since after the rule simplification process, we only need to test on the conditions that $r_{sup} > 0$ while $r_{inf} = 0$. If this condition is asserted and a superior rule with empty body exists, then according to Theorem 3.16, that particular rule becomes redundant and can be removed from the theory, and a negative conclusion may derive accordingly. Thus, tracing the set of superiority relations in theory for detecting inferiorly defeated rules becomes no longer necessary.

Besides, the theory interface also provides useful methods that allows users to modify/manipulate the theory according to their application needs, which helps in simplifying their development process.

### 4.1.2 Reasoning Components

Reasoning components provide SPINdle with theory transformations and reasoning capabilities. Through these components, SPINdle is able to perform various transformations to the defeasible theories, and finally reasoning on them using the reasoning engines.

Reasoning components can act independently from the base components and their activities are totally triggered by the transformations/reasoning processes required by users.

There are two groups of reasoning-related components in SPINdle:

- **Theory Normalizers**: implement various transformation algorithms that transform the
defeasible theories to a form that can be processed by the associated inference engines.

- **Inference Engines**: implement the defeasible reasoning mechanisms. It captures the intuitions of the defeasible theories based on the context and user configured information directly obtained from the context management services, and returns the results to users via reasoning.

The steps carried out by these components are:

- Receive a defeasible theory from an user and/or application and decide which types of theory normalizers and inference engines should be used based on the elements contained in the defeasible theory. An exception is thrown if no suitable theory normalizer or inference engine is available.

- Apply transformations to the theory as of required by the associated inference approaches, or as are directed by the users.

- Identify the constraints in the theory that cannot be honoured by the inference engines by checking the elements in the theory (e.g., whether the theory contains any facts, defeaters) and send the candidate constraints to the user, if any.

- Inference on the theory and send the conclusions inferred to the user.

Currently, SPINdle is equipped with two theory normalizers (one for standard defeasible theory (SDL) and one for modal defeasible theory (MDL)), and eight inference engines (four for inferencing with SDL\(^5\) and another four for MDL). The inference engines capabilities are selected and controlled by the users as they specify the reasoning approaches as well as the intuitions that they want to capture from the context management services. However, the use of SDL or MDL components will be selected automatically by the SPINdle system upon receiving the defeasible theories. Simply speaking, if there exists a modalised element\(^6\) in the theories then an MDL component(s) will be selected. Otherwise, a SDL component(s) will be used instead.

### 4.1.2.1 Theory Normalizers

Depending on the inference algorithm selected, for example if Maher [151] algorithm is used, we may also have to remove the superiority relations before the inference process actually takes place. For this purpose, based on the transformations described in Definition 3.4–3.6, several theory normalizers have been designed, and implemented, to perform these theory transformation tasks.

\(^5\)The four SDL inference engines include: one for inferencing with ambiguity blocking with (optional) well-founded semantics and one for inferencing with ambiguity propagation with (optional) well-founded semantics using the algorithms proposed in Chapter 3; and the other two are of the same capabilities but based on Maher’s approach [151]. The four MDL inference engines are following the same pattern as well.

\(^6\)Modalised element can be either a rule attributed by a modal operator or a modal literal appeared in the language of the theory.
In addition to the features that are available in other defeasible reasoners, as mentioned in the introductory section of this chapter, SPINdle also exposes an additional feature of preference handling. It supports conclusion reparation through the use of the preference operator $\otimes$ (Definition 5.1 and 2.27), and is denoted as rules with multiple heads in the SPINdle defeasible theory language, using comma (\text{"\text{,}\text{\text{)}}}\) to separate different reparative conclusions.

Notice that in this perspective the meaning of the expression:

\[ A_1, \ldots, A_n \vdash B_1, \ldots, B_n \]

is: the sequence $A_1, \ldots, A_n$ comports that $B_1$ is the case; but if $\neg B_1$ is the case, then $B_2$ is preferred, and so on. It is not a short-hand notation for grouping rules with similar bodies together. Normatively speaking, this means that the consequence determined by $A_1, \ldots, A_n$ is $B_1$; however the violation $B_1$ can be repaired by $B_2$, and so on.

Let us consider the following standard rules for negation [91]:

\[
\begin{align*}
A, B &\vdash C \\
A &\vdash \neg B, C
\end{align*}
\]

\[
\begin{align*}
A &\vdash B, C \\
A, \neg B &\vdash C
\end{align*}
\]

The effect of these rules is to move a formula on the other side of $\vdash$, and change its polarity. Accordingly, given the rule

\[ A \Rightarrow B \quad (4.1) \]

and its exception

\[ A, \neg B \Rightarrow C \quad (4.2) \]

we can obtain\textsuperscript{7}

\[ A \Rightarrow B, C \quad (4.3) \]

by applying the rule for the negation on (4.2).

So, the rule in (4.1) says that $B$ should be the case when $A$ is obtained. According to (4.2) $C$ should be the case given $A$ and $\neg B$; consequently we have (4.3).

To recap, the above semantics can be characterized by the following definition.

**Definition 2.27** (Reparation rule). Let $r_1 : \Gamma \Rightarrow \bigotimes_{i=1}^m a_i \otimes b$ and $r_2 : \Gamma' \Rightarrow c$ be two goal rules. Then $r_1$ subsumes $r_2$ iff $\Gamma \cup \{ \neg a_1, \ldots, \neg a_n \} = \Gamma'$, $c = e$ and $b = (e \otimes (\bigotimes_{k=1}^n d_k))$, $0 \leq n$.

Base on this definition, we define below a transformation sim\_heads that transforms every defeasible theory into an equivalent defeasible theory such that every defeasible rules in $R$ with multiple heads will be transformed into a set of single headed defeasible rules.

\textsuperscript{7}Note that as discussed in Section 2.7.2.2, facts are indisputable statements and strict rules are inferred in classical sense, leaving defeasible rules as the only rule type that can have heads with more than one literals.
Definition 4.1. Let \( D = (F, R, >) \) be a defeasible theory, \( r : A(r) \Rightarrow C(r) \) be a rule in \( R \) and let \( ||r|| \) be the number literals in \( C(r) \). Define sim_heads\((D) = (F, R', >') \) where:

\[
R' = \{ R \setminus R_d \} \cup \bigcup_{r \in R_d} \text{sim_heads}(A(r), C(r))
\]

and

\[
\text{sim_heads}(A(r), C(r)) = \begin{cases} 
\{ q : A(r) \Rightarrow q \} \cup \text{sim_heads}(A(r) \cup \neg q, C(r) \setminus \{ q \}) & \text{if } ||r|| > 1, \\
r & \text{otherwise}
\end{cases}
\]

where \( q \) is the first literal that appears in \( C(r) \).

The superiority relation \( >' \) is defined by the following condition

\[
\forall r', s' \in R'(r') >' s' \iff \exists r, s \in R \ r' \in \text{sim_heads}(r), s' \in \text{sim_heads}(s), r > s
\]

where \( r \) and \( s \) are conflicting.

Example 4.2. Given a rule \( r : a \Rightarrow b, c, d \). We apply to \( r \) the transformation sim_heads, obtaining the rules:

\[
\begin{align*}
\ r_b & : \ a \Rightarrow b \\
\ r_c & : \ a, \neg b \Rightarrow c \\
\ r_d & : \ a, \neg b, \neg c \Rightarrow d
\end{align*}
\]

In cases where there is another rule \( s : \Rightarrow \neg c \) in \( R \) and a superiority relation \( s > r \). Then the superiority relation \( >' \) will be updated accordingly, as follow: \( s > r_c \).

So, for a defeasible rule \( r \) with head size \( ||r|| = n \), it will first be expanded to an equivalent set of rules of size \( n \) using sim_heads before any other transformations take place, and the rule labels in superiority relations will be updated accordingly using the newly generated (unique) rule labels.

Accordingly, Figure 4.5 depicts the modified defeasible theory inference process. Note in the figure that the transformation sim_heads has been added at the beginning of the inference process so as to replace the defeasible rules with multiple heads with an equivalent set of single head rules at the beginning of the inference process. With the improved inference algorithms described in Chapter 3, the superiority relation removal transformation becomes superfluous. However, we keep this transformation in the theory normalizers to cater for the cases where users may want to infer the conclusions using other approaches.

In addition, since sim_heads applies only to the small set of defeasible rules with multiple heads and will be used only once at the beginning of the inference process, it creates no effect on the computational complexity of the algorithms that we have discussed in the previous chapter.
4.1.2.2 Inference Engines

Inference engines provide SPINdle with the defeasible reasoning capabilities through implementing different defeasible reasoning algorithms.

SPINdle is implemented on the algorithms that we presented in the Chapter 3\(^8\). As discussed before, the inference process is based on a series of transformations, which contains two steps: (i) a theory simplification step which progressively simplifies a defeasible theory, and (ii) a conclusions generation step which derive new conclusions from the simplified theory and accumulate the set of conclusions inferred. It corresponds to a restricted form of transition system where some sequences of transitions are not allowed. The engines will, in turn:

1. Remove any inferiorly defeated rules that are redundant in the theory as per determined using Theorem 3.16.

2. Assert each fact (as an atom) as a conclusion and remove the atom from the rules where the atom occurs positively in the body, and “deactivates” (remove) the rule(s) where the atom occurs negatively in the body. The atom is then removed from the list of atoms.

3. Scan the list of rules for rules with empty body, takes the head element as an atom and search for rule(s) with conflicting head. If there are no such rules then the atom is appended to the list of facts and the rule will be removed from the theory. However, if

---

\(^8\)As mentioned before, in SPINdle, in addition to the four inference engines implemented based on the algorithms presented in the previous chapter, another four inference engines based on Maher’s approach is also included in the package, which provides users a flexibility to choose the approach they want to inference on.
the current rule is an inferiorly defeated rule that satisfies the condition stated in Theorem 3.16, it will, then, be removed from the theory and proved negatively if there exists no more rules with the same head literal in the theory. Otherwise the atom will be appended to the pending conclusion list until all rule(s) with conflicting head can be proved negatively.

4. Repeat the first step.

5. The algorithm terminates when one of the two steps fails or when the list of facts is empty. On termination the inference engine output the set of conclusions by either export them to the user or to save them in theory database for future use through the I/O Manager.

However, it is important to note that the descriptions above are a generalized version that is common in inferencing both standard defeasible logic (SDL) and modal defeasible logic (MDL). In the case of modal defeasible logic, due to the modal operator conversions, an additional process adding extra rules to the theory is needed. In addition, for a literal \( p \) with modal operator \( \Box_i \), besides its complement (i.e., \( \Box_i \neg p \)), the same literal with modal operator(s) in conflict with \( \Box_i \) should also be included in the conflict literal list (step 2) and only literal with strongest modality should be concluded.

The inference mechanisms we have in SPINdle are very similar to that of SLG resolution [56] except that, in the theory simplification process, we also have to consider the effect of the inferiorly defeated rules that may arise due to the superiority relations. That is, after removing literals from a defeasible rule, we have to verify whether or not the rule is an inferiorly defeasible rule that appears at the end of a superiority chain. If this is the case, then its conclusion should be expelled from the conclusion set since it will be overruled by some superior rules. If, on the other hand, some superior rules with non-empty body appears, then the conclusions generations step should be deferred until:

- All superior rules become inapplicable. In this case, a positive conclusion of the defeasible rule can be derived if no rule with conflicting conclusions is applicable.

- The body of some superior rules become empty. In this case, the defeasible rule becomes inferiorly defeated. However, according to Theorem 3.16, we still need to keep this rule in the theory until it becomes the weakest rule in the superiority chain.

Several implementation techniques have been exploited to enhance the performance of SPINdle inference engines. Among these the two most importants are memoing (or tabling) and delay literals for negative literals.

Memoing is one of the most effective and commonly used techniques in reducing redundant subcomputations. Recall the notion of memoing in [161]. The concept of memoing is simple. In deterministic language, during execution, we can maintain a table of procedure calls and the values return by these procedures [218]. If the same procedure call is made later in the
computation, then the stored values in the table will be used to update the current state, instead of calling the procedure again and re-compute the result. It is well-known that such strategy can save us a lot of time and can turns an exponential algorithm to polynomial time in some cases, like naive Fibonacci [2].

In defeasible reasoning, the concept of memoing is the same but the situation is a bit different. During the inference process, we memoize all conclusions that have been proved. If the same conclusions are derived later in the computation, then these conclusions will be ignored and we will not re-execute the inference procedure on the literals to simplify the theory or infer any new conclusions.

Delay literals refer to the cases where the inference on some literals must delay at least some calls before inferring the conclusions. It corresponds to a (perhaps temporary) assumption that a literal is undefined when conflicting literals that are involved are still pending to reason. Suspending the process of these literals allow literals in the body of other rules can be evaluated. A delayed literal may later be found to be positively or negatively provable; in which case it is simplified away, the simplification process may lead to further answers [192, 57].

To add memoing to SPINdle, two global tables, one used to store the accumulated set of conclusions that have been inferred while the other contains the set of conclusions pending to infer, are used. For each iterations of the loop in the algorithms, a pending conclusion will be extracted from the pending conclusions set. It will, first be added to the accumulated set, and then perform the simplification process. If new conclusions are derived, then only those that do not appear in the accumulated set or pending set will be added to the pending set for further process.

To facilitate the simplification of delayed literals, another global table will be used to temporary store the undefined conclusions set. Each conclusions in the table will be updated when (i) all rules with conflicting head are removed from the theory. In this case, a positive conclusions of the literal can be inferred; or (ii) a conflicting literal can be proved positively. Then in this case, due to the sceptical nature of DL both literals will be inferred negatively.

4.1.3 Extending SPINdle

Before starting the actual inference, users have to select the types of intuition that they want to capture. Along this work, we have implemented support that provide defaults for this selection for various types of transformation and inference algorithms. These selections are grouped into alternative configurations which correspond to the reasoning engines and mechanisms to be used by SPINdle in inferencing process.

The representation formalism presented in Appendix C and the XML schema in Appendix D is rather simple, but is sufficient for representing defeasible theory in SPINdle so far. However,
in practice, they may not be expressive enough to model real-world applications. Taking this into account, users may want to design their own languages to represent their defeasible theories and utilize the inference mechanisms provided by SPINdle to inference on the theories. Besides, in some cases, users may, as well, would like to implement their own inference algorithms.

As mentioned in the introduction of this chapter, SPINdle is designed using a plug-in architecture that can be extended easily. That is, in addition to the representation formalisms or inference engines supported by SPINdle, users are free to extend the features in the package as if they are necessary to fulfill their requirements.

To extend SPINdle with a new representation formalism, one mainly needs to provide a procedure for parsing the lexical form of the theory, which includes (at least) procedures of parsing the lexical form of facts, all type of rules, literals and superiority relations, and associate this procedure with the theory data type (represented using a text string)\(^{10}\). SPINdle makes this task easy by allowing users to create this procedure through:

- create an entirely new theory parser class that implements the theory parser interface; or
- extend an existing theory parser in the SPINdle I/O package.

That is to say, users can implement their new theory parsers by either extending the features currently supported by SPINdle or by starting afresh a new representation formalism and implement it from scratch. In either way, users are free to integrate any third parties libraries available to the SPINdle package, if necessary\(^{11}\).

In a similar vein, the same also applies to theory outputters such that users are free to implement their own theory outputters according to the formalism that they want, or according to their applications needs.

After implementing the theory parser/outputter, users can modify the configuration file located at `spindle/resources/io_conf.xml` in the package, by adding a line indicating the class name of the new parser/outputter class. The following code shows the content of the configuration file that comes with the package.

```xml
<spindle>
  <io classname="spindle.io.parser.XmlTheoryParser" />
  <io classname="spindle.io.parser.DflTheoryParser" />
  <io classname="spindle.io.outputter.XmlTheoryOutputter" />
  <io classname="spindle.io.outputter.DflTheoryOutputter" />
</spindle>
```

Or, at users’ preference, the I/O Manager can be configured to detect all theory parser/outputter classes that are available on the classpath automatically, without the need of modifying

\(^{10}\) Under normal circumstance, this theory data type will be used as the extension name of the file storing the defeasible theory.

\(^{11}\) As an example, users may consider using Jena (http://jena.sourceforge.net) to parse a RDF file when creating a RDF theory parser.
the underlying configuration of the package. However, doing this is not recommended for large applications as considerable time will be used in searching classes in the classpath.

The concepts for extending the theory normalizers and inference engines in SPINdle are just the same as what we have described except that, instead of modifying the content of the configuration file, we have to drill down to the code level and modify the information in the reasoning engine factory\textsuperscript{12} such that a desired theory normalizer (or respectively inference engine) can be loaded into the memory accordingly.

Since the goal of this thesis is to give readers an overview of the SPINdle architecture, discussions on the implementation and configuration details of how SPINdle could be extended are beyond the scope of this thesis. Readers interested in this subject please refer to the SPINdle User Guide [141], Chapter 4 on “Embedding SPINdle in Java Applications” for details.

4.2 Performance Evaluation

SPINdle, from its start, is designed to compute the conclusions of all arguments of a defeasible theory. It has been extensively tested for correctness, scalability and performance using different forms of theories. In this section, we report on the experimental evaluations that we conducted to compare the performance of the traditional approach and our new approach. Besides, we also compare the performance of SPINdle with other defeasible reasoning system, namely, Delores [154].

4.2.1 Design of the Experiments

To get a rough idea of how SPINdle performs, we employed the DTScale tool of Deimos (cf. Section 2.6.2) to generate different sets of defeasible theories that were designed to test different aspects of the implementations. For instance, the first group of test theories are used to test only undisputed inferences. For example, in chain\((n)\), \(a_0\) is appeared at the end of a chain of \(n\) rules, \(r_i : a_{i+1} \Rightarrow a_i\). In circle\((n)\), \(a_0\) is appeared as part of a circle of \(n\) rules, \(a_{i+1 mod n} \Rightarrow a_i\). In tree\((n, k)\), \(a_0\) is the root of a \(k\)-branching tree with depth \(n\), in which every literals occur only once.

The second group of the test theories, on the other hand, are used to perform test on disputed inferences, under different situations. In test theory teams\((n)\), every literals are disputed, with two rules for \(a_i\) and two others for \(\neg a_i\) such that the rules for \(a_i\) is superior to the rules for \(\neg a_i\). And this situation is repeated recursively for a depth \(n\). Beside, in order to realize the performance benefit of our new approach, we have created a new test theories, supTest\((n)\) by adding \(n\) disputed rules \(r_i^\neg : \Rightarrow \neg a_i\) and \(n\) superiority relations \(r_i > r_i^\neg\) to chain\((n)\).

Appendix A and Table 4.1 show a detailed descriptions and some metrics of the test theories, respectively. Notice that in our performance evaluation, the theory sizes refer to the value of \(n\)

\textsuperscript{12}In the source package: spindle.engine.ReasoningEngineFactory
<table>
<thead>
<tr>
<th>Theory</th>
<th>Facts</th>
<th>No. of Rules</th>
<th>No. of Superiority Relations</th>
<th>Size$^a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>chain($n$)</td>
<td>1</td>
<td>$n$</td>
<td>0</td>
<td>$2n + 1$</td>
</tr>
<tr>
<td>circle($n$)</td>
<td>0</td>
<td>$n$</td>
<td>0</td>
<td>$2n$</td>
</tr>
<tr>
<td>dag($n, k$)</td>
<td>$k$</td>
<td>$nk + 1$</td>
<td>0</td>
<td>$nk^2 + (n + 2)k + 1$</td>
</tr>
<tr>
<td>tree($n, k$)</td>
<td>$k^n$</td>
<td>$\sum_{i=0}^{n-1} k^i$</td>
<td>0</td>
<td>$(k + 1)\sum_{i=0}^{n-1} k^i + k^n$</td>
</tr>
<tr>
<td>teams($n$)</td>
<td>0</td>
<td>$4\sum_{i=0}^{n} 4^i$</td>
<td>$2\sum_{i=0}^{n} 4^i$</td>
<td>$10\sum_{i=0}^{n-1} 4^i + 6(4^n)$</td>
</tr>
<tr>
<td>supTest($n$)</td>
<td>1</td>
<td>$2n$</td>
<td>$n$</td>
<td>$5n + 1$</td>
</tr>
</tbody>
</table>

$^a$Size refers to the overall size of the theory, defined as the sum of the numbers of facts, rules, superiority relations, and literals in the bodies of all rules.

Table 4.1: Sizes of test theories (depicted from [154])

that we used in generating the test theories, and with $k = 3$ if applicable. The *reasoning time* in the results include the time used in loading theory onto SPINdle, and the time used in (all) theory transformations and inference process. For each theories, except the *circle* theories, a proof of $+\partial a_0$ would require the use of all facts, rules and superiority relations; while in the *circle* theories, since the rules themselves will form a loop, no positive defeasible conclusions can be inferred.

Here, we choose to compare the performance of SPINdle against the Delores because it is the only defeasible reasoner available that can compute the total conclusions of a defeasible theory; others were built based on query-based algorithms that can only inference on one particular literal at a time. The basic characteristics of Delores is presented in Section 2.6.

### 4.2.2 Configuration for the Experiments

In the experiments, SPINdle is compiled using the SUN Java SDK 1.7 without any optimization flags. Based on the sizes of the test theories, the experiments begins with a heap size of 512MB and gradually increased to 2GB. The times presented in the experiments are those measured by the system functions supported by the Java Virtual Machine (JVM).

For the evaluation of Delores, we compiled it with gcc without optimization flags. Times are measured using the standard *time* library. However, it is important to note that the pre-processing transformations in Delores are not fully completed and/or still required tuning, test theories imported are assumed to be in regular form and with no superiority relations and defeaters. To resolve this issue, the pre-processing transformations will be performed using an application written by us, before importing the theories into Delores.

In the experiments, we focus on defeasible inference, assuming the ambiguity blocking behavior of the test theories All experiments were performed on the same lightly loaded Intel Core2 (2.83GHz) PC operating under Linux (CentOS) and with 4GB main memory. Each timing datum is the mean of several executions. There was no substantial variation among the executions, except as noted.
4.2.3 Experimental Results

Figures 4.6 to 4.11 present the time (in CPU time) and memory usage required by SPINdle to find the appropriate conclusion for $a_0$ using the traditional approach and our new algorithms. The experiments are designed to execute all rules and literals of each test theories, and each test theories have been executed for at least ten times. The time showed in the figures are the average values of these tests, which includes the time used to import the test theories into SPINdle, the time used on theory transformations and the time used for inferencing.

It can be seen from the figures that SPINdle can handle inferences with thousand of rules in less than three seconds. In almost all cases, the inference time per clause is less than few milliseconds. Even though some fluctuation is seen when the value of $n$ is large, in general, the performance scalability of SPINdle, in terms of reasoning time and memory consumption, is (in most case) linearly proportional to the size of the theories tested.

To recap, the purpose of our new algorithm is to reduce the number of literals or propositions to be introduced to the defeasible theories due to transformations, and to remove redundant rules that can no longer be used to generate positive conclusions from the theory. However, our approach does not make any different to the traditional approach for theories without superiority relation or defeater. This can be seen from Figures 4.6 to 4.9 that for theories with undisputed inferences (chain, circles, dag and tree), the traditional approach perform a bit better than our approach. This is due to the fact that defeasible theories under this situation can be inferenced directly without performing any transformations. However, in our approach, an additional step for inferiorly defeated rule verification is required for each conclusions to be inferred, which impose little negative impact (less than 5% on average) to the reasoning process. In some cases, our approach may also requires a little more memory than the traditional approach for storing the inferiorly defeated rules information.

On the contrary, and as predicted, our approach outperforms the traditional approach for theories with disputed inferences (teams and supTest) by several orders of amplitude, as can be seen from Figures 4.10 to 4.11. Our approach improve the defeasible inference process in both reasoning time and memory usage. For examples, to theory teams with $n = 6^{13}$, we have a total of 21844 defeasible rules and 21844 superiority relations. Our approach is able to complete the inference process in 4.13 seconds with maximum memory consumption of 287.25MB; while it takes 124.336 seconds with maximum memory consumption of 565.909 MB in the traditional approach. For the same theory with $n = 7$ we have 87380 rules and 43690 superiority relations. Our approach requires 27.946ms to complete the inference process with maximum memory consumption of 692.29MB; while the traditional approach does not able to complete the inference process due to memory restriction. The results are even more significant to supTest due to the existence of a long chain of related literals-superiority relations. For $n = 10000$, it takes about 4.50 seconds with 384.32MB maximum memory consumption in our

---

13 Notice again that in our performance evaluation, the theory sizes here refer to the value of $n$ that we used in generating the test theories. The number of rules, literals and superiority relations that appear in the theories should be calculated using the formulas as shown in Table 4.1.
approach, while it takes 1637.29 seconds with 740.27MB maximum memory consumption in the traditional approach. This is due to the fact that, after a rule $r_i$ is proven, it complementary rule $r_i^-$ becomes redundant and hence be removed from the theory according to Theorem 3.16, which eventually simplify the inference process at a level similar to that of a chain theory.

We now turn to the comparison between SPINdle (based on our new approach) and Delores. In algorithmic terms, both implementations start by initializing a set of conclusions that can be immediately be established: all facts are provable while those literals without rules for them are unprovable, and then proceeds by modifying the rules in the theory based on their underlying algorithms.

However, technically speaking, SPINdle compute all definite and defeasible conclusions of a defeasible theory; while Delores compute only the positive defeasible conclusions and derive the negative conclusions by a dual process. In this sense, SPINdle compute (on average) four times
Figure 4.8: Reasoning time and memory usage of theory “dag” under different approaches

Figure 4.9: Reasoning time and memory usage of theory “trees” under different approaches

Figure 4.10: Reasoning time and memory usage of theory “teams” under different approaches
Figure 4.11: Reasoning time and memory usage of theory “supTest” under different approaches

more conclusions than Delores.

Besides, SPINdle can handle literals and/or rules with modal operators (and with modal conversion and conflict resolution); while, on the contrary, Delores provides no function to this and can handle only propositional based defeasible theories. Moreover, SPINdle can handle theories imported directly from users and can perform all transformations internally as necessary; while in Delores, since the pre-processing transformations are not fully completed, it can only inference on regular form defeasible theory with no defeaters and superiority relations. That is, we have to transform the theories first prior to importing them to Delores.

Figures 4.12 and 4.13 (and their numerical values as shown in Tables E.1 and E.2 of the Appendix) below show the results of our experiments. Note that since Delores does not provide any information on memory usage, we can only compare the two implementations by the reasoning time. Besides, since the pre-processing transformations in Delores are not fully completed, all test theories are pre-transformed by an application written by us before importing to Delores. For the sake of a better comparison, the reasoning time present in the figures (and the tables) include only the time used on importing the test theories to the reasoners and the inferencing process, and excluding the time used on theory transformations.

Comparison of the behaviors of SPINdle and Delores, the two implementations has a small but significant overhead on start-up due to the initial conclusions set generation. Above this overhead, the cost of initialization is proportional to the size number of distinct literals that appear in the theory. Nevertheless, even though the reasoning time of both implementations are linearly proportional to the size of the theories, Delores is substantially more efficient than SPINdle when there are no disputing rules (as in chain, circle, dag and tree). However, after considering the number of conclusions being computed by SPINdle, which is almost four times of that computed by Delores, and the functionalities that SPINdle supported, we believe that this difference, even though considerably significant, but is acceptable. Not to mention the overhead introduced by the Java Virtual Machine.
On the contrary, Delores performs badly when disputing rules are common, and may have time growing exponentially in the problem size. As mentioned in [154], these performance drop may due to the increase in the number of distinct literals that appear in the undisputed inferences. However, this problem has no effect on SPINdle and it seems that our approach performs even better than Delores as the number of superiority relations appears in the theories increase (as in teams).

In order to test the performance of SPINdle under real-life application, we have tested the implementation using the UAV navigation theory described in [145], as shown in Appendix F. The theory consists of about a hundred rules and 50 superiority relations, and is used to govern the behaviors of a UAV under different situations while navigating in urban canyon.

Table 4.2 show the result of this test. Again, our approach performs significantly better than the traditional approach in both reasoning reasoning time and memory consumption.
Figure 4.13: Reasoning time for Theories with Disputed Inferences

<table>
<thead>
<tr>
<th>Approach</th>
<th>Superiority relation removal (ms)</th>
<th>Conclusions generation (ms)</th>
<th>Total Reasoning Time (ms)</th>
<th>Memory Usage (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional</td>
<td>47</td>
<td>156</td>
<td>203</td>
<td>1.52</td>
</tr>
<tr>
<td>New</td>
<td>-</td>
<td>93</td>
<td>93</td>
<td>1.36</td>
</tr>
</tbody>
</table>

Table 4.2: Reasoning time used in UAV navigation theory

### 4.3 Summary

In summary, Theorem 3.16 enables us to reason on a defeasible theory without removing the superiority relations. The essence of this method lies in the ability of identifying the set of inferiorly defeated rules (Section 3.3) that is redundant and can be removed from the theory and can preserve the representation properties of DL across different variants.

Our results show that, even though this theorem is simple, it helps in simplifying the subsequent process and the performance gains are significant as the number of superiority relations appear in the defeasible theory increase. It also help preserving the representation properties of defeasible theory under different variants.

In terms of implementation, SPINdle can be used as a standalone defeasible reasoner, or can be embedded in any Java applications as a defeasible rule engine. Its built-in data structure enables us to retrieve the set of rules associated with a particular literals in constant time, which subsequently enables us to manipulate the theory in an efficient way. Beside, the plug-in architecture of SPINdle also allows software developers or researchers to create their own plug-in, which provide users the freedom to represent defeasible theories according to their needs; and can also utilize the inference mechanisms provided by SPINdle to inference on the theories.

The current implementation covers both the traditional approach proposed by Maher, and the algorithms presented in this thesis. However, as in the case of defeasible theories repre-
sentations, the inference mechanisms in SPINdle can also be easily extended by users through implementing different plug-ins.
Chapter 5

Contextualized Ambient Intelligence through Distributed Defeasible Speculative Reasoning

Computing is moving towards pervasive and ubiquitous in which devices are expected to seamlessly connect and collaborate with each other to support human objectives, anticipating needs, and provide right information at the right time to the right person at the right level of subsidiary [202, 219]. In this new world, computing are no longer be tethered to desktops: users are mobilized. As they move across the environments, they are advocated to access a dynamic range of computing devices or software services through their handheld devices, and will have freedom to select their interests and/or resources to perform their computing tasks. To shed light on this, Pervasive Computing and Ambient Intelligence are considered to be key issues in the research and development of the next generations of Information and Communication Technologies (ICTs).

The study of ambient intelligence and pervasive computing has introduced lots of research challenges in the field of distributed AI during the past few years. These are mainly caused by the imperfect nature of the environment and the special characteristics of the entities that process and share the context information available [31]. As described in [125]:

… Ambient Intelligence implies a seamless environment of computing, advanced networking technologies and specific interfaces. This environment should be aware of the specific characteristics of human presence and personalities; adapt to the needs of users; be capable of responding intelligently to spoke or gesture indications of desire; and even result in systems that are capable of engaging in intelligent dialogue. Ambient Intelligence should also be unobtrusive – interaction should be relaxing and enjoyable for the citizens, and not involve a steep learning curve.

It is in general agreed on that most ambient intelligence processes are contextual in the sense that they rely on the environment, or context, where they are carried on. They should have the ability to apprehend the system’s environment [132]. To display these kinds of complicated behaviors,
each entity – an ambient agent, should have the ability to realize the system environment: where entities can identify themselves, be aware of other entities in the environment and can interoperate with each others at syntactic and semantic level [171], so as to proactively response to people (in the environment) intelligently according to their needs.

Ambient agents that work in such an environment can be developed independently. They are autonomous in nature, and can always be in total control of their own resources (local resources). They are expected to have their own goals, perception and capabilities, and can collaborate with each other to achieve their common goals. They depend on each other for query information, computing resources, forwarding requests, etc [163]. Such collaboration is an inclusionary process that promotes lateral communication among agents. When a change of the environment is detected, they may need to update/revise their knowledge, which may subsequently affecting the answers computed for their received queries. In that case, such belief revision must also be propagated to the querying agent through answer revision [147]. Consequently, the main tasks of reasoning in ambient intelligence are to: (i) detect possible errors, (ii) make estimates to the missing values, (iii) determine the quality and validity of the context data, (iv) transform context data into meaningful information, and (v) infer new implicit context information that may be relevant to the agent or its users [32].

However, in practice, it is often difficult to guarantee efficient and reliable communications between agents. For example, if an agent was deployed on an unreliable network, such as the Internet, or if an agent requires human interaction, then communications might be largely delayed or even fail. Besides, due to the dynamic (open and restricted) nature of the environment, agents may not know a priori all other entities will be present at a specific time frame nor can they be able to, or willing to, provide the information requested. Henricksen and Indulska [115] has characterized four types of imperfection that may appear in context information: unknown (as described), ambiguous, imprecise and erroneous. Ambiguity occurs when conflicting information about a context property is received from multiple sources. Imprecise appears when the information is a correct yet inexact approximation of the true value, which is common in sensor-derived information; while erroneous arises may due to machine failures or human errors.

Taking into account the high rate of context changes where users and ambient agents can move in-and-out from the context at any time and potentially vast amount of context data, the reasoning tasks become even more challenging. In short, belows are the main challenges in performing the reasoning tasks in ambient intelligence.

- Reasoning with highly dynamic, imperfect and open context.
- Extract and organize potentially vast amount of context data from multiple sources in a real-time fashion, in accordance to the computational capabilities of the devices and the qualities of wireless communications.
- Distributed reasoning and collective intelligence through disseminating contextual information inferred.
So far, ambient intelligent systems have not yet managed to handle all the challenges listed above. In terms of reasoning approaches, as has been surveyed in [33], most of the ambient intelligence systems are following the classical reasoning approaches and lacking of a reasoning model that has the abilities to handle cases that may inherit from uncertain (or in some cases missing) or ambiguous context information.

Regarding the distributed reasoning and collective intelligence, most of the systems developed so far have been based on fully centralized approach such that a designated entity is responsible for collecting the context data from all entities that appear in the context and reasoning on the data collected, which is an unrealistic assumption in ambient intelligence. As pointed out in [31], a context may not be restricted to a small room, an apartment, or an office; cases of broader areas must be considered. In addition, there is no guarantee on the connections between the ambient agents and the centralized reasoning entity. The communications may be unreliable or restricted under certain conditions (e.g., the range of the wireless transmitters).

To rectify these shortcomings, a fully decentralized approach seems to be a necessity. In the previous discussions our attentions have been limited to reasoning in local context, meaning that we only reason on defeasible theories and (contextual) facts that are potentially available and will not be affected by other entities or information that appear in the environment. We call this local reasoning.

In this chapter, we shift our attention to the problem of distributed reasoning, which combines the consequences of local theories using askable literals, and, based on the Multi-Context System paradigm [79, 81, 83], proposes a totally distributed approach for contextual ambient intelligence through distributed defeasible speculative reasoning, as a means of specifying and implementing the ambient intelligence systems.

As stated before, DL has been advocated as the suitable formal tool to reason over incomplete and contradictory information while preserving low computational complexity. Our approach model ambient agents as autonomous logic-based entities. Knowledge possessed by an agent is formalized as a local context theory and associations between the knowledge possessed by other ambient agents using askable literals. Inconsistencies and ambiguities in local context theory are handled by the semantics of DL and ambient agents are free to select the intuitions that they want to infer according to their (or users’) needs; while uncertainties or missing context information, on the other hand, will first be substituted by the default values used in the speculative computation process [170] and will be replaced by the “real” information when they are available. Even though it is assumed that context theory are locally consistent, the same assumption will not necessarily hold for the global knowledge base. This is due to the fact that context theories model possibly different view-points. Given the lack of centralized control, the union of local theories (the global theory) may result in inconsistencies caused by the askable literals that receive from multiple sources. To handle this situation, we have first to detect inconsistencies, and second to reason in spite of them in a satisfactory way.

The rest of this chapter is structured as follows. The next section describes three motivating scenarios from the ambient intelligence domains and discuss some of their common charac-
Section 5.2 present the background information and related work on multi-context systems, defeasibility of argumentation and speculative computation. We then extend the notion of modal defeasible logic and characterize the Distributed Defeasible Speculative Reasoning (DDSR) framework through extending the argumentation semantics presented in Section 2.3. Section 5.6 presents the operational model of the framework in the form of distributed queries processing with belief revision using speculative computation. Section 5.7 presents a prototypical implementation of the framework implemented on the top of JADE - Java Agent Development Environment [126], which allows agents running on different host to exchange information through FIPA messages. Besides, agents in the framework can also configure to provide either DDSR services to other agents as presented in this chapter, or distributed reasoning services as proposed in [4]. Concluding remarks in the last section summarize our main findings.

5.1 Motivating scenario from Ambient Intelligence domain

In the following, we describe some use case scenarios from the Ambient Intelligence domain and discuss the challenges of reasoning with the available context information. The central feature of the scenarios is that people (including the agent that work on behalf of a person) are at the forefront of the information society, and benefiting from services and applications that are seamlessly supported by new technologies.

5.1.1 Context-aware mobile phone

Context-aware applications employ implicit sensing and complex reasoning mechanisms to try to understand the current context and situations of users, make appropriate adaptations according to users preferences. This scenario, adapted from [31], describes a context-aware mobile phone that has been configured by Dr. Amber, who is a professor working at a local university, to decide when it should ring. Dr. Amber has the following preferences: His mobile phone should ring in case of an incoming call, unless it is in silence mode or he is giving a course lecture.

Consider the case that Dr. Amber is currently located in a university classroom “RA201”. It is class time, but Dr. Amber has just finished his lecture and remains in the classroom reading emails on his laptop. The mobile phone, running in normal mode, received an incoming call from his colleagues.

The mobile phone cannot decide whether it should ring based only on its local context knowledge, which includes information about incoming calls and the operation mode of the mobile phone. However, it has no knowledge of other important contextual parameters, such as Dr. Amber’s location and his current activities. Therefore, the mobile phone attempts to contact other ambient agents that are located nearby through the university’s wireless network, and import from them further context information that can be used to reach a decision.
To be able to determine whether Dr. Amber is in the process of giving a lecture, it requires two rules. The first rule states that if at the time of the call Dr. Amber has a scheduled lecture and is located in a university classroom, then Dr. Amber has a very high possibility of giving a lecture. Information about Dr. Amber’s class schedule can be imported from Dr. Amber’s laptop, which hosts Dr. Amber’s calendar. According to this, Dr. Amber has a scheduled class at the time of the call. Information about Dr. Amber’s current location can be imported from the university wireless network localization service, which locates Dr. Amber is in classroom “RA201” at the time of the call. The second rules states that if there is no activities in the classroom. If this is the case then it is unlikely that Dr. Amber is giving a lecture. Information about this can be imported from the classroom manager located in “RA201”. In this case, based on the local information about the state of the projector, and the number of people in the classroom that can be imported from a classroom sensor, which detects only one person, the classroom manager infers that there is no class activity.

Figure 5.1 depicts the overall information flow of this scenario. Eventually, Dr. Amber’s mobile phone will receive ambiguous information from different ambient agents in the environment. Information from Dr. Amber’s laptop, together with the information provided by the localization service leads to the conclusion that Dr. Amber is giving a lecture at the time of the incoming call; while information imported from the classroom manager leads to the contradictory conclusion that Dr. Amber is not teaching at that particular moment. To resolve this conflict, the mobile phone should have the ability to evaluate the information that it receives from various sources, which call for intelligibility in context-aware applications. That is, ambient agents should have the ability to represent to their users what context information the agents have inferred, how the reasoning took place, and what adaption has been performed as a result of the evaluation [25]. For instance, in the scenario, if the mobile phone is aware that information provided by the classroom manager is more accurate than the information provided
from Dr. Amber’s laptop, it will determine that Dr. Amber is not giving a lecture and hence reach the conclusion that it should ring.

5.1.2 Digital-me

The second scenario describes what living with ambient intelligence might be like for ordinary people in the future. It is envisioned that in the next century, each person will have a personal social and communication-based ah-hoc networking device, “Digital-me” (or something alike), that registers, processes and offers information on private lives. It aims at facilitating socially based networking and relations through providing a communication interface and at taking decisions base on users’ preferences in specific situations. It illustrates emergent communication and relation behaviors in the ambient intelligence landscape. For instance:

- **Managing existing relations**, to add, remove or update existing personal contacts, or communication-related professions and communities (such as doctors, engineers, priests, social workers, etc.).

- **Creating new relations**, to provide interface for self-expression, free chatting, net-games, profiling, matchmaking, etc. Associated domains of applications are people location services, data mining, internal security controls or surveillance systems.

- **Participating in virtual relations**, to provide a virtual environment of download, store, or process data that is created by the social communities or digital avatars.

It is three o’clock in the afternoon. Janet, sales manager and senior pharmacist of a local pharmacy warehouse, is working busily in her office and does not want to be excessively bothered during the office hour. Nevertheless, all the time she is receiving and answering incoming calls and mails.

Due to her job nature, and as are many of Janet’s friends and colleagues, she is very proud of being communicating and socializing with many different people around the world. She is wearing a small digital device, “Digital-me” which act as a digital avatar of herself, wherever she is working or traveling. This little device is both a learning device, learning about Janet from her interactions with the environment and activities that she has with her friends; and an acting device with communication, data collection and decision-making capabilities. Janet has programmed the device a little bit so as to formalize her identities and the way she envisage her relations, and will update it periodically.

An important functionality of Digital-me is the intelligence that it supports in enhancing user’s experience. From time-to-time, the device will acquire, store and analyze data about Janet, making part of the analyzed information available according to the situations and preferences set by Janet. For example, Digital-me will learn from the past experience and adopt to Janet’s “personality” as she enters her office. The room temperature and default lighting are set, and Janet’s daily schedule together with a selected list of news will be displayed on her office desktop.
Digital-me will also confronts the available data registered from Janet’s environment and her personal database (such as locations, voices, schedules, etc.) to match the situation with her private calls. Under normal situation, while Janet is in the middle of a meeting, Digital-me will answer the calls and store the messages to the virtual environment automatically so that Janet can hear the message back when she is free. However, if the call is made by Janet’s boss or from someone who is important to Janet, then in the first attempt, Digital-me will run a brief conversations with the intention of negotiating a delay while explaining the situation.

Simultaneously, while still at work, Janet’s Digital-me has caught a message from an elderly’s Digital-me located in a nearby shopping mall. The elderly has left his home without his medicine and would feel at ease if know where and how to find a similar drugs in an easy way. As a pharmacist, Janet has addressed his query in natural speech to his Digital-me. After knowing the health problem of that elderly and the drugs he is needed, Janet’s Digital-me than issue a query to the pharmacy warehouses nearby for the availability of drugs and offer the information to the senior. However, due to the privacy level set by Janet, the Digital-me has decided to retain Janet’s anonymity by neither revealing Janet’s identity nor offering direct help to the elderly, but to list the closest pharmacy warehouse with the alternate drugs available and their contacts. To avoid useless or overlapped information overload, this information is shared through the elderly’s Digital-me rather than with the elderly himself. After gathered all information and analyze on it, the elderly’s Digital-me will target the adequate information and communicate it with the elderly in the short-term. If possible, the Digital-me could also offer to establish a new relationship to the warehouse.

### 5.1.3 UAV Navigation in urban canyon

Typical Unmanned Autonomous Vehicles (UAVs) operate at high altitudes where the space is obstacle-free. However, when navigating within an urban environment, a UAV would have to deal with canyons that have 90-degrees bends, T-junctions and dead-ends [122], which impose another level of difficulties to the autonomous control of UAVs. In the sense of this, to be able to travel from one location to another, a UAV should be able to mimic the limited and usually repetitive conditioned responses, and be smart enough to make proper decisions when an undesired traffic condition (such as collision or traffic jam) might occur.

To illustrate the combination of techniques to model the Vehicle Routing Problem (VRP) in urban canyon, we have the following problem scenario:

Given a city map with specific targets and obstacles (Figure 5.2), a number of UAVs has to navigate through the city from a starting location to a destination without colliding with each other. There is a GPS enabled application that informs the UAVs about the current traffic situations and the locations of other UAVs. To navigate successfully, the UAVs have to follow some guidelines about how and when they should alter their route.

The above scenario revealed how a UAV should interact with its environment. It presumes an
execution cycle consisting of a phase where the UAV collects information through its sensors, decides an action and then applies this action [220].

In order to travel from one location to another, a UAV has to gather different types of information from the GPS monitor within a proximity range and detects if any traffic problems might appear. The Knowledge Base (KB) of a UAV is a well-documented limited set of behavioral interactions that describes the behavior of a UAV under different situations, in particular it contains (prescriptive) rules for determining who has right of way over who. It is complemented with formal logics (and in particular DL) to represent significant issues regarding the domain of operations.

In case of a possible collision, a UAV will utilize the information in its KB and incorporate into it the set of context-related information (such as traffic situation, information about the...
vehicles nearby, etc) and derive a safe direction of travel or eventually to temporary stop its motion. (Consider the scenario as shown in Figure 5.3 where vehicles $V_3$, $V_4$ and $V_5$ are moving towards the same location (the red circle) and collisions may occur if none of the vehicles alter their route.) This perception-action cycle (Figure 5.4) can be conceived not only as an issue of control, but also lays out the interface on how the UAVs should interact with the environment [40]. Here, the sensors (in our case the GPS monitor) collect information about the environment (as described above) and which is then combined them with the knowledge base for common-sense decision making. The behavior controller then reasons on these information and triggers the associated actuator(s) (whether to change its current travel direction, speed or even to stop its current motion) based on the conclusions derived.

5.1.4 Common characteristics of the scenarios

Intuitively, in an ambient environment, an ambient agent can receive partial\(^1\) answers from other ambient agents in the environment and then consolidate these partial and/or “local” answers into a global answers. The above scenarios describes the types of applications in which each ambient agents should be aware of the type and qualify of knowledge they receive from the environment and how (part of) this knowledge relates to its local knowledge. Specifically, the following assumptions have been implicitly made.

- There is an available means of communication in which an ambient agent can be used to exchange (a subset of) its knowledge with other agents in the environment.

- Each ambient agent has some computing and reasoning capabilities that can be use to support context-aware decision making according to the preferences decided by users, or

\(^1\)A representation is partial when it describes only a subset of a more comprehensive state of affairs.
defined by applications.

The challenges of reasoning with multi-context and context-aware decision making in the scenarios above include the following:

- Local context may be incomplete, meaning that there are situations in which agents may not have immediate access to all available context information in the environment. This may depend on the unavailability of agents, poor or unreliable communication channel, delayed due to user interactions, etc. In some situations, this may also be affected by the willingness of an agent to disclose or share part of its local knowledge to other agents in the environment.

- Context knowledge may be ambiguous. This is due to the fact that ambient agents in the environment are independent in nature. They can have their own belief and perception about the environment, which may be contradictory to each others.

- Context knowledge may be inaccurate, meaning that the information received from other agents may not truly reflect the most updated situations that appear in realities. For example, in the first scenario, it appears in Dr. Amber’s schedule that Dr. Amber is having a lecture during the time an incoming call has received. However, in the scenario, Dr. Amber has already completed his lecture and is remained in the classroom reading his email.

- The computational capabilities of the devices used in the scenarios above are restricted. Most of them do not have the abilities to accomplish the reasoning task in a real-time fashion, which may lead to a large computational overhead.

5.2 Background & Related work

5.2.1 Multi-Context System

Based on the seminal papers by McCarthy [157] and Giunchiglia [82] several formalizations of (inter-) contextual information flow and contextual reasoning have been proposed. The two most notable are the Propositional Logic of Context (PLC) proposed by McCarthy and Buvač [160], and the Multi-Context System (MCS) proposed by Giunchiglia and Serafini [83], which later have been associated with the Local Model Semantics proposed by Ghidini and Giunchiglia in [79]. In [197] the authors have undergone an in-depth comparison between PLC and MCS, and concluded that MCS constitute a more general formal framework of contextual reasoning. This result has later been supported by a more conceptual argument (partiality, approximation and proximity) of Benerecetti et al. [27].

A simple illustration of the main intuitions underlying MCS is provided by the so-called “magic-box” example, as depicted in Figure 5.5. In the figure, Mr.1 and Mr.2 are agents
looking at the box, which consists of six sectors, each sector possibly containing a ball, from different angles. The box is magic since neither Mr.1 nor Mr.2 can distinguish the depth inside it. As some sections of the box are out of sight, both agents have partial viewpoint of the box, which they believe to be true. Even though some of their beliefs may be the same, their respective interpretation of their beliefs are independent. Besides, the belief of one may also be meaningless to the others. For example, both Mr.1 and Mr.2 could agree on that there are two balls in the box. However, from the perspective of Mr.1 one of the balls is blue and another is green; while Mr.2 could believe that both balls are green. Moreover, neither of them could have the knowledge about the red ball that appears at the top right hand corner of the box. Any statement regarding the red ball is meaningless to both of them. The bottom line is that they both have their own knowledge and language in which they use to express their beliefs [32].

Another thing worth mentioning is that both of them may have (partial) access to each other’s belief about the box. For example, Mr.2 may have access to the fact that Mr.1 believes that there is a blue ball in the box. Mr.2 may interpret this fact in terms of his own language and adapt his believe accordingly. This mechanism is called information flow among contexts [44].

Intuitively, a MCS describes the availability and flow of information among a group of contexts (to a number of agents/person/databases/services/modules), where each context is a subset approximation theory of the world from some individual perspective [84]. The contexts themselves may be heterogeneous meaning that different logical languages and inference formalisms can be used and no notion of global consistency is required [44]. The information flow is modeled via so-called bridge rules which can refer in their premises to information from other contexts.

As described in [32], contextual reasoning is a combination of local reasoning, which is performed in a local context; and distributed reasoning, which takes into the account the relationships between local contexts. From the logical modeling perspective [190], as logical descriptions of agents are broken into a set of contexts, each holds a set of related formulae, we effectively get a form of many-sorted logic with the concomitant advantages of scalability and efficiency.
Besides, this makes it possible to build agents which use several different logics in a way that keeps the logics neatly separated, which subsequently increases the representational power of the logical agents or simplify agents conceptually. However, each contexts may encode assumptions that are not fully explicit, the relationships can be described only to a partial extent. Besides, there is no guarantee that every context knowledge in the MCS can be fully translated from one to another, nor the required context information is available at the time of reasoning (Figure 5.6).

Roughly, MCS can be divided into two types: monotonic and non-monotonic MCS. Examples of the former kind include [82, 159, 79]; while [44, 46, 187, 60, 34] are of the later kind. Note that the general MCS framework of [44] is of special interest as it generalizes the previous approaches in contextual reasoning and allows for heterogeneous and non-monotonic MCS, i.e., a system that may have different, possibly non-monotonic logics in its context, and model bridge relations between contexts as default rules, handling cases of incomplete and inconsistencies information from imported knowledge, and hence enable interlinking between different non-monotonic contexts. However, except [34], none of these approaches have taken the notion of priority or preference among imported knowledge into consideration, which could potentially be used for conflict resolution.

5.2.2 Defeasibility of Arguments

In general, the idea of defeasibility is linked to the idea of exceptions. That is, an argument is defeasible if it is eventually not applied due to the presence of some defeating condition in the context. It is a vital aspect of human cognitive behavior as it displaces the monotonic character of deductive reasoning and introduces uncertainty to the conclusions being derived.

Argumentation systems constitute a way to describe how argumentation uses conflicting information to construct and analyse arguments for and against certain conclusions. It may also involve weighting, comparing, or evaluating arguments.
As discussed in Section 2.3, the heart of an argumentation system is the notion of an *acceptable argument*. It is based on the assumption that the process of argumentation is built from two types of jobs: formulating the arguments (including the interactions that exists between them) and evaluating the validity of the arguments [52]. In logic programming terms, it refers to a set of formulae and then exhaustively lay out arguments and counterarguments, where a counter-argument is either a rebut (i.e., negates the claim of the argument) or an undercut (i.e., negates the support of the argument) [29]. In preference-based argumentation systems (PBAS) [6], this process is enabled by a preference relation, which is either implicitly derived from elements of the underlying theory [206, 200], or explicitly defined on the set of arguments [128, 101, 26, 7, 178].

Our framework extends the argumentation semantics of DL presented in Governatori et al. [101] with the notion of askable literals and preference among system contexts. In our framework, defeasibility of argument may exists at both local and external level. The former refers to conflicting arguments that may arise when inferencing with agent’s local argument (Definition 5.23). In this case, the acceptability of argument is totally determined based on the proof theory as defined by defeasible logic such that arguments using superior rules win. While the latter one corresponds to the situations when mapping arguments (Definition 5.24) are considered. In this case, preference between conflicting arguments are derived both from the structure of arguments - arguments based on local rules are considered stronger than mapping rules, and from a preference ordering calculated based on the information sources (Definition 5.25).

Here we would like to emphasis the notion of *context* is of great importance in our framework. As mentioned in [3], the sentence “I trust my doctor for giving me advice on medical issues but not on financial ones” is only one example that shows how important contexts can be. One of the limitations of current PBAS is that they are not able to take into account different preorderings on the beliefs and the preorderings considered should be *contextual preferences* meaning that “preferences” should depend upon a particular context [7]. Hence, and for the sake of simplicity, the *preference ordering* defined in this chapter considered for one local context, meaning that we do not distinguish the evaluation of reputation in different contexts, which may possibly exists in realities.

### 5.2.3 Speculative Computation

Speculative computation is an implementation technique that aims at speeding up the execution of programs, by computing piece of information in advance, without being sure whether these computations are actually needed [43]. It is an eager computational approach that can yield improvements over conventional approaches to parallel computing by providing: (i) a means to favor the most promising computations; and (ii) a means to abort computation and reclaim computation resources [170]. It is a process which use default hypothesis as a tentative answer and continue the process when the resources information cannot be obtained in time [216].

As discussed before, agents in ambient intelligence environment will exchange knowledge, ask questions and provide services to each other. However, as arises in human organizations,
communication may be an issue. Information flow may be largely delayed or even failed, which subsequently leads to incompleteness of information in the reasoning structure [53]. In such unideal situations, when problem-solving is at stake, framework of speculative computation for multi-agents system were proposed [195, 194, 53, 123, 193, 124]. Their aims are to resolve the above incompleteness problem in query answering between agents by using a default as tentative answer (which are somehow most likely to be the answers according to the current context), and continues the computation speculatively without much waiting for the response [124]. If the revised information is different from the default or the values that are currently used, the current process will be suspended/preempted and an alternate line of computation will be started. (Figures 5.7 and 5.8 show the difference between ordinary computation and speculative computation.) Besides, it also features the possibility of reusing parts of the computation already done during the phases when answers arrive and answer revision is required, as the partially computed suspend process may becomes active again and resume its computation.

Even through one may argue that there may be a high possibility that the computation done (using default hypothesis) at the beginning of the process may be wasted as consistency between the default hypothesis and arguments in the external context may not exist. However, in an ambient environment, the agent has no idea of whether or when a returned answer, or its new revising answers will arrive. Besides, while waiting for replies, an agent may sit idle if no default were used, which is also considered as another kind of waste of resources. In either case, the main overhead is the extra computation required to decide, during answer arrival phase, whether the revised answers are consistent with the existing one [147]. Besides, as pointed out in [170], some computations may be more promising than others. As resources are limited, it is important to use them efficiently. So, instead of letting an agent sitting idle while waiting for answers from other agents, it is better if we can allocate resources to computation to favor some promising computations.

Belief revision, on the other hand, is an important feature in speculative computation for both the sake of flexibility and efficiency. It allows agents to process information before it is
Figure 5.8: Speculative Computation (simplified)
complete and save the time if the prior information is later entailed [53]. In [195, 216] the authors has limited the possibility for speculative computation to the master agent and the returned answers from slave agents are final since there is no possibility of revising them. If, however, we allow every agents to perform speculative computation, an ambient agent might revise her previous answers whenever a change of context is detected, which might subsequently create a chain reaction of belief revision among the ambient agents [53]. This phenomenon has first been observed in [194] in which a revisable speculative computation framework, which can handle belief revision of an answer even during execution, was proposed.

Since then the framework has been extended to solve more general problems using constraints processing [147, 121] where the answer sets can be a set of constraints over the variables in the query. However, insofar as the methods proposed only consider situations where agents in the systems are of hierarchically structured and queries can only be issued from agents at the upper level to agents at lower level, whereas agents in ambient intelligence environment are of no centralized control or hierarchical organization in nature.

Among the operational models proposed, the one in [53] is the most complex but powerful. A practical implementation for it is pretty much desired not only for both proof-of-context testing and benchmark investigation, but also for discovering future development and improvement of the model [148].

**Relation between Speculative Computation and Abduction**

First coined by C.S. Pierce [172] abduction is defined as the inference process of forming hypothesis that explains some plausible phenomena or observations. Often abduction has been interpreted broadly as any form of “inference to the best explanation” [127] where best refers to the fact that the generated hypothesis is subjected to some optimality criterion. This broad definition provides a platform for studying different phenomena involving some form of hypothetical reasoning. Studies of abduction range from philosophical treatments of human scientific discovery down to formal and computational approaches in different formal logics, across many areas of AI [65].

Technically speaking, given a program \( P \), a goal \( G \) and a set of literals \( \Delta \) representing the default hypotheses, speculative computation of the goal \( G \) is defined as:

- \( P \cup H \models G \),
- \( P \cup H \) is consistent, and
- \( H \subseteq \Delta \).

where \( H \) is the set of hypotheses that supplement the missing information in \( P \) to derive \( G \).

In this sense, speculative computation is characterized by abduction [195]. Comparing the two, justification of hypothesis \( H \) in abduction depends partly on evaluating the quality of the discovery process and the pragmatic considerations which include the possibility of gathering further evidence before deciding; while in speculative computation, hypothesis might be replaced
by the “real” information which are known only during the computation of $\mathcal{G}$, implying that the answer of the goal might change accordingly. By contrast, in abduction, the observation $\mathcal{G}$ is given initially and the set $\Delta$ of hypothesis is, once assumed, never revised during the computation of explanations.

5.3 Extending Modal Defeasible Logic

Having the basic of DL is not sufficient enough. The most expressive part of developing a MCS system is on creating the framework of norms, etc. for representing the behaviors of the ambient agents. In this section, we follow the idea presented by [94] (cf. Section 2.4) and introduce three modal operators: BEL (Believe), OBL (Obligation), INT (Intentions), to the theory when formalizing the behaviors of ambient agents.

To recap, as discussed in Section 2.7.2.2, the intuitive reading of a preferred expression like $a \otimes b \otimes c$ is that, under normal situation, $a$ is preferred, but if $\neg a$ is the case, then $b$ is preferred; if $\neg b$ is the case, given $\neg a$, then the third choice $c$ is preferred. We call $\otimes$ the preference operator over a set of conclusions, whose properties are given in Definition 5.1, as shown again below.

**Definition 5.1** (Preference operator). A preference operator $\otimes$ is a binary operator satisfying the following properties: (1) $a \otimes (b \otimes c) = (a \otimes b) \otimes c$ (associativity); (2) $\bigotimes_{i=1}^{n} a_i = (\bigotimes_{i=1}^{k-1} a_i) \otimes (\bigotimes_{i=k+1}^{n} a_i)$ where exists $j$ such that $a_j = a_k$ and $j < k$ (duplication and contraction on the right).

**Definition 5.2.** Let MOD = \{BEL, OBL, INT\} be a set of modal operators and $P$ a set of propositional atoms.

- If $p \in P$, then $p$ and $\neg p$ are literals.
- If $l$ is a literal, and $\square$ is a modal operator, then $\square l$ and $\neg \square l$ are modal literals.
- If $l_1, \ldots, l_n$ are literals, then $l_1 \otimes \ldots \otimes l_n$ is an $\otimes$-expression.

Notice that we do not allow nesting of modal operators. This is a simplification aimed at keeping the system manageable, but it does not pose severe limits for our purposes (see [93] and [94] for discussions about this limitation, and how to extend the language to overcome this limitation).

**Definition 5.3.** A rule is an expression

$$r : \phi_1, \ldots, \phi_n \rightarrow \square \psi$$

where

- $r$ is a unique label, identifying the rule:
- $\phi_i$, $1 \leq i \leq n$, is either a literal or a modal literal.
• $\rightarrow, \Rightarrow, \sim \in \{\rightarrow, \Rightarrow, \sim\}$, and $\Box$ is a modal operator;

• $\psi$ is either a literal or, if $\rightarrow=\Rightarrow$, an $\otimes$-expression.

According to the above definition it is not possible to have modal literals in the conclusion of a rule (so modal operators cannot occur in $\otimes$-expressions). This is in agreement with the idea that rules are used to introduce modal operators with their conclusions.

Given a set $R$ of rules, we denote the set of all strict rules in $R$ by $R_s$, the set of strict and defeasible rules in $R$ by $R_{sd}$, the set of defeasible rules in $R$ by $R_d$, and the set of defeaters in $R$ by $R_{df}$. $R[q]$ denotes the set of rules in $R$ with consequent $q$. For some $i$, $1 \leq i \leq n$, such that $c_i = q$, $R[c_i = q]$ denotes the set of rules with the head $\otimes_{j=1}^{n} c_j$. Finally, $R^\Box$ denotes the subset of $R$ where the arrow of the rule is labelled with the modal operator $\Box$.

**Definition 5.4.** The conversion relation $\text{Convert}$ is defined as follows:

$$\text{Convert} \subseteq \text{MOD} \times \text{MOD}$$

The conflict relation $\text{Conflict} \subseteq \text{MOD} \times \text{MOD}$ is such that

$$\forall X, Y \in \text{MOD}, \text{Conflict}(X, Y) \Rightarrow \neg(\text{Conflict}(Y, X)) \quad (\text{asymmetry})$$

Specifically, for realistic social agents, we are going to set the following conflicts and conversions:

$$C = \{\text{Convert(BEL, INT)}, \text{Conflict(BEL, OBL)},$$

$$\text{Conflict(BEL, INT), Conflict(OBL, INT)}\}$$

**Definition 5.5.** A defeasible modal theory is a structure

$$D = (F, R^{\text{BEL}}, R^{\text{INT}}, R^{\text{OBL}}, >, C)$$

where

• $F$ is a finite set of facts, i.e., a set of literals and modal literals;

• $R^{\text{BEL}}, R^{\text{INT}}, R^{\text{OBL}}$ are three finite sets of rules such that each rule has a unique label;

• The superiority relation $>$ is such that $>^{sm} \cup >^{\text{Conflict}}$, where $>^{sm} \subseteq R^X \times R^X$ such that if $r > s$, then if $r \in R^X[p]$ then $s \in R^X[\sim p]$ and $>^{\text{Conflict}}$ is such that

$$\forall r \in R^X[p], \forall s \in R^Y[\sim p], \text{if Conflict}(X, Y), \text{then } r >^{\text{Conflict}} s$$

• $C$ is the set of conflict and conversion relations given above for realistic social agents.
The construction of the superiority relation combines two components: the first \( >^{sm} \) considers pairs of rules of the same mode. This component is usually given by the designer of the theory and captures the meaning of the single rules, and thus encodes the domain knowledge of the designer of the theory. The second component, \( >^{\text{Conflict}} \) is obtained from the rules in a theory and depends on the meaning of the modalities.

**Definition 5.6.** Given a defeasible modal theory \( D \), a proof in \( D \) is a linear derivation, i.e., a sequence of labelled formulas of the type \( +\Delta q \), \( -\Delta q \), \( +\Box q \) and \( -\Box q \), where the proof conditions defined in the rest of this section hold.

The interpretation of the proof tags for modal defeasible logic is the same as that of standard defeasible logic, the only modification is that the conclusion has been obtained, at least in the last step, using rules of mode \( \Box \).

We start with some terminology. As was explained, the following definition states the special status of belief rules, and that the introduction of a modal operator corresponds to being able to derive the associated literal using the rules for the modal operator.

**Definition 5.7.** Let \( \# \in \{\Delta, \Box\} \), \( \Box \in \text{MOD} \) and \( P = (P(1), \ldots, P(n)) \) be a proof in \( D \). A (modal) literal \( q \) is \( \# \)-provable in \( P \) if there is a line \( P(m) \) of \( P \) such that either

1. \( q \) is a literal and \( P(m) = +\#\text{BEL} q \) or
2. \( q \) is a modal literal \( \Box p \) and \( P(m) = +\#\Box p \) or
3. \( q \) is a modal literal \( \neg\Box p \) and \( P(m) = -\#\Box p \).

A literal \( q \) is \( \Box \)-rejected in \( P \) if there is a line \( P(m) \) of \( P \) such that either

1. \( q \) is a literal and \( P(m) = -\#\text{BEL} q \) or
2. \( q \) is a modal literal \( \Box p \) and \( P(m) = -\#\Box p \) or
3. \( q \) is a modal literal \( \neg\Box p \) and \( P(m) = +\#\Box p \).

The definition of \( \Delta \Box \) describes just forward chaining of strict rules:

\[ +\Delta \Box : \text{If } P(n + 1) = +\Delta \Box q \text{ then} \]
\[ (1) q \in F \text{ if } X = \text{BEL} \text{ or } \Box q \in F \text{ or} \]
\[ (2) \exists r \in R^\Box_s[q] : \forall a \in A(r) \text{ } a \text{ is } \Delta \text{-provable or} \]
\[ (3) \exists r \in R^\Box_s[q] : \text{Convert}(\Box, \Box) \in \mathcal{C} , A(r) \neq \emptyset , \forall a \in A(r) \text{ } \Box a \text{ is } \Delta \text{-provable.} \]

\[ -\Delta \Box : \text{If } P(n + 1) = -\Delta \Box q \text{ then} \]
\[ (1) q \notin F \text{ if } X = \text{BEL} \text{ and } \Box q \notin F \text{ and} \]
\[ (2) \forall r \in R^\Box_s[q] \exists a \in A(r) : a \text{ is } \Delta \text{-rejected and} \]
\[ (3) \forall r \in R^\Box_s[q] : \text{if } \text{Convert}(\Box, \Box) \in \mathcal{C} \text{ then } \exists a \in A(r) \text{ } \Box a \text{ is } \Delta \text{-rejected.} \]
For a literal $q$ to be definitely provable we need to find a strict rule with head $q$, whose antecedents have all been definitely proved previously. And to establish that $q$ cannot be proven definitely we must establish that for every strict rule with head $q$ there is at least one of antecedent which has been shown to be non-provable. Condition (3) says that a belief rule can be used as a rule for a different modal operator in case all literals in the body of the rules are modalised with the modal operator we want to prove. Thus, for example, given the rule $p, q\rightarrow_{\textsc{bel}} s$, we can derive $+\Delta_{\textsc{int}} s$ if we have $+\Delta_{\textsc{int}} p$ and $+\Delta_{\textsc{int}} q$.

Conditions for $\partial_\Box$ are more complicated. First we have to account for $\otimes$-expressions. Then we have to define when a rule is applicable or discarded. A rule for a belief is applicable if all literals in the antecedent of the rule are provable with the appropriate modalities, while the rule is discarded if at least one of the literals in the antecedent is not provable. As before, for the other types of rules we have to take conversions into account. We have thus to determine conditions under which a rule for $Y$ can be used to directly derive a literal $q$ modalised by $\Box$. Roughly, the condition is that all the antecedents $a$ of the rule are such that $+\partial_\Box a$.

**Definition 5.8.** Given a derivation $P$, $P(1..n)$ denotes the initial part of the derivation of length $n$. Let $X, Y, Z \in \text{MOD}$.

- A rule $r \in R_{sd}[c_i = q]$ is applicable in the proof condition for $\pm \partial_X$ iff
  
  (1) $r \in R^X$ and $\forall a \in A(r)$, $+\partial_{\textsc{bel}} a \in P(1..n)$ and $\forall Z a \in A(r)$, $+\partial_{\Box} a \in P(1..n)$, or
  
  (2) $r \in R^Y$, $\text{Convert}(Y, X) \in C$, $A(r) \neq \emptyset$ and $\forall a \in A(r)$, $+\partial_X a \in P(1..n)$.

- A rule $r$ is discarded in the condition for $\pm \partial_X$ iff
  
  (1) $r \in R^X$ and $\exists a \in A(r)$ such that $-\partial_{\textsc{bel}} a \in P(1..n)$ or $\exists Z a \in A(r)$ such that $-\partial_{\Box} a \in P(1..n)$; or
  
  (2) $r \in R^Y$ and, if $\text{Convert}(Y, X)$, then $\exists a \in A(r)$ such that $-\partial_{\Box} a \in P(1..n)$, or
  
  (3) $r \in R^Z$ and either $-\text{Convert}(Z, X)$ or $-\text{Conflict}(Z, X)$.

We are now ready to provide proof conditions for $\partial_X$:

$+\partial_X$:\ If $P(n+1) = +\partial_X q$ then

1. $+\Delta_X q \in P(1..n)$ or
2. $-\Delta_X \neg q \in P(1..n)$ and
   
   (2.1) $r \in R_{sd}[c_i = q]$ such that $r$ is applicable, and $\forall i, +\partial_{\textsc{bel}} c_i \in P(1..n)$; and
   
   (2.2) $\forall s \in R[c_j = \neg q]$, either $s$ is discarded, or $\exists j' < j$ such that $-\partial_{\textsc{bel}} \neg c_{j'} \in P(1..n)$, or
   
   (2.3) $\exists t \in R[c_k = q]$ s.t. $r$ is applicable, $t > s$

   $\forall k' < k$, $-\partial_{\textsc{bel}} c_{k'} \in P(1..n)$ and

   either $t, s \in R^Z$ or $\text{Convert}(Y, X)$ and $t \in R^Y$.  

114
\( -\partial_X q \): If \( P(n + 1) = -\partial_X q \) then

1. \(-\Delta_X q \in P(1..n)\) and either
2. \((2.1) +\Delta_X \sim q \in P(1..n)\) or
   \( (2.2) \forall r \in R_{sd}[c_i = q], \) either \( r \) is discarded or
   \( \exists i' < i \) such that \(-\partial_{\text{BEL}} \sim c_{i'} \in P(1..n)\), or
3. \((2.3) \exists s \in R[c_j = \sim q], \) such that \( s \) is applicable and
   \( \forall j' < j \), \(+\partial_{\text{BEL}} \sim c_{j'} \in P(1..n)\) and
   \( (2.3.1) \forall t \in R[c_k = q] \) either \( t \) is discarded, or \( t \neq s \), or
   \( \exists k' < k \) such that \(+\partial_{\text{BEL}} c_{k'} \in P(1..n)\) or
   \( t \in R^Z, s \in R^{Z'}, Z \neq Z' \) and
   if \( t \in R^Y \), then \(-\text{Convert}(Y,X)\).

For defeasible rules we deal with \( \otimes \) formulas. To show that \( q \) is defeasibly provable we have two choices: (1) We show that \( q \) is already definitely provable; or (2) we need to argue using the defeasible part of a theory \( D \). For this second case, three (sub)conditions must be satisfied. First, we require that there must be a strict or defeasible rule for \( q \) which can be applied \( (2.1) \); in case the conclusion of the rule is an \( \otimes \)-expression, and the conclusion we want to prove is not the first element, we have to ensure that for every element of the \( \otimes \)-expression that precedes \( q \) the complement is provable with belief mode. Second, we need to consider possible reasoning chains in support of \( \sim q \), and show that \( \sim q \) is not definitely provable \( (2.2) \). Third, we must consider the set of all rules which are not known to be inapplicable and which permit to get \( \sim q \) \( (2.3) \); the second part of this condition checks that we can use \( \sim q \) if this is not the first element of the \( \otimes \)-expression corresponding to the head of the rule. The idea is that an obligation has not been violated, similarly for intentions. Essentially, each such a rule \( s \) attacks the conclusion \( q \). For \( q \) to be provable, \( s \) must be counterattacked by a rule \( t \) for \( q \) with the following properties: (i) \( t \) must be applicable, and (ii) \( t \) must be stronger than \( s \). Thus each attack on the conclusion \( q \) must be counterattacked by a stronger rule. In other words, \( r \) and the rules \( t \) form a team (for \( q \)) that defeats the rules \( s \). However, since we can have rules for different modes, we have to ensure we have the appropriate relationships among the rules. Thus clause \( (2.3.1) \) prescribes that either the rule that attacks the conclusion we want to prove \( (s) \) and the rule used to counterattack it (i.e., \( t \)) have the same mode (i.e., \( s, t \in R^Z \)), or that \( t \) can be used to produce a conclusion of the mode we want to prove (i.e., \( t \in R^Y \) and \( \text{Convert}(Y,X) \)). \(-\partial_X q \) is defined in an analogous manner.

**Properties of Modal Defeasible Logic**

In this section we are going to explore some properties of the logic that make the logic amenable for applications.

The first property is that Modal Defeasible Logic with the preference operator \( \otimes \) is consistent and coherent. Consistency means that for any literal and proof tag it is not possible to prove and reject the literal (with one and the same proof tag). Coherence means that the inferential
The mechanism of defeasible logic does not allow us to defeasibly derive both \( p \) and \( \neg p \) with the same mode, unless the monotonic part of a theory (i.e., facts and strict rules) is inconsistent; and even in that cases the inconsistency does not propagate to the entire theory. In this respect defeasible logic is paraconsistent.

**Proposition 5.9.** Let \( D \) be modal defeasible theory, \( l \) be a literal, \( \# \in \{\Delta, \partial\} \), and \( \Box \in \text{MOD} \). It is not the case that \( D \vdash +\#_\Box l \) and \( D \vdash -\#_\Box l \).

**Proof.** The proof is an immediate consequence of the framework used to define the proof tags [107]. The proof tags for the negative proof tags, i.e., \( -\Delta \) and \( -\partial \), are the ‘strong’ negation of that for the positive proof tags [15]. The strong negation of a formula is closely related to the function that simplifies a formula by moving all negations to an innermost position in the resulting formula and replaces the positive tags with the respective negative tags and vice-versa.

**Proposition 5.10.** Let \( D \) be modal defeasible theory, \( l \) be a literal, \( \Box \in \text{MOD} \). \( D \vdash +\partial_\Box l \) and \( D \vdash +\partial_\Box \neg l \) iff \( D \vdash +\Delta_\Box l \) and \( D \vdash +\Delta_\Box \neg l \).

**Proof.** The right to left direction is immediate given clause (1) of the definition of the proof conditions for \( +\partial \).

For the left to right direction, for BEL, since we cannot have \( \otimes \) in the conclusion of rules, see Proposition 5.5 [18]. For the other cases let us reason as follows: Let us suppose that we have the left hand side of the iff but not the right hand one. This means that either we have (i) only one of \( D \vdash +\Delta_\Box l \) and \( D \vdash +\Delta_\Box \neg l \) or (ii) none of them.

For (i), let us assume that we have \( D \vdash +\Delta_\Box l \) (the case where we have \( D \vdash +\Delta_\Box \neg l \) is symmetrical). Since we do not have \( D \vdash +\Delta_\Box \neg l \), then condition (2.1) of \( +\partial_\Box \) holds, that is, \( -\Delta_\Box \neg l \), thus condition (1) of \( -\partial_\Box \) holds, and we know \( +\Delta_\Box l \), thus it holds that \( D \vdash -\partial_\Box \neg l \), and thus from Proposition 5.9 we get a contradiction.

For (ii): First of all, it is easy to verify that no rule can be at the same time applicable and discarded for the derivation of \( \pm\partial_\Box l/\neg l \). Then, since both \( +\partial_\Box l \) and \( +\partial_\Box \neg l \) holds, then we have that there are applicable rules for both \( l \) and \( \neg l \). This means that clause (2.3.1) holds for both \( +\partial_\Box l \) and \( +\partial_\Box \neg l \). Therefore, for every applicable rule for \( l \) there is an applicable rule for \( \neg l \) stronger than the rule for \( l \), and symmetrically, for every applicable rule for \( \neg l \) there is an applicable rule for \( l \) stronger than the rule for \( \neg l \). The set of rules in a theory is finite, thus, this means that the situation we have just described is only possible if there is a cycle in the transitive closure of the superiority relation. Accordingly, we have a contradiction, because the superiority relation of defeasible theory is acyclic (the transitive closure of the relation does not contain cycles.).

In [94], it was proved that the modal defeasible logic obtained by extending defeasible logic with BIO modal operators (BEL, INT, and OBL) and the mechanism of conversion and conflict still has linear complexity (in the size of a theory, where the size of the theory is given by the number of rules and number of distinct literals). In [107] the complexity result was extended to

116
deontic defeasible logic with the $\otimes$-operator. The techniques used to prove the two complexity results for modal operators and $\otimes$ can be combined. In the rest of this section we propose algorithms for computing the extension of a modal defeasible theory based on the result just discussed. The algorithms proposed heretofore implement both modal operators and $\otimes$-expressions extend the algorithm proposed by Maher [151].

Before presenting the algorithms we introduce some notation used in it. Given a defeasible modal theory $D$, a modal operator $\Box \in \text{MOD}$, and a set of rules $R$ (such that the rules appears in the theory); for $r \in R$, $r_{\sup} = \{ s : (s, r) \in > \}$ and $r_{\inf} = \{ s : (r, s) \in > \}$. The Herbrand Base of $D$, $HB_D$ is the set of literals such that the literal or its complement appears in $D$, where ‘appears’ means that it is could be a subformula of a modal literal occurring in the theory. The Modal Herbrand Base of $D$, $HB^\Box = \{ \Box_i l : \Box_i \in \text{MOD}, l \in HB_D \}$. Given an $\otimes$-expression $c = a_1 \otimes \cdots \otimes a_{i-1} \otimes a_i \otimes a_{i+1} \otimes \cdots \otimes a_n$, the operations $c!a_i$ (truncation at $a_i$) and $c \ominus a_i$ (removal of $a_i$) are defined as follows:

$$c!a_i = a_1 \otimes \cdots \otimes a_i$$

$$c \ominus a_i = a_1 \otimes \cdots \otimes a_{i-1} \otimes a_{i+1} \otimes \cdots \otimes a_n$$

The idea of the algorithms is to perform a series of transformations that reduce a defeasible modal theory into a simpler equivalent theory.

To simplify the presentation of the algorithm to compute the definite extension, i.e., the set of conclusions that can be proved with $\pm \Delta$, we assume that the set of facts is empty. This can be easily achieved by replacing every literal $l$ in $F$ with the rule $\rightarrow \text{BEL} f$, and every modal literal $\Box l$ with the rule $\rightarrow \Box l$.

Algorithm 5.1 starts by initialising the global sets of definite conclusions to the empty sets (lines 1 and 2). In addition for any conversion relationship it identifies the set of rules that can be used for a conversion (line 3). Given a conversion $\text{Convert}(\Box_i, \Box_j)$ a rule for $\Box_i$ can be used to derive a conclusion of type $\Box_j$, if the body of the rule is not empty and all the elements of the body are literals (not modal literals) and they are derived with mode $\Box_j$ (see clause 3 of the conditions for $\pm \Delta \Box$).

After the initialisation, the algorithm begins its main loop. At each cycle of the loop we reset the set of definite conclusions that are derived during a cycle of the loop (lines 5–6). Then, we scan the modal literals in the Modal Herbrand Base. For each of such modal literals, we check whether there are strict rules that can produce it. This means that for a modal literal $\Box l$ we have to check the rule of mode $\Box$ as well as the rule for all modes involved in a conversion $\text{Convert}(\Box_i, \Box)$. If there are no such rules (line 8), then clauses (2) and (3) of the conditions for $\pm \Delta \Box$ are vacuously satisfied, thus we know that we can prove $\neg \Delta \Box l$, thus we add it to the set of local negative conclusions (line 9). Lines 10–12 take care of the effects of the conclusion $\neg \Delta \Box l$. If a rule has $\Box l$ in the body (or $l$ if $\Box = \text{BEL}$, given the reflexivity of the BEL operator), then the rule is discarded (it cannot longer be used to prove a positive conclusion), and thus we can remove such rules without affecting the conclusions that can be derived from the theory.
Algorithm 5.1: ComputeDefinite

Algorithm: ComputeDefinite

Data: \( \mathcal{D} = (F, R, >) \): a defeasible theory

Result: \( +\Delta_\Box \): set of definitely provable conclusions

Result: \( -\Delta_\Box \): set of definitely not provable conclusions

1. \( +\Delta_\Box \leftarrow \emptyset \)
2. \( -\Delta_\Box \leftarrow \emptyset \)
3. \( R_{s_i, s_j} \leftarrow \{ r \in R_{s_i}^\Box : \text{Convert}(\Box_i, \Box_j), A(r) \neq \emptyset, A(r) \text{ is a set of literals} \} \)
4. repeat
5. \( \Delta_\Box^+ \leftarrow \emptyset \)
6. \( \Delta_\Box^- \leftarrow \emptyset \)
7. for \( \Box l \in HB^\Box \) do
8. if \( R_{s_i}^\Box[l] \cup R_{s_i, s_j}^\Box[l] = \emptyset \) then
9. \( \Delta_\Box^- \leftarrow \Delta_\Box^- \cup \{l\} \)
10. \( R_s \leftarrow R_s - (\{ r \in R_s : \Box l \in A(r) \} \cup \{ r \in R_s : l \in A(r), \Box = \text{BEL} \}) \)
11. \( R_s \leftarrow \{ A(r) - \{\neg \Box l \} \rightarrow_X C(r) : r \in R_s \} \)
12. \( R_{s_i, s_j}^\Box \leftarrow R_{s_i, s_j}^\Box - \{ r \in R_{s_i, s_j}^\Box : l \in A(r) \} \)
13. \( HB^\Box \leftarrow HB^\Box - \{\Box l\} \)
14. if \( \exists r \in R_{s_i}^\Box[l] \cup R_{s_i, s_j}^\Box[l] : A(r) = \emptyset \) then
15. \( \Delta_\Box^+ \leftarrow \Delta_\Box^+ \cup \{l\} \)
16. \( R_s \leftarrow \{ A(s) - \{\Box l \} \rightarrow_X C(s) : s \in R_s \} \)
17. \( R_s \leftarrow \{ A(s) - \{l\} \rightarrow_X C(s) : s \in R_s, \Box = \text{BEL} \} \)
18. \( R_{s_i, s_j}^\Box \leftarrow \{ A(r) - \{l\} \rightarrow_{\Box_i} C(r) : r \in R_{s_i, s_j}^\Box \} \)
19. \( HB^\Box \leftarrow HB^\Box - \{\Box l\} \)
20. \( +\Delta_\Box \leftarrow +\Delta_\Box \cup \Delta_\Box^+ \)
21. \( -\Delta_\Box \leftarrow -\Delta_\Box \cup \Delta_\Box^- \)
22. until \( \Delta_\Box^+ = \emptyset \) and \( \Delta_\Box^- = \emptyset \)
Similarly if \( l \) appears in the body of a conversion rule (line 12). In addition \(-\Delta_\Box l\) makes \(-\Box l\) \( \Delta \)-provable, thus we can safely remove the occurrences of such modal literal from the body of the rules. The modal literal no longer contributes for the rule being applicable or rejected, these two properties now depend on the other (modal) literals in the body of the rules. After this step we remove \( \Box l \) from the modal Herbrand Base (line 13). We have already assessed the status of it and there is no need to further iteration on it.

The next step is to figure out whether the literal is positively provable. The strategy we use is to see whether there is a rule with empty body among the rules that can lead the modal literal, clauses (2) and (3) of \(+\Delta_\Box\). Notice that for conversion, we first create a set of rules that can be used for conversion \( (R_i^2, B_j) \), and we do not look for rules that satisfy the condition among all rules for a mode involved in a conversion. In lines 17–19, we remove (modal) literal that do not further influence the outcome of the computation. At the end of this phase we remove the modal literal for the Modal Herbrand Base, thus we do not have to consider it again in the loop.

In the main loop (lines 7–22) we have two blocks: lines 7–14 and lines 15–21. The applicability conditions for those two blocks are mutually exclusive, and when of the two succeeds, the output of the cycle is not empty, and at the same time we reduce the complexity of the theory, either by removing rules or by removing literals from the body of rules. In case that both blocks are unsuccessful, then there are no modifications in the theory, thus any other repetition of the loop would not produce any further conclusion, thus the exit condition in line 25. All conditions to be verified and the operation to be performed in the algorithm requires either constant time or in the worst case linear time. Thus the complexity of the algorithm is dominated by the number of times we have to repeat the main loop; but this is bounded by the number of rules. To have a successful cycle we either had to remove a rule (making the set of rule for a literal \( l \) empty) or reduced a rule to a rule with empty body (and after that we could remove the rule\(^2\)). In the worst case we remove a single rule at every cycle and at every cycle we have to scan the Modal Herbrand Base. In addition the set of rules and the Modal Herbrand Base are finite, thus the algorithm terminates and the complexity of the main loop is \( O(|HB^D| * |R_s|) \).

The discussion of Algorithm 5.1 above shows that

**Proposition 5.11.** The definite extension of a Defeasible Modal Theory \( D \) can be computed in time linear to the size of the theory, where the size of the theory is determined by the number of literals and rules in it.

Before moving to the algorithm to compute the defeasible extension of a Defeasible Modal Theory we introduce an auxiliary procedure that is used several time in the computation of the defeasible extension. The procedure in Algorithm 5.2 do the ‘housekeeping’ operations related to when we prove \(-\partial_\Box l\). In nature it is very similar to the operations in lines 9–12 of Algorithm 5.1, the difference is that it operates on all types of rules and not only on strict rules. In addition

\(^2\)In the algorithm this effect is achieved by remove the literal for the Modal Herbrand Base, thus while the rule is not remove, effectively is removed, since we do not scan the literal
when it removes a rule it also removes the rule from the superiority relation. Furthermore, when 
\( \Box = \text{BEL} \) it operates on \( \otimes \)-expression in the head of rules. For these expressions it truncates
the \( \otimes \)-expression keeping only the part of the expression preceding the first occurrence of the
complement of the literal the procedure takes as parameter (line 9). This operation implement
clause (2.3) of \( +\partial_\Box \) and the or part of clauses (2.2) and (2.3.1) of \( -\partial_\Box \).

Algorithm 5.2: Discard

Algorithm: Discard \((l, \Box)\)

Data: \( D = (F, R, >) \): a defeasible theory

Data: \( l \): a literal in \( D \)

Data: \( \Box \): a modal operator

1. \( \partial_\Box \leftarrow \partial_\Box \cup \{l\} \)
2. \( R \leftarrow R - \{s \in R : \Box l \in A(r)\} \)
3. \( R \leftarrow \{A(r) - \{\neg \Box l\} \rightarrow_X C(r) : r \in R\} \)
4. \( R^{\Box, \Box}_i \leftarrow R^{\Box, \Box}_i - \{s \in R^{\Box, \Box}_i : l \in A(r)\} \)
5. \( >\leftrightarrow -\{(r, s), (s, r) \in >; \Box l \in A(r)\} \)
6. if \( \Box = \text{BEL} \) then
7. \( R \leftarrow R - \{s : l \in A(s)\} \)
8. \( R \leftarrow \{A(r) \rightarrow_X C(r)!\sim l : r \in R\} \)
9. \( >\leftrightarrow \{(r, s), (s, r) \in >; l \in A(r)\} \)

We are now ready to present the Algorithm (Algorithm 5.3) to compute the defeasible
extension of a Defeasible Modal Theory, i.e., the set of conclusions that can be proved with \( \partial_\Box \).

Most of the aspects of Algorithm 5.3 are essentially identical to the corresponding operation
and condition in Algorithm 5.1. Thus the focus here is just on the aspect specific to the task
at hand.

For the initialisation we take the output of \textit{ComputeDefinite} Algorithm and we populate
the global sets of positive and negative defeasible conclusions according to clause (1) of \( +\partial + \Box \)
and clause (2.1) of \( -\partial_\Box \). We then remove all literal included in the two sets just initialised from
the Modal Herbrand Base. We also generate clones of the rules for the set of rules that can be
applied in a conversion. The last initialisation step is to create a set where to store the rules
that are weaker than applicable rules (line 5).

As in \textit{ComputeDefinite} we have a main loop, where, for each cycle we initialise the set of
conclusions that can be computed at that iteration of the main loop. Again the main loop has
two mutually exclusive blocks. The first block (lines 10–13) corresponds to the similar block in
\textit{ComputeDefinite}. The difference here is that we operate on all types of rules and not only on
strict rules, but the operations performed are the same.

The real differences are in the block at lines 15–34. The starting point is to figure out if there
are applicable rules (of an appropriate mode). The rules we have to consider are the rule for a
modal operator plus all rules that can be used in a conversion to that modal operator. In this
case, since defeasible rules can have \( \otimes \)-expressions as they heads, the literal we are interested to
is the first element of the \( \otimes \)-expression. For all applicable rules we collect the rules weaker than
Algorithm 5.3: ComputeDefeasible

**Algorithm:** ComputeDefeasible

**Data:** \( \mathcal{D} = (F, R, >) \): a defeasible theory

**Data:** \( +\Delta_\square \): set of definitely provable conclusions

**Data:** \( -\Delta_\square \): set of definitely not provable conclusions

**Result:** \( +\partial_\square \) - set of defeasibly provable conclusions

**Result:** \( -\partial_\square \) - set of defeasibly not provable conclusions

\[
+\partial_\square \leftarrow +\Delta_\square \\
-\partial_\square \leftarrow \{ l : \sim l \in -\Delta_\square \} \\
R^{\square,\square} \leftarrow \{ r \in R^{\square} : \text{Convert}(\square_i, \square_j), A(r) \neq \emptyset, A(r) \text{ is a set of literals} \} \\
HB^{\square} \leftarrow HB^{\square} - \{ \square l : l \in +\partial_\square \cup -\partial_\square \} \\
R_{inf_d} \leftarrow \emptyset \\
\text{repeat} \\
\quad \text{for } \square l \in HB^{\square} \text{ do} \\
\quad \quad \text{if } R_{sd}^{\square}[l] \cup R_{sd}^{\square}[\sim l] = \emptyset \text{ then} \\
\quad \quad \quad \text{DISCARD}(l, \square) \\
\quad \quad \quad HB^{\square} \leftarrow HB^{\square} - \{ \square l \} \\
\quad \text{for } r \in R^{\square} \cup R^{\square_i, \square_j} \text{ do} \\
\quad \quad \text{if } A(r) = 0 \text{ then} \\
\quad \quad \quad R_{inf_d} \leftarrow R_{inf_d} \cup r_{inf} \\
\quad \quad \text{Let } l \text{ be the first literal of } C(r) \\
\quad \quad \text{if } R^{\square_i}[\sim l] \cup R^{\square_i}[\sim \sim l] - R_{inf_d} \subseteq r_{inf}, \text{ for } \square_i \text{ s.t. Conflict}(\square_i, \square) \text{ then} \\
\quad \quad \quad \text{DISCARD}(\sim l, \square_i) \text{ for } \square_i \text{ s.t. Conflict}(\square_i, \square_i) \\
\quad \quad \quad HB^{\square} \leftarrow HB^{\square} - \{ \square l : \text{Conflict}(\square_i, \square_i) \} \\
\quad \quad \text{if } r \in R_{sd} \text{ then} \\
\quad \quad \quad \partial_\square^{+} \leftarrow \partial_\square^{+} \cup \{ l \} \\
\quad \quad \quad R^{\square,\square} \leftarrow \{ A(s) - \{ l \} \rightarrow_X C(s) : s \in R \} \\
\quad \quad \quad R^{\square_i, \square} \leftarrow \{ A(s) - \{ l \} \rightarrow_X C(s) : s \in R^{\square_i, \square_j} \} \\
\quad \quad \text{if } \square = \text{BEL} \text{ then} \\
\quad \quad \quad R \leftarrow \{ A(s) - \{ l \} \rightarrow_X C(r) \Theta \sim l : s \in R \} \\
\quad \quad \quad R^{\square_i, \square} \leftarrow \{ A(s) \rightarrow_X C(r) \Theta \sim l : s \in R^{\square_i, \square_j} \} \\
\quad \quad \quad HB^{\square} \leftarrow HB^{\square} - \{ \square l \} \\
\quad \text{until } \partial_\square^{+} = \emptyset \text{ and } \partial_\square^{-} = \emptyset 
\]
them and we group them in the set $R_{inf}$. This set contains all rules for which clause (3.2.1) of $+\varnothing$ holds.

The next step, line 19, is to check whether there are rules for the complement of the literal of the same mode, or of a mode conflicting with the mode we want to prove the literal with. The rules for the complement should not be defeated by an applicable rule, i.e., they should not be in $R_{inf}$. When the condition in line 19 is satisfied, then clauses (2.2) and (2.3) of $-\varnothing$ are satisfied, and thus can assert that the complement of the literal is rejected with mode $\Box$, thus we have $-\varnothing \sim l$, and we call the Discard procedure to do the required housekeeping.

After that, what we have to do is to check if the rule is a rule that can support a positive conclusion, i.e., not a defeater. This is the final step to verify that the condition for deriving $+\varnothing$ is satisfied. The last thing to do is to do the housekeeping to reduce the complexity of the theory. Thus we remove all instances of $\Box l$ from the body of rules, $l$ form the bodies of the appropriate rules for conversion as we did for the case of ComputeDefinite. However, we have to do an additional operation. In case $\Box = \text{BEL}$ we have to remove the literal from $\otimes$-expressions occurring in the head of rules, to ensure that we properly capture the second conjunct of clause (2.2) of $+\varnothing$, the third of clause (2.3.1) of the same condition, and the second conjunct of clause (2.3) of $-\varnothing$.

The termination analysis is the same as that of ComputeDefinite. For the complexity, in this case the complexity of main loop is dominated by the for loop in lines 15–34. For this we have to repeat the loop one time for each literal in $HB^\Box$, hence the complexity of this loop is again $O(|HB^\Box| \ast |R|)$.

From the discussion about Algorithm 5.3 above we can conclude the following proposition.

**Proposition 5.12.** The defeasible extension of a Defeasible Modal Theory $D$ can be computed in time linear to the size of the theory, where the size of the theory is determined by the number of literals and rules in the theory.

### 5.4 The framework

In this section, we give a formal definition of an MCS which handle Distributed Defeasible Speculative Reasoning (DDSR) with multi-agent belief revision using conventional concepts and notation from logic programming. It is based on the argumentation semantics of DL presented in Governatori et al. [101] with the notion of askable literals and preference among system contexts.

**Definition 5.13.** An MCS $P = \{P_1, \ldots, P_n\}$ is a set of peers where each peer $P_i \in P$ is defined as a tuple $P_i = (id, grp_i, C_i, \Psi_i, T_i)$, where:

- $id$ is a symbol that identifies the agent $P_i$ within the context, called the agent identifier of $P_i$,
- $grp_i$ is the context groups that $P_i$ associated with in $P$,
• \( C_i = (V_i, R_i) \) is the context (local) defeasible theory (or the knowledge base) in \( P_i \) where \( V_i \) is the vocabulary used by \( P_i \) and \( R_i \) is the set of rules defined in \( C_i \),

• \( \Psi_i \) is the set of default hypotheses assumed by \( P_i \),

• \( T_i \) is a reputation score table on \( P \).

\( V_i \) is a finite set of positive and negative (modalised) literals. A literal is either of the form \( p(t) \) (called local or non-askable literal) or \( p(t)@S \) (called askable or foreign literal in belief), where \( p \) is its predicate (name), \( t \) is a shorthand for its arguments as a vector of terms \( t_1, \ldots, t_n (n \geq 0) \) and \( S \) is an context group identifier. We assume that each peer uses a distinct vocabulary.\(^3\)

\( R_i \) is a set of rules of the form:

\[ r^i_{\bullet}: a^i_1, a^i_2, \ldots, a^i_n \hookrightarrow a^i_{n+1} \otimes \cdots \otimes a^i_{n+m} \]

where \( \hookrightarrow = \{ \rightarrow, \Rightarrow \} \), each of \( a^i_{n+1}, \ldots, a^i_{n+m} \) is a local literal, and each of \( a^i_1, \ldots, a^i_n \) is a literal.

Rules that do not contain any askable literals in their bodies are called local rules and are interpreted in the classical sense. That is, whenever the literals of the body of the rule are consequences of the local theory, so is the literal in the head of the rule. Local rules with empty body denote factual knowledge.

Rules that contain askable literals in their bodies are called mapping rules and are used to express uncertainty that may appear in the environment. They are used to associate local literals with literals from external contexts (foreign literals). An askable literal \( q@S \) in a mapping rule has two functions:

• It represents a question/query to agents in the context of group \( S \);

• An askable literal in \( \Psi_i \) represents a belief of truth value of the literal.
  
  – If \( p(t)@S \in \Psi_i \), the agent assumes that \( p(t)@S \) is true or defeasibly provable.
  
  – If \( \neg p(t)@S \in \Psi_i \), the agent assumes that \( p(t)@S \) is false or not defeasibly provable.
  
  – Otherwise, the agent will assume that \( p(t)@S \) does not exist and will continue the inference process using the underlying semantics.

\( \Psi_i \) is the default hypotheses of assumed by \( P_i \), which can be obtained based on the \( P_i \)’s partial knowledge or as specified by the agent’s users, and can be considered as a heuristics used for prioritizing parallel computation by the agent, which may have impact on the performance of the speculative computation [147].

Finally, each peer \( P_i \) defines a reputation score table \( T_i = [\rho_j, \rho_k, \ldots, \rho_n] \) on \( P \) and each \( \rho_j \) is a name-value pairs \( (C_m, v) \) where \( C_m (m \neq i) \) and \( v \in [0,1] \) are the context that appear in \( P \) and its reputation score w.r.t. \( C_i \), denoted \( \text{Rep}_{C_i}(C_m) \), respectively. It is a quantitative measure of agents’ reputation in the environment and can be used to express the confidence

\(^3\)However, it is possible to extend this framework to support sharing (part-) of vocabulary between agents.
in the knowledge that an agent import from other agents. Besides, it also provides a way to resolve potential conflicts that may arise from the interaction of contexts through their mapping rules. Accordingly, for two agents $P_j, P_k \in P$, $P_j$ is preferred by $P_k$ if $v_j > v_k$, and are equally trustable if $v_j = v_k$.

**Definition 5.14 (Query message).** A query is a triple $Q = (AID, GS, hist)$ where

- $AID$ is the agent identifier,
- $GS$ is a finite set of askable literals,
- $hist$ is a sequence of askable literals.

Given a query, the ambient agents collaboratively compute the answers using their respectively knowledge and assumptions to the environment. Each query contains a finite set of askable literals $GS$ to be proven (the goal set) and a history $hist$ which is initialized to the empty sequence. A history is a sequence of askable literals $[l_n, l_{n-1}, \ldots, l_0]$ represents a branch of reasoning initiated by the goal set of the initial query. It is used to avoid asking redundant queries to other agents. $AID$ is the agent identifier of the querying agent and is used to return the results back to the agent after computations.

**Definition 5.15 (Reply message).** A message reply from an agent for an askable literal $L@CG$ is defined as $Msg = (Sender, Receiver, L@CG, CT)$, where

- $Sender$ and $Receiver$ are the unique identifiers of the agents that send and receive the message respectively,
- $L$ is a literal, and
- $CT$ is a conclusion tag, and
- $CG$ is the context group that the literal belongs to.

where $CT \in \{+\partial, -\partial\}$ as defined in DL, or undefined if the agent $Sender$ has no information about $L$ and cycle if the literals that $L$ relies on depending on each others and form a cycle.

**Definition 5.16 (Message update).** Let $CurrRep = \{Msg_1, \ldots, Msg_n\}$ be the set of replies received by an agent and $Msg = (Sender, Receiver, L@CG, CT)$ be a message sent from an agent. Then a message update in $CurrRep$ is defined as:

$$CurrRep \setminus \{(Sender, Receiver, L@CG, CT)\} \cup \{(Sender, Receiver, L@CG, CT)\}$$

where $+\partial = -\partial$ and $-\partial = +\partial$.

The above definition specifies the message update process that appear in the agents. Here, $CurrRep$ contains the set of most updated replies received from external contexts and will distinguish them according to the context identifier and the context group of the foreign literals. It is used to replace an outdated contextual information with an updated one sent from the same agent.
**Definition 5.17** (Argument score). Let $\text{Msg} = (\text{aid}_j, \text{aid}_i, L @ CG, CT)$ be a message reply send from an ambient agent $P_j$ to another agent $P_i$. The argument score of the literal $L$ w.r.t. context $C_i$, $\text{Rep}_{C_i}(L, C_j)$, is equal to the value of $\text{Rep}_{C_i}(C_j)$.

The above definition states that, given a message received by an agent $P_i$, the argument score of a foreign literal is equals to the reputation score of the context (in $T_i$) that sent the message.

**Example 5.1** (Context-aware mobile phone). The framework described above is applied as follows to the context-aware mobile phone described in Section 5.1. The local knowledge of Dr. Amber’s mobile phone (denoted as $C_1$), together with other context-related information, is encoded as the following rules:

- $r_{1,1}$: $\text{normal}_1 \Rightarrow \text{OBL allow\_ring}_1 \And \text{allow\_vibrate}_1$  
- $r_{1,2}$: $\text{silence\_mode}_1 \Rightarrow \text{OBL allow\_ring}_1$  
- $r_{1,3}$: $\text{INT alert\_user}_1, \text{OBL allow\_ring}_1 \Rightarrow \text{ring}_1$  
- $r_{1,4}$: $\text{INT alert\_user}_1, \text{OBL allow\_vibrate}_1 \Rightarrow \text{vibrate}_1$  
- $r_{1,5}$: $\text{incoming\_call}_1, \text{BEL} \neg \text{lecture}_1 \Rightarrow \text{INT alert\_user}_1$  
- $r_{1,6}$: $\Rightarrow \text{normal}_1$  
- $r_{1,7}$: $\Rightarrow \text{incoming\_call}_1$  

$r_{1,2} > r_{1,1}$

The above rules describes the propositional concerns of Dr. Amber’s mobile phone, which includes the operation mode and the incoming calls. It is assumed that, when an incoming call arrive, Dr. Amber’s mobile should ring if it is operated under normal conditions, or vibrate if it is operated under silence mode ($r_{1,1}$ and $r_{1,2}$).

In case the mobile phone cannot reach a decision based on its local knowledge, it starts importing knowledge from other ambient agents. The following mapping rules associates the local knowledge of the mobile phone with the context knowledge of Dr. Amber’s laptop ($C_2$), the localization service ($C_3$) and the classroom manager ($C_4$).

- $r_{1,8}$: $\text{BEL classtime}_1 @ \text{LAPTOP}, \text{location\_RA201}_1 @ \text{REGISTRY} \Rightarrow \text{BEL lecture}_1$  
- $r_{1,9}$: $\text{BEL} \neg \text{class\_activity}_1 @ \text{CALSSROOM\_MANAGER} \Rightarrow \text{BEL} \neg \text{lecture}_1$  

$r_{1,9} > r_{1,8}$

The local context knowledge of the laptop ($C_2$), the localization service ($C_3$), the classroom manager ($C_4$) and the classroom sensors ($C_5$) is encoded in rules $r_{2,1}, r_{3,1}, r_{4,1} - r_{4,3}$ and $r_{5,1}$ respectively. To import the knowledge from the classroom sensor, the classroom manager uses rule $r_{4,4}$.  

125
The rules above state that the classroom manager will determine whether there is class activity in the classroom based on the number of people in the classroom and the current condition of the projector. By default, the classroom manager assumes that there is class activity in the classroom (rule $r_{4,2}$). However, if there is no or only one person in the classroom and the projector is turned off, this default will be overruled by either rules $r_{4,3}$ or $r_{4,4}$, and the classroom manager will believe that there is no classroom activity in the classroom. In the scenario, since Dr. Amber is the only person in the classroom, checking his email and with the projector turned off, the classroom manager will infer that there is no class activity taking place at this moment.

Note however that this condition may change if, at the same time, another classroom sensor ($C_6$) with higher reputation score than $C_5$ (in the reputation score table of $C_4$ ($T_4$)) saying that many people have been detected in the classroom. Then, under this situation, the classroom manager will conclude that Dr. Amber is having class activity and will not alert him about the incoming calls.

5.5 Semantics of DDSR

Similar to the construct in [31], the distributed defeasible speculative reasoning semantics that we propose uses arguments of local range, in which conclusions derived are from a single context. Arguments made by external contexts are, at least conceptually, linked by bridges (Definition 5.18) through mapping rules. It is intended to provide an argumentative characterization of Defeasible Logic with notions of distributed information and preference among system contexts.

5.5.1 Preliminary Definitions

**Definition 5.18.** Let $P = \{P_1, \ldots, P_n\}$ be an MCS and each peer $P_i = (id, grp_i, C_i, \Psi_i, T_i)$ (Section 5.4). The Bridges ($\Phi_P$) of $P$ is the set of tuples of the form $(p_i, C_i, \Psi_i, PT_{p_i})$, where $p_i \in V_i$ and $PT_{p_i}$ is the proof tree for literal $p_i$ based on a set of local and mapping rules of $C_i$ and the set of default hypotheses $\Psi_i$. 

126
Definition 5.19 (Proof Tree). Let \( P = \{P_1, \ldots, P_n\} \) be an MCS and each peer \( P_i = (id, grp_i, C_i, \Psi_i, T_i) \). A proof tree \( PT_p \) of \( P \) is a tree with nodes labeled by literals such that the root is labeled by \( p \) and for every node \( q \):

- If \( q \in V_i \) and \( a_1, \ldots, a_n \) label the children of \( q \) then
  - If \( \forall a_i \in \{a_1, \ldots, a_n\} : a_i \in V_i \) and the set \( \{a_1, \ldots, a_n\} \) contains no askable literals, then there is a local rule \( r \in C_i \) with body \( a_1, \ldots, a_n \) and head \( q \).
  - If \( \exists a_j \in \{a_1, \ldots, a_n\} \) such that \( a_j \notin V_i \) or is askable, then there is a mapping rule \( r \in C_i \) with body \( a_1, \ldots, a_n, \ldots, a_n \) and head \( q \).

- If \( q \in V_j \neq V_i \) then this is a leaf node of the tree and there is a tuple of the form \( (p_i, C_i, \Psi_i, PT_{p_i}) \) in \( \Phi_P \).

  - the arcs in a proof tree are labeled by the rules used to obtained them.

and \( \forall q \in \Psi_i, q \) is an askable literal in \( V_i \).

Definition 5.20. An argument for a literal \( p_i \) is a tuple \( (p_i, C_i, \Psi_i, PT_{p_i}) \) in \( \Phi_P \).

Definition 5.21. Given an MCS \( P = \{P_1, \ldots, P_n\} \), the set of arguments that can be generated from \( P \) is denoted by \( \text{Args}_P \), while the set of arguments that can be generated from each peer context \( C_i \) is denoted by \( \text{Args}_{C_i} \). Consequently, \( \text{Args}_P = \bigcup \text{Args}_{C_i} \).

Any literal labeling a node of an argument \( A \) of a proof tree \( PT_{p_i} \) is called a conclusion of \( A \). However, when we refer to the conclusion of an argument, we refer to the literal labeling the root of the argument (i.e., \( p_i \)).

Definition 5.22. A (proper) subargument of an argument \( A \) is a (proper) subtree of the proof tree associated to \( A \).

Based on the literals used in the proof tree, an argument can be classified into two different types: local argument and mapping argument.

Definition 5.23. A local argument of context \( C_i \) is an argument with a proof tree that contains only local literal of \( C_i \). If a local argument contains only strict rules, then it is a strict local argument; otherwise it is a defeasible local argument.

Definition 5.24. A mapping argument of context \( C_i \) is an argument with proof tree that contains at least one foreign literal of \( C_i \).

Example 5.2. Consider the following defeasible context theory \( C_1 \):

\[
\begin{align*}
  r_{1,1} : & \quad \Rightarrow a_1 \\
  r_{1,2} : & \quad a_1, a_2 \Rightarrow x_1 \\
  r_{1,3} : & \quad \Rightarrow \neg x_1 \\
  r_{1,4} : & \quad c_1 \Rightarrow x_1 \\
  r_{1,5} : & \quad \Rightarrow c_1 \\
  r_{1,6} : & \quad a_3 \Rightarrow \neg a_1 \\
  r_{1,7} : & \quad a_4 \Rightarrow \neg c_1 \\
  r_{1,2} & > r_{1,3}
\end{align*}
\]
Here $a_1, c_1, x_1$ are local literals of $C_1$, where $a_2, a_3$, and $a_4$ are local literals of $C_2, C_3$ and $C_4$ respectively, which all belong to the same Multi-Context System $P$. Assuming that for $a_2, a_3$ and $a_4$ there are tuples of the form $(a_i, C_i, \Psi_i, PT_{a_i}), i \in \{2, 3, 4\}$, in $Args_P$, $Args_{C_1}$ contains the arguments as shown in Figure 5.9.

The subargument of $A_1$ and $A_3$ with conclusions $a_1$ and $c_1$ respectively are also arguments of $Args_{C_1}$. From the figure, since $A_2$ and $A_3$ contain only local literals, they are local arguments of $C_1$. $A_2$ is a strict local argument since it relies only on a strict rule ($r_{1,3}$), while $A_3$ is a defeasible argument since it contains one defeasible rule in the derivation. On the other hand, $A_1, A_4$ and $A_5$ are all mapping argument of $C_1$ since they both contain foreign literals in $C_1$.

The derivation of local logical consequences of a context, such as $C_1$ in the example above, relies only on their local arguments. Actually, the conclusions of all local arguments are just the same of the logical consequences of $C_i$. However, the derivation of distributed logical consequences is different. It relies on the combination of both the local arguments in $Args_{C_i}$ and mapping arguments in $Args_P$. In this case, we have to consider conflicts that may arise when mapping arguments from external contexts attack each other.

As mentioned before, we can resolve the conflicts among mapping arguments by ranking them according to the agents’ reputations score in $T_i$, and select the one with the highest value. That is, given three conflicting arguments $A_1, A_2, A_3$ received from contexts $C_1, C_2, C_3$ respectively: If $Rep(C_1) < Rep(C_2)$ and $Rep(C_2) < Rep(C_3)$, we should conclude $A_3$ and falsify both $A_1$ and $A_2$. This approach is equivalent to the preference ordering approach employed in [32] such that the context with highest reputation value will dominate the results being concluded.

However, in a collaborative and dynamic environment like what we are proposing, justifying mapping arguments based on the preference ordering may not be good enough. Ambient agents in the environment may not have enough knowledge to justify whether a particular context is having a stronger evidence for the correctness of her answers, nor do they have enough knowledge about the environment. Besides, we also have to consider situations where the same foreign literals may receive positive and negative support by mapping arguments coming from multiple contexts.

Here the role of docility is worth citing. According to Herbert Simon, humans are docile
in the sense that their fitness is enhanced by “[…] the tendency to depend on suggestions, recommendations, persuasion, and information obtained through social channels as a major basis for choice” [201]. In other words, human support their limited decision-making capabilities through receiving inputs, data, perceptions from the social environment [196]. It is the social context that gives human beings the main data filter to increase their individual fitness.

Put it into our context, due to the limited capabilities of an ambient agents, it should enhance its fitness in the ambient environment by learning, or receiving information, from those that surround it. Ambient agents should have their own knowledge and perceptions about the environment, but should also be able to adapt to the norms of the context in such a way that group collaboration between agents can be evolve adaptively. This is where belief revision can be expected to help. It ensures that the information that an ambient agent received from others ages remains applicable and can be constantly updated when more information becomes accessible.

**Definition 5.25** (Argument rank). Let $p_i$ be a foreign literal and $\text{CurrRep} = \{M_s g_1, \ldots, M_s g_n\}$ be the set of most updated message received by an agent $P_i$. The argument rank of $p_i$ w.r.t. the context $C_i$ is equals to the sum of all argument scores $\text{Rep}_{C_i}(p_i, C_i)$ of messages that appear in $\text{CurrRep}$ concerning $p_i$, denoted by $\Sigma_{C_i}(p_i)$.

The definition above states that, the argument rank of a foreign literal is equal to the sum of the reputation scores of the contexts that support it. Here, it is worth mentioning that numerous mechanisms have been proposed in the literature [179, 217, 198, 221] to evaluate agents’ opinion or agents’ trust management under distributed environment. In our framework, we evaluate an askable literal using the sum of agents’ reputation score since we want to leverage individual agent’s reputation as well as the majority of agents’ belief within a particular context.

### 5.5.2 Conflicting Arguments: Attach and Undercut

It is important to note that the definitions of attach and defeat apply only for local defeasible and mapping arguments.

**Definition 5.26.** An argument $A$ attacks a defeasible local or mapping argument $B$ at $p_i$ if $p_i$ is a conclusion of $A$ and $\sim p_i$ is a conclusion of $B$, and the subargument of $B$ with conclusion $p_i$ is not a local argument.

**Definition 5.27.** An argument $A$ defeats a defeasible local or mapping argument $B$ if $A$ attacks $B$ at $p_i$, and for the subargument of $A$, $A'$ with conclusions $p_i$ and the subargument of $B$, $B'$ with conclusion $\sim p_i$, $\Sigma_{C_i}(p_i) \geq \Sigma_{C_i}(\sim p_i)$.

To links arguments to the mapping rules that they contain, we introduce the notion of argumentation line, as follow.

**Definition 5.28.** For a literal $p_i$ in context $C_i$, an argumentation line $\text{AL}_{C_i}(p_i)$ is a sequence of arguments in $\text{Args}_P$, constructed using the following steps:
• In the first step add in $AL_{C_i}(p_i)$ an argument for $p_i$.

• In each next step, for each distinct literal $q_j$ labeling a leaf node of the proof tree of the arguments added in the previous step, add one argument with conclusion $q_j$ from $\Pi_{C_i}(p_i)$ which satisfy the following restriction.

• An argument $B$ with conclusion $q_j$ can be added in $AL_{C_i}(p_i)$ only if $AL_{C_i}(p_i)$ does not already contain a different argument $D$ with conclusion $q_j$.

For an argument $p_i$ in context $C_i$ with argumentation line $AL_{C_i}(p_i)$, $p_i$ is called the head argument of $AL_{C_i}(p_i)$ and it conclusion $p_i$ is also the conclusion of $AL_{C_i}(p_i)$. If the number of steps required to build an argumentation line is finite, then $AL_{C_i}(p_i)$ is also finite. An infinite argumentation line implies that some mapping arguments that $p_i$ relies on in $ArgscC$ depend on each other forming a loop. An argument in an infinite argumentation line can participate in attacks against counter-arguments but may not be used to support the conclusions of their argumentation line.

**Definition 5.29 (Tentative belief state).** Let $CR_i$ be a subset of the askable literals. The tentative belief state of $P_i$ w.r.t. $C_i$ and $\Psi_i$ is a set:

$$\{L|L \in CR_i \text{ and } \sim L \in \Psi_i\} \cup \{L|L \in \Psi_i \text{ and } \sim L \not\in CR_i\}$$

and is denoted $\Pi_{C_i}(CR_i, \Psi_i)$.

In an ambient environment, tentative replies from different agents may frequently arrive at the querying agent. The notion of belief state for an ambient agent provides a mean to (1) substitute mapping arguments which have not yet been confirmed with default hypotheses and start the inferencing process while waiting for the pending replies from external contexts. This is particularly important in our framework as proactive inferencing is the key of success for speculative computation. (2) it helps in handling the possibly conflicting mapping arguments that the agent may receive from multiple external contexts.

Hence, in the definition above, $CR_i$ can be regarded as the current set of belief derived based on the latest replies ($CurrRep$) received from the external contexts. In the case where mapping arguments from different contexts attack each other, it is then possible to justify them according to their argument rank ($\Sigma_{C_i}$). Therefore, for every foreign literals that appear in $C_i$, its tentative belief state, $\Pi_{C_i}(CR_i, \Psi_i)$, will be assigned first according to value available in $CR_i$, or otherwise the value that appear in default hypotheses $\Psi_i$.

**Definition 5.30.** An argument $A$ is supported by a set of arguments $S$ if

• every proper subargument is in $S \cup \Pi_{C_i}(CR_i, \Psi_i)$, and

• there is a finite argumentation line $AL_{C_i}(A)$ with head $A$ such that every arguments in $AL_{C_i}(p_i) \setminus \{A\}$ is in $AL_{C_i}(A)$.
Despite the similarity of name, this concept is not directly related to support in defeasible logic, nor to supportive arguments/proof trees. It is meant to indicate when an argument may have an active role in proving or preventing the derivation of a conclusion.

**Definition 5.31.** An argument $A$ is undercut by a set of arguments $S$ if for every argumentation line $ALC_i(A)$ with head $A$: there is an argument $B$, such that $B$ is supported by $S$, and $B$ defeats a proper argument of $A$ or an argument in $ALC_i(A) \setminus A$.

### 5.5.3 The status of arguments

The heart of an argumentation semantics is the notion of acceptable argument. Based on this concept it is possible to define justified arguments and justified conclusions - conclusions that may be drawn even taking conflicts into account.

**Definition 5.32.** An argument $A$ is acceptable by a set of arguments $S$ if:

1. $A$ is a strict local argument; or

2. (a) $A$ is supported by $S$, and
   (b) every arguments that defeat $A$ in $\text{Args}_P$ is undercut by $S$.

Intuitively, an argument $A$ is acceptable w.r.t. a set of arguments $S$ if, once we accept $S$ as valid arguments, we feel compelled to accept $A$ as valid.

Based on this concept we proceed to define justified arguments and justified literals.

**Definition 5.33.** Let $P$ be a multi-context system. We define $J^P_i$ as follows.

- $J^P_0 = \emptyset$;
- $J^P_{i+1} = \{a \in \text{Args}_P \mid a$ is acceptable w.r.t. $J^P_i\}$.

The set of justified arguments in an MCS $P$ is $\text{JArgs}^P = \bigcup_{i=1}^{\infty} J^P_i$. A literal $p$ is justified in $P$ if it is the conclusion of a supportive argument in $\text{JArgs}^P$. That is, an argument $A$ is justified means that it resists every reasonable refutation; while a literal is justified if it is a logical consequence of $P$, from the perspective of $P_i$.

Finally, we proceed to define the notion of rejected arguments and rejected literals for the characterization of conclusions that are not derivable in $P$. Roughly speaking, an argument is rejected by a sets of arguments $S$ and $T$ (defined below) if it has a rejected subargument or it cannot overcome an attack from another argument, which can be thought of as the set of justified arguments from $P$.

**Definition 5.34.** An argument $A$ is rejected by a sets of argument $S, T$ if:

1. $A$ is not a local strict argument;
2. (a) a proper subargument of $A$ is in $T$, or
(b) $A$ is defeated by an argument supported by $S$, or
(c) for every argumentation line $AL_P(A)$ with head $A$ there exists an argument $A' \in AL_P(A) \setminus A$ s.t. either a subargument of $A'$ is in $S$; or $A'$ is defeated by an argument supported by $T$,

where $S$, $T$ is the set of arguments supported and rejected by $P$ respectively.

Based on this we then proceed to define rejected arguments and rejected literals.

**Definition 5.35.** Let $P$ be a multi-context system and $JArgs^P$ be the set of justified arguments in $P$. We define $R_i^P$ as follows.

- $R_0^P = \emptyset$;
- $R_{i+1}^P = \{a \in Args_P \mid a$ is rejected by $R_i^P, JArgs^P\}$.

The set of rejected arguments in an MCS $P$ is $RArgs^P = \bigcup_{i=1}^{\infty} R_i^P$. A literal $p$ is rejected in $P$ if there is no argument in $Args_P \setminus RArgs^P$ with conclusion $p$. That a literal is rejected means that we are able prove that it is not a logical consequence of $P$, from the perspective of $P_i$.

### 5.5.4 Properties of the framework

Lemma 5.36~5.38 below describe some properties of the framework. Their proofs are presented in Appendix B.

Lemma 5.36 refers to the monotonicity of the justified arguments ($J_i^P$) and rejected arguments ($R_i^P$), while Lemma 5.37 states that no argument is both justified and rejected.

**Lemma 5.36** (Monotonicity). The sequences of sets of arguments $J_i^P$ and $R_i^P$ are monotonically increasing.

**Lemma 5.37** (Coherence). In a Defeasible Multi-Context System $P$:

- No argument is both justified and rejected.
- No literal is both justified and rejected.

If consistency is assumed in the local context theory, then using the previous lemma, it is possible to prove that the entire framework is consistent, as described in the lemma below.

**Lemma 5.38.** If the set of justified arguments in $P$, $JArgs^P$, contains two arguments with conflicting conclusions, then both arguments are strict local arguments.
5.6 An Operational Model with Iterative Answer Revision

Proactive inferencing is the key of success for speculative computation. Given an external query, an ambient agent will compute all answers with a local inference procedure, which is based on two phases: a process reduction phase and an answer arrival phase [147, 194]. The former is a normal inference process executed within an ambient agent and is basically the iff reduction proposed in [135]. It is a process of inferencing the set of conclusions to the query, which are abducible, based on agent’s current (tentative) belief state (Definition 5.29), and new queries are sent out to other ambient agents (according to their agent groups) when askable literals are reduced. After completing the inference process, the (tentative but possibly final) conclusions derived will then return to the querying agent for further processing.

Whenever an answer (either new or revised) arrives, the process reduction phase is interrupted and the answer arrival phase takes over to revise the current computation accordingly. Instead of discarding any completed but outdated computation, i.e., conclusions that are derived using the default hypotheses or an old answer being revised, the revision process is designed based on the principle of reusing it as much as possible [147], which is the major novelty of speculative reasoning. The algorithms present in the following sections are based on a demand-driven approach and a top-down procedure is employed for speculative computation.

5.6.1 Preliminary Definitions

**Definition 5.39** (Speculative process). A process is a tuple \( P = (\text{pid}, \text{ps}, \text{GS}, \Pi, \Theta) \) where

- **pid** is the unique process identifier,
- **ps** is state of this process,
- **GS** is a set of (askable or non-askable) literals to be proven, or called the goal set of the process,
- **\( \Pi \)** is the set of foreign literals, corresponding to the tentative belief state of the agent \( \Pi_{C_i}(CR_i, \Psi_i) \) as defined in Definition 5.29, and
- **\( \Theta \)** is a set of conclusions derived w.r.t. \( \Pi \).

Each process in an agent represents an alternative way of computation w.r.t. the set of belief state \( \Pi \). It is created when a new choice point is encountered, such as case splitting and new/revised answers arrival. There are two kinds of process (ps): active and suspended. A process is active when its belief state is consistent with the set of agent’s current belief state; while a suspended process is a process using a belief state which is (partially) contradictory with the current belief state.

---

4In [194] the authors are using the term fact arrival phase instead of answer arrival phase.
5Note that the execution can be done in each ambient agent independently of other agents.
Definition 5.40. In each ambient agent \( P_i \in P \), we have:

- A current belief state \( \Pi_{C_i}(CR_i, \Psi_i) \), which contains a subset of askable literals in \( C_i \) w.r.t. the set of beliefs \( CR_i \) derived based on the latest replies (\( CurrRep \)) received from other ambient agents and the default hypotheses \( \Psi_i \).

To facilitate our discussions on process reduction phase, we have the following definition.

Definition 5.41.

- A set of already asked queries \( AAQ \) is a set of queries that have been sent by the agent.
- A set of already sent answers \( ASA \) is a set of askable literals and their conclusions.
- An active process set \( APS \) is a set of active processes.
- A suspended process set \( SPS \) is a set of suspended processes.
- The initial query \( IQ \) with a single askable literal \( p@S \) in the \( GS \).

\( AAQ \) is used to avoid asking redundant questions to other agents. \( ASA \) is used to accumulate the set of previously computed answers. It is used to avoid sending redundant same answers to querying agents and calculating the same answer redundantly when the same questions have already asked by other agents. \( APS \) and \( SPS \) is used to store the set of active and suspended processes respectively. All \( ASA \), \( APS \) and \( SPS \) are initialized to empty set when an ambient agent starts its operations.

### 5.6.2 Process Reduction Phase

The inference procedure is best described as a *state rewriting* process w.r.t. the agent’s context theory and its perception about the current situation of the environment, i.e., the agent’s current belief state, \( \Pi_{C_i}(CR_i, \Psi_i) \). It is triggered by the reception of a *query* message \( Q = (AID, GS, hist) \) sent by an agent \( AID \) to agents in the context group, as defined in the \( GS \), which executes the procedure: on the demand of \( Sender \), with which it shares the literals that appear in the \( GS \), it process the literals specified in \( GS \) and return the conclusion of each literal \( L \) to \( Sender \) with one of the values: (1) \( +\partial \); indicates that \( L \) is justified in the local context; (2) \( -\partial \); indicates that \( L \) is rejected locally; (3) undefined; indicates the queried agent have no information about \( L \); and (4) cycle; indicates that \( L \) appears in a cycle under the current environment and cannot be concluded.

In the algorithm we let the local ambient agent be \( P_i \) with agent identifier “Self” and its associated context group be \( S \); and \( Sender \) be the agent identifier of the agent who issues the query.

The inference procedure proceeds in two steps. In the first step, when a new query arrives, *NewQueryArrival* (Algorithm 5.4) initializes the procedure and determines if any conclusions that match the goal set (\( GS \)) appear in the set of previously computed answers. If conclusions
are found, then the agent will reply to the Sender with the conclusions previously computed. Otherwise, a new process will be created according to the GS and the agent’s current belief state $\Pi_C(CR_i, \Psi_i)$.

### Algorithm 5.4: Process reduction phase: New query arrival

**Algorithm:** $\text{NewQueryArrival}(\Pi_i, Q)$

**Data:**
- $\Pi_i$: current belief state of the agent corresponding to $\Pi_C(CR_i, \Psi_i)$
- $Q = (\text{Sender}, \text{GS}, \text{hist})$: query received from agent Sender

1. if $\text{GS} = \emptyset$ then
2. sendReply message($\text{Self}, \text{Sender}, \emptyset, \emptyset$)
3. else if there is a conclusion substitution $\Theta$ s.t. $\text{GS} \cdot \Theta \in \text{ASA}$ then
4. sendReply message($\text{Self}, \text{Sender}, \text{GS}, \text{GS} \cdot \Theta$)
5. else
6. $\text{APS} \leftarrow \text{APS} \cup \{(\text{pid, active, GS, } \Pi_i, \epsilon)\}$

*aGS $\cdot \Theta$ is the result of applying assignments of $\Theta$ to $\{\text{GS, GS}\}$ where $\overline{\text{GS}}$ is the point-wise complement of each literals in $\text{GS}$.

*b$\epsilon$ is an empty substitution.

In the second step, $\text{ProcessReduction}$ (Algorithm 5.5) will iterate on the set of active processes that are consistent with the current agent belief state. If a process is found and the required literals appear in the conclusions set, the conclusions will then be retrieved from the process and return it to the Sender directly. (Lines 1 to 3).

Otherwise, the procedure will select a process that is consistent with the agent’s current belief set and continues the reduction process iteratively (Lines 5 to 23). For a non-askable literal $L$, $\text{ProcessReduction}$ proceeds by determining whether $L$ or its negation $\overline{L}$ are consequences of the local rules (Lines 10 to 15). If it is the case, the process continues by adding the set of body literals for rules with conclusion $L$ to the GS. Otherwise a reply indicating that the literal $L$ is undefined in the local context will be sent to the agent Sender.

Note that, by the answer arrival phase defined below, $\Pi_C(CR_i, \Psi_i)$ is always consistent. The condition defined in line 17 is used to handle cases when cyclic literals dependencies occur in the external context. That is, when foreign literals in the contexts depend on each other and form a loop, no further query for the same foreign literal (in the same query) will be sent. Instead, the default hypotheses will be used as the ambient agent, in that situation, does not have the ability to determine the true value of the foreign literals under the current environment, which follows the idea used in handling literal-dependencies in the well-founded semantics. However, instead of falsifying the literals using the failure-by-looping strategy, here we will use the default hypotheses as the agent’s belief state in the computations. Otherwise, the agent will issue a query to agents in context group $S'$, as is indicated by the literal (Lines 19 to 22).

### 5.6.3 Answer Arrival Phase

The answer arrival phase is triggered by the reception of a reply message $Q = (\text{Sender}, \text{Receiver}, L@S, \text{CT})$ sent by the peer Sender to peer Receiver which executes the pro-
Algorithm 5.5: Process reduction phase: Iterative step

Algorithm: ProcessReduction($C_i, \Pi_i, Q$)

Data: $C_i = (V_i, R_i)$: local context theory

Data: $\Pi_i$: current belief state of the agent corresponding to $\Pi_C(CR_i, \Psi_i)$

Data: $Q = (Sender, GS, hist)$: query received from agent $Sender$

1. if $\exists P = (\_active, \_\Pi_i, \Theta) \in APA$ s.t. $L \cdot \Theta \notin ASA$ then
   2. $ASA \leftarrow ASA \cup \{L \cdot \Theta\}$
   3. sendReply message($Self, Sender, L, L \cdot \Theta$)
else
5. select an active process $P = (\_active, \_\Pi, \Theta)$ from $APS$
6. $APS' = APS \setminus \{P\}$
7. Select a literal $L$ from $GS$
8. $GS' = GS \setminus \{L\}$
9. $AL = \text{the set of askable literals in } GS$
10. if $L$ is a non-askable literal then
11.     if $R_i[L] = \text{null}$ and $R_i[\overline{L}] = \text{null}$ then
12.         sendReply message($Self, sender, L, \text{undefined}$)
13.     else
14.         $APS \leftarrow APS' \cup \{(\text{pid}_{new}, \_active, (\text{body}(R) \cup GS')\theta, \Pi_i, \Theta \circ \theta) \mid \exists R \in R_i$
15.         and $\exists$ most general unifier (mgr) $\theta$ s.t. $\text{head}(R)\theta = \{L, \overline{L}\}\theta$\(^a\)
16.     else
17.         /* $L$ is an askable literal */
18.         if $L \in \text{hist}$ then
19.             sendReply message($Self, Sender, L, \text{cycle}$)
20.         else if $L \notin AAQ$ and $\overline{L} \notin AAQ$ then
21.             $Q = message(Self, L, \text{hist} \cup \{AL\})$
22.             sendQuery $Q$
23.             $AAQ \leftarrow AAQ \cup Q$
24.             $APS \leftarrow APS' \cup \{(\text{pid}_{new}, \_active, GS', \Pi_i, \Theta)\}$

\(^a\theta_\Theta\) is an assignment for variable in the query and $\circ$ is a composition operator of assignments.
procedure: it processes the conclusion $CT$ sent back by $Sender$ for the literal $L$, updates its belief state of the environment w.r.t. $\Pi_{i}(CR_{i}, \Psi_{i})$ and consequently adapt its behavior according to the conclusions derived based on its local theory.

**Algorithm 5.6: Answer Arrival Phase**

Algorithm: $AnswerArrival(CurrRep, \Pi_{i}(CR_{i}, \Psi_{i}), Msg)$

**Data:**
- $CurrRep$: the set of updated reply messages received from external agents
- $\Pi_{i}(CR_{i}, \Psi_{i})$: the current belief state of the agent
- $Msg = (\text{Sender, Self, } Q\mathcal{CG}, CT)$: the message received from an external agent

1. update $CurrRep$ w.r.t. the $Msg$ received
2. update $CR_{i}$ w.r.t. $CurrRep$
3. $\Pi = \Pi_{i}(CR_{i}, \Psi_{i})$
4. update $\Pi_{i}(CR_{i}, \Psi_{i})$ w.r.t. $CR_{i}$ and $\Psi_{i}$
5. if $Q \in \Pi_{i}(CR_{i}, \Psi_{i})$ and $\sim Q \notin \Pi$ then
   - $L = Q$
7. else if $\sim Q \in \Pi_{i}(CR_{i}, \Psi_{i})$ and $Q \notin \Pi$ then
   - $L = \sim Q$
9. else
   - $L = \text{null}$
11. if $L \neq \text{null}$ then /* execute only when a change of literal value appears */
   12. $\Xi = \{(\sim, \text{active}, \sim, \Pi, \sim) \in APS \mid L \in \Pi\}$
   13. $\Lambda = \{(\sim, \text{suspended}, \sim, \Pi, \sim) \in SPS \mid L \in \Pi\}$
   14. if $\Lambda = \text{null}$ then /* create a new process if no process in the suspended set are consistent with $\Pi_{i}(CR_{i}, \Psi_{i})$ */
   15. $\Lambda = \{(\text{pid}_\text{new}, \text{active}, \text{body}(R) \cup GS')\theta, \Pi_{i}(CR_{i}, \Psi_{i}), \Theta \circ \theta) \mid \exists R \in R_{i}$
   16. and $\exists$ most general unifier (mgr) $\theta$ s.t. $\text{head}(R)\theta = \{L, \sim L\}\theta$
   17. $APS \leftarrow APS \setminus \Xi$
   18. $SPS \leftarrow SPS \setminus \Lambda$
   19. change all process state of processes in $\Xi$ to “suspended” and all process state of processes in $\Lambda$ to “active”.
   21. $APS \leftarrow APS \cup \Lambda$
   22. $SPS \leftarrow SPS \cup \Xi$

$AnswerArrival$ (Algorithm 5.6) ensures that if a returned answer confirms the agent’s current belief, then the computation continue (Lines 5 to 10). If, on the other hand, the revised answer contradicts the agent’s current belief, then processes that are inconsistent with the current belief will be set to the suspended mode and will temporary removed from the active process set; while processes that are consistent with the current belief will be set to the active mode and will be added to the active process set for further process (Lines 11 to 22).

### 5.6.4 Properties of the Inference Procedure

Below, we describe the properties of the inference procedure regarding its termination, soundness and completeness w.r.t. the argumentation framework and complexity. Their proofs are
Proposition 5.42 refers to the termination of inference procedure and is a consequence of the cycle detection within the process reduction phase.

**Proposition 5.42 (Termination).** For a multi-context system $P = \{P_1, \ldots, P_n\}$ where each $P_i = (id, grp_i, C_i, \Delta_i, T_i)$ is a peer in $P$, the inference process is guaranteed to terminate in finite time returning one of the values: true, false and undefined as an answer for the queried literal.

Proposition 5.43 associates the answers returned in inference procedure with the concepts of justified and rejected literals.

**Proposition 5.43 (Soundness and Completeness).** For a multi-context system $P = \{P_1, \ldots, P_n\}$ where each $P_i = (id, grp_i, C_i, \Delta_i, T_i)$ is a peer in $P$, and a literal $p_i \in C_i$, the operational model returns:

1. $CT_{p_i} = +\partial$ iff $p_i$ is justified in $P$.
2. $CT_{p_i} = -\partial$ iff $p_i$ is rejected in $P$.
3. $CT_{p_i} = \text{cycle}$ iff $p_i$ appears in a cycle such that literals in the cycles are depending on each other.
4. $CT_{p_i} = \text{undefined}$ iff $p_i$ is neither justified or rejected in $P$.

### 5.7 Prototypical implementation

We have developed a prototype system, called Conspire, using the operational model proposed in the previous section to support our research in distributed defeasible speculative reasoning in ambient environment. It is envisioned to be a common framework for interoperating knowledge-based intelligence agents, as proposed in the Semantic Web [28].

The Conspire framework is built on top of JADE (Java Agent Development Environment). In the prototype, agents can run on different hosts and exchange queries/answer through ACL messages. Agents in the framework are extended with reasoning capabilities and can maintain their own set of knowledge bases and answers reply entry in their data stores (Figure 5.10). Active processes are stored inside a blocking queue and will be processed by the reasoning engine accordingly; while a suspended process will be stored inside the data store and will be put into the queue once it becomes active.

Note that even though defeasible logic is the main topic of this thesis, Conspire was designed to provide greatest flexibility to support research in distributed reasoning such that different reasoning engines can be plugged into the framework and provide reasoning service to an agent provided that a proper Reasoner interface (in Java programming language) has been implemented. In the light of this, agents in the framework can communicate, but not necessarily

---

6Note that the discussion of proof in Appendix B show only the proof of the properties under normal query processing and excluding the cases where answers can be updated after certain time frame.
Figure 5.10: Conspire architecture. In each ambient agent, circle represent the local theories described using a form of logical formalism, double arrows represent the semantic relations and interactions among peers

share a single common logical formalism. Therefore, it is vital for them to find a way to exchange their position arguments seamlessly. To this, our approach is to transform their results into a unified (logical) framework and encode/decode the results using the ACL codec provided by JADE before sending/after receiving the messages. In addition, besides DDSR, agents in the framework can also be configured (individually) using distributed reasoning as described in [4].

To simplify our implementation, currently, we use the Directory Facilitator (DF) service provided by JADE in agents registration and discovery process such that, at any time, an agent may request to modify its own description (in our case, the agent’s context group) in a DF and may issue a search request to a DF to discover descriptions matching supplied search criteria.

Communications between agents are performed based on the iterated version of FIPA-Request protocols [73] such that communication session (between agents) can be terminated when one of the following occurs: (i) a CANCEL message has been sent from the initiator to the responder; (ii) the responder replies with a negation reply, i.e., REFUSE, NOT_UNDERSTAND or FAILURE. In our case this may refer to the case where the queried agent(s) does not contains the literal(s) requested which thus replies with a REFUSE message. (iii) the responder replies with a termination flag to terminate the communication. However, one of the disadvantage of the iterative protocol is that further actions can only be performed after all the result notifications from the responders have been received. To this end, we have modified the protocol a little bit such that a queried agent will issue an INFORM message to a querying agent (the query initiator) the results of the literals concerned are modified.

Since methods on how to update the reputation scores in $T_i$ is not the focus of this thesis, we simply modify the values by adding 0.1 to the agent’s reputation score if the final value justified is the same as the value received from that agent, and minus 0.1 otherwise, and will normalize the scores w.r.t. the total reputation scores of $T_i$ afterwards.
Example Trace

The scenario of the context-aware mobile phone in Section 5.1 (and Example 5.1) has been implemented as an example of our distributed defeasible speculative computation framework. Figure 5.11 and 5.12 show the message flow between agents captured by a JADE sniffer and the result of the user agent (Dr. Amber’s mobile) respectively.

As described before, while receiving a call, Dr. Amber’s mobile will decide whether to ring based on Dr. Amber’s preferences and context, as well as the information derived from the ambient environment.

So, at the beginning, when a call is received, as showed in Figure 5.11\textsuperscript{7}, while inferencing a tentative conclusion with the default hypothesis, the user agent (Dr. Amber’s mobile) will issue two queries to DF (message #1 to #4) simultaneously to acquire the list of agents currently available in the environment which belong to context group \textit{Classroom manager} (for classroom information) and \textit{Class Registry} (for localization information). Upon receiving the information, the user agent then issues queries to these ambient agents to request the necessarily information.

\textsuperscript{7}Note that in the example, agent \textit{ClassRegistryAgent} is configured using \textit{distributed reasoning services} as described in \cite{4}; while others are configured using \textit{defeasible speculative reasoning services} as described in this thesis.
Figure 5.12: User agent: mobile phone

(message #5-#18). Note that even agent CR_sensor-01 is associated with context group Classroom manager, since it is an agent embedded in a sensor, it does not contain the information (class_activity) as per user agent’s request and thus replied with a REFUSE message (message #9); while the other two agents (CR_ManagerAgent-01 and ClassRegistryAgent), which contain the requested information, reply with an AGREE message (message #8 and #10), indicating their willingness of providing the acquired information to the user agent, and then followed by a series of INFORM messages which contains the tentative/final information.

The same pattern has also appeared when the classroom agent CR_ManagerAgent-01 is inferencing on whether there is any class activity (class_activity) in the classroom (message #11, #13 and #15-#17), before replying the user agent.

Then, after accomplishing a series of information acquisition and inferencing process, the user agent finally decided that it should ring as the literal ring is defeasibly provable, as showed in Figure 5.12. In between, while receiving tentative responses from external agents (message #14 and #18), several tentative conclusions have also been generated.

Figure 5.13 and 5.14 show a scenario when another agent, CR_ManagerAgent-X, which belongs to context group Classroom Manager and with higher reputation than CR_ManagerAgent-01, appeared in the environment but with conflicting response, i.e., it has detected that there are class activities currently taking place in the classroom. As discussed, while acquiring information from external agents, since CR_ManagerAgent-X has better reputation than CR_ManagerAgent-01, the information received from CR_ManagerAgent-X on class activity (class_activity) will overrule the one received from CR_ManagerAgent-01. Thus, at this moment, the user agent will believe that Dr. Amber is still having his class actively, consequently implying that it should not ring (Figure 5.14).

In terms of querying time, the time used from sending a REQUEST message to the first INFORM message arrival (conclusions generated using the default hypotheses) by a DDSR
Figure 5.13: Message flow between agents (conflict message)

Figure 5.14: User agent: mobile phone (conflict message)
agent is about 90ms; while the time used by a distributed reasoning from sending REQUEST to answer arrival is about 120ms, indicating that our approach can significantly reduce the response time of a querying agent. Note also that the time used by distributed reasoning agents here do not include the time delay due to user interactions and/or other reasoning/network constraints (such as poor network connection, as described before) which may significantly affect the reasoning message response time.

5.8 Discussion

In this chapter we presented a treatment of distributed defeasible speculative reasoning framework for ambient intelligence. As mentioned before, ambient agents are often used in environments that have a highly dynamic nature. In order to work collaboratively to support human objectives and behave properly, an ambient agent should have the ability to realize the environment, predict user’s intentions and behave pro-actively. Besides, an ambient agent should have the ability to handle incompleteness of information and have a mechanism to recover if the actions turn out to be inappropriate when more information about the current situations of the environment becomes available.

The idea of speculative computation have been employed in several areas of computer science, from optimistic transaction in databases to execution of functional programming and computer architecture. In multi-agents environment, instead of waiting for other agents to provide the information required, an ambient uses a predefined value to continue its computation virtually. In this sense, proactive inferencing is the key of success in speculative reasoning. It improves the efficiency of an ambient agent by inferencing a tentative, but possibly final, set of conclusions using some default hypothesis while waiting for the replies from external agents. If the received replies are consistent with the default hypothesis, the computation will continue without any interruption. Otherwise, the agent will revise its computation with the replies received externally, which features the possibility of reusing parts of the computation already done during the phases when answers arrive and answer revision is required, as the partially computed suspend process may becomes active again and resume its computation. It is particular desirable in situations where some actions should be taken in advance even without having a complete plan (due to information incompleteness), or immediate actions are required from users or other agents.

Even though we may risk wasted work in the speculative computation, which throughput-oriented measures would discourage, agents in the environment have no idea on whether or when an answer will arrive. On the other hand, if we allow the agents to sit idle while waiting for the replies and no default were used, then the computation time of the agents are wasted. In either case, the main overhead is the extra computation required to decide, during answer arrival phase, whether the revised answers are consistent with the existing one [147]. Besides, as pointed out in [170], some computations may be more promising than others. As resources of ambient agents are always limited, it is important to use them efficiently. So, instead of letting
an agent sitting idle while waiting for replies from other agents, it is better if we can allocate resources to computation to favor some promising computations. It is a compromised approach that prevent agents from idling or expending agents’ computation power doing unnecessary computation while waiting for the answers.

Moreover, even if we assume that local theories are consistent, it is not necessary the case in the process of further acquisition from external context. The approach that we used in our framework is a simplified approach that is commonly used in reputation-based system, such as e-Bay, such that agents reputation are updated purely based on the answer received and the revised belief state of the receiving agent. It may not be subjective enough when compare to other frameworks, such as [138] which included a separate yellow pages service, our approach is simple but adequate enough to differentiate the situations that we needed to identify.

Another problem which may appear is that foreign literals may depend on each other and thus forming a loop. This kind of distributed cyclic-literals dependencies has received some attentions in the past decade but only a few systems have been successfully developed [4, 55]. In our framework, this requirement can be removed as the history of queried literals are recorded and will propagate to agent in the next level of query chain. However, as the complexity of queries growth, more time will be required to send and interpret the messages. So, there is often a trade-off between efficiency and flexibility.

Kofod-Petersen and Aamodt [131] has proposed solving the reasoning problem in ambient intelligence through case-based reasoning, and have shown how different concepts for reasoning and modeling be combined. However, as pointed out in [130] their approach may suffer from maintaining the potentially very large case base, which is a risk when running in an on-line manner. In addition, it may not be feasible for an ambient device to store a very high number of cases. On the other hand, Wang et al. [216] have extended the speculative computation framework with deadline (and resources negotiation), and have showed that their approach can improve the accuracy of speculative computation and reduced the risk of the result.

While most of the previous work in speculative computation [124, 53, 121, 147] required agents to be arranged in a hierarchical order such that queries can only be sent from agents in the higher level of the hierarchy to the one in the lower level, our approach does not have this requirement. Agents in the ambient environment can continuously gather and update their belief with acquired new information and response to the external environment. This type of approach is related to the reactive behavior of intelligent agent, which have been studied by Kowalski and Sadri [134] and Dell’Acqua et al. [64, 63] intensively.

The concept of reputation in multi-agent based modeling is gaining popularity. It refers to the trustworthiness of agents in the artificial society [129] and will affect the acceptability of arguments in our framework. As mentioned before, to determine whether an argument is acceptable, it must compare to its counter-argument. In our framework, the preference relation of arguments from local context is defined based the underlying belief base, i.e., in our case, it is defined base on the semantics of DL. However, the preference relation of foreign arguments
have to be defined separately.

In [7], Amgoud et al. introduce the notion of contextual preferences in the form of several pre-orderings on the belief base, which explicitly defined the preference relations on the set of contexts. Conflicts between preferences may appear when these preferences are expressed in different contexts [30]. That is, an argument A is preferred to another argument B in a context $C_1$ while the argument B is preferred to argument A in a context $C_2$. To resolve these conflicts, a meta-preferences, which define the preferences relation in the form of total pre-ordering on the set of contexts, are used. Later, the abstract argumentation framework of Modgil [164] integrates the meta-preferences in Dung’s argumentation theory and extends the notion of defeat to account for preferences between foreign arguments.

Finally, Chatalic et al. [55] extends the consequence finding algorithm for distributed propositional theory proposed by Adjiman et al. [4] to allow for mutually inconsistent peers that are connected by mapping clauses. However, their approach allows for a formula and its negations to be derived at the same time [42].

Nevertheless, none of the approaches mentioned above have considered cases when support of conflicting arguments are received from different multiple contexts, and consider only the support that are received from the most confident, highly reputable agents that are appeared in the environment. This differentiate us from other approaches as all arguments that are received from the external contexts will be put into consideration when deriving a conclusions. Besides, our approach also enables agents in the environment to be part of the system even without register with an authority, i.e., prior knowledge of other agents that appear in the environment is not a necessity under our reasoning framework.

In conclusion, exogenous knowledge exist in the ambient environment will affect the reasoning in ambient agent. The distributed defeasible speculative reasoning framework presented here allows agents in ambient environment to efficiently handle inconsistent, incomplete and revisable information returned by other agents. Thanks to the formalism supported by defeasible logic, our model supports (foreign) literals without negation as failure, and hence can be applied in solving many real-life problems.
Chapter 6

Conclusions

To conclude this thesis, we summarize and discuss its main contributions, and propose directions for future research.

6.1 Synopsis

The province of non-monotonic reasoning is the derivation of a set of plausible, but not fallible, conclusions from a knowledge base. It is a formalization of reasoning with incomplete and inconsistent information and is, in general, understood that the conclusions derived are tentative and can be retracted when new information arrive.

Defeasible Logic (DL) is a sceptical approach to non-monotonic reasoning with distinctive features: (i) low computation complexity (linear w.r.t. the size of a theory), and (ii) its build-in preference handling facilities allowing one to derive plausible conclusions from incomplete or even contradictory information in a natural and declarative way. It is suitable to handle cases of highly dynamic environments, where the availability of data is not always complete and unambiguous, and sometimes even changing continuously.

The present strategy to compute the extensions of a defeasible theory is to apply a series of pre-processing transformations that transform a defeasible theory into an equivalent theory without defeaters and superiority relations, and then applies the reasoning algorithm to the transformed theory.

However, as mentioned in Chapter 3, this approach has suffered from several drawback. First, the transformations would result in an increase in theory size by at most a factor of 12. Secondly, some representation properties of the original theory are removed in the transformed theory. Thirdly, the reasoning algorithm can only be applied when inferencing with ambiguity blocking. Other variants encompass are not supported and cannot be inferred using such approach.

Hence the motivation for undertaking this research is to devise algorithms that can overcome the deficiencies mentioned above when inferencing with defeasible theories under different variants and contexts. In our course of study, we discussed the problem of the present approach and presented new theorems and algorithms to resolve the problem.
Below are the main contributions of this thesis.

In chapter 3, through introducing the notion of superiority chain and inferiorly defeated rule, a theorem was developed which allow us to reason on a defeasible theory without removing the superiority relations. The essence of this approach lies in the ability in identifying the set of redundant rules that can be removed from the theory without affecting the conclusions derived, which help in preserving the representation properties of DL across different variants and simplified the work in the subsequent processes. Our experimental results show that our approach outperforms the present approach, in terms of reasoning time and memory usage, under the same model of complexity. Besides, the two algorithms mentioned above can also be incorporated into this algorithm directly without any modifications.

In addition, we devised algorithms for computing the consequences of ambiguity propagation variant and well-founded semantics of defeasible logic, which contributes to the computational aspect of the two variants in a practical approach such that consequences of both variants (as well as their combination) can be computed with linear complexity, which make DL a possible candidate for some computational demanding jobs, or tasks that require immediate response, such as reasoning on the Semantic Web and in Ambient Intelligence.

To realize the algorithms discussed above, a reasoner, called SPINdle, is implemented, to support our ongoing research in DL. Chapter 4 discussed the system architecture of SPINdle: the components it has, the languages it uses, and the functions that it supports. In short, SPINdle is a defeasible logic reasoner that covers both standard defeasible logic (SDL) and modal defeasible logic (MDL). It can be used as a standalone theory prover or be embedded into any applications as a defeasible rule engines. It is a full implementation of DL that support fact, strict and defeasible rule, defeaters, and superiority relations, and support rules with multiple heads (in both SDL and MDL) and modal operator conversion in MDL. It supports theories represented in XML or language presented in Section 4.1.1.1. To facilitate the process of algorithms/theorem defined in Chapter 3, a specific data structure for storing literals-rules association was defined (Figure 4.4).

SPINdle has been extensively tested for correctness, scalability and performance using different forms of theories generated by a tool that comes with Deimos. Our results show that SPINdle can handle inferences with thousand of rules in less than three seconds, and the reasoning time growth (in most cases) is almost linear to the size of the theories tested, which is coherent with the complexity analyzed. Also, SPINdle, in terms of implementation, does not impose any limitation on the theory size. The largest theory size tested so far is with \( n = 1,000,000 \) (1 million) rules [142].

Chapter 5 describes a formal model for representing and reasoning with the imperfect and distributed context knowledge in Ambient Intelligence environments. We have modeled ambient agents as a (defeasible) logic-based entities in a multi-context environment, and interactions between agents are achieved through mappings of askable literals. In order to handle cases of delayed or missing information, the concept of speculative computation was introduced such that
while waiting for the replies from external agents (which may possibly be delayed or missing),
a tentative (but possibly final) conclusions will be derived based on a set of default hypothesis.
The agent, then, will keep-on updating the current belief state and conclusions set upon arrival
of replies from external agents, as necessary.

We have extended the BIO approach to the development with defeasible logic. We have
extended its capabilities in handling violations and preferences, and can derive the conclusions
with the avoidance of transformations, which consequently speed up the execution of the com-
putations.

On the top of this, an argumentation framework that extends the argumentation semantics
of defeasible logic is developed. It introduces the notion of argument rank, which is based on
Herbert Simon’s theory of docility, as a preference generator when multiple and/or conflicting
arguments are received from different agents. Besides, we have also introduced an operational
modal with iterative belief revision for the framework based on speculative computation.

Lastly, a prototypical implementation based on a context-aware mobile phone scenario show-
casing the wealth of our approach have been implemented.

6.2 Future Directions

Research is an ongoing process. Even if the algorithms and tool presented in this thesis have
achieved their intended goals, our study on defeasible reasoning can be extended in various
dimensions. In the following, we will present some ideas which show the directions of future
research could take.

6.2.1 Inference algorithm without using transformations

Even though propositional defeasible theory can be inferenced in linear time and the algorithm
that we presented can lower the constant further by a factor of 4, the current approach still relies
on two transformations to: (1) transform the input theory into regular form, and (2) eliminate
defeaters. To further optimize the inference process, our preliminary investigations have showed
that, in addition to superiority relations, defeaters can also be incorporated into the inference
process without affecting the complexity.

Currently, we are working on the proof of soundness and completeness as well as the imple-
mentation of the new algorithm. We believe that our work would foster the development on
defeasible reasoning as more efficient algorithms can be devised. Besides, it would also be in-
teresting to investigate on whether this approach can be, in general, applied to other rule-based
non-monotonic formalism which use preference operators to describe the relative strength of
rules, such as Preference Logic.
6.2.2 Temporal Defeasible Logic

A topic which has not been addressed in this work is the extension of (Modal) Defeasible Logic into temporal domain. Temporal Defeasible Logic (TDL) extends standard defeasible logic with time. In [100, 106, 105, 103, 102] defeasible logic has been extended to capture some temporal aspect in legal reasoning. [105] provided a conceptual analysis of deadlines in TDL. [100] proposes an extension to cope with durative facts and with delays between the antecedent and the consequent of rules. Recently, a family of TDL [106, 102] are proposed to cater the normative modifications cases that may arise in legal reasoning, and the complexity of TDL has been studied formally in [96].

However, the approach in [96] is still based on the algorithm proposed by Maher [151]. That is, a series of theory transformations (to eliminate superiority relations and defeaters) is needed to be performed prior to the inference process, and their results are restricted only to theories where the superiority relation is empty.

In the light of this, we believe that there is room for a complete study of TDL in which more advanced algorithm(s) can be devised to cater different aspects (variants) of defeasible reasoning in the temporal domain.

6.2.3 Argumentation Semantics and Game Theory

Agents in an ambient environment are self-interested in the sense that they only interest in furthering their own goals, which may or may not be coincide with others. When the goal does not depend solely on the agents’ factual premises, they would engage in argument. At first sight, similar to the strategy that we presented in Chapter 5, one may conclude from this observation that the outcome of a dispute cannot be predicted at all, and content oneself with setting the rules of the procedure. However, on the other hand, one can also argue strategically in such a way that makes it more likely for their argumentative goals to be achieved [180]. Roth et al. [188] have the following example in their paper:

A worker is dismissed for having caused considerable damage to company property and for having lost credit from his superiors. The worker challenges the dismissal in court, claiming that it can be voided. He argues that the working atmosphere has not been affected and that he is highly esteemed as a colleague. The following four legal rules are in place. The first says that a dismissal can be voided if considerable damage was done to company property but the working atmosphere was not affected. The second says that the dismissal can be voided if the worker lost credit from his superiors but is highly esteemed as a colleague. The third says that the dismissal cannot be voided if the working atmosphere was not affected but the worker lost credit from his superiors. The forth rule says that the dismissal cannot be voided if the worker is highly esteemed as a colleague and did considerable damage to company property. It is commonly accepted by both the worker and the employer that it is very probable that the worker is highly esteemed as a worker, less probable that the
working atmosphere has not been affected, even less probable that the damage was considerable, and the least probable that the worker lost credit from his superior

So which strategy is best for the worker? Should he use the first rule first and then see what the employer does? If the employer uses the third rule, should the employer use the second one, or just simply stick to the first? Likewise, what should the employer do? Or, in general, what will both parties do in the dispute?

The mathematically study of strategic interaction is Game Theory, which was first established by von Neumann and Morgenstern in 1944 [211]. A setting of strategic interaction is modeled as a game which consists of a set of players, set of action available to them, and a set of rules that determine outcomes of players’ chosen action [180]. The prisoner’s dilemma is a fundamental problem in game theory that demonstrate two people might not cooperate even if it is in both their best interest to do so.

[188] is a prior to this (in terms of defeasible reasoning) and filled the gap by using defeasible logic in combination with standard probability calculus so as to make sure that the proof conditions hold. The probability of a claim was then interpreted in the game theoretical sense as the payoff of the component for the proponent of the claim, which allow them to analyse the problem and prescribe the strategies that the agent should adopt.

However, it seems that little has been done in this area, even outside the scope of defeasible reasoning. In this sense, it is among our future plans to extend the argumentation semantics framework of defeasible reasoning to cover the cases in game theory and we believe that this will introduce a new line of research in the research community.

6.2.4 Business Process Compliance

Business processes specify the activities a business does to achieve its business goals. It is typically regulated by authorities that stem from legislation and regulatory bodies (e.g. ETH, IEEE, HIPAA), standards and codes of practice (e.g. ISO9000, SCOR), but also from business contracts that regulate cross-organizational interactions. It describes how a process is execute - what can be done and what cannot be done by a process.

Business Process Compliance (BPC) corresponds to the adherence or consistence of a set of specifications modeling a business process and a set of specifications modeling the norms for a particular business [104]. It is a process that aims at ensuring that business process, operations and practice are in accordance with a prescribed and/or agreed set of norms [98].

Checking compliance amounts to an affordable operations when the corresponding processes are small or simple. But things are tremendously harder when the processes become complex and larger, or articulated systems such as bodies of legal provisions. In [98], the authors have stated a few source of complexities that may arise when developing compliance checking system, namely: exception handling, heterogeneities of obligations, and level of automated detection. Exception handling resides in the fact that legal norms regulate processes by usually specifying actions to be taken in case of breaches of some of the norms. Heterogeneities of obligations states the
conditions where different types of obligations may require to regulate a process; while the level of automated detection specifies whether it is required to check compliance at runtime or design time.

Currently there are two main approaches towards achieving compliance: auditing and monitoring [99, 191]. However, both methods are based on retrospective reporting and are suffering from the same problem that there is no guarantee to fix the processes correctly when compliant did not occur.

To resolve this problem, in [95, 98], the authors have proposed a method to specify the activities of a business process and the conditions to set up the norms relevant for the process, and have presented an algorithm to determine whether a process is compliant by defining a new formalism called Process Compliance Language (PCL) based on defeasible logic. In the sense of this, it is interesting for us to investigate how defeasible logic (and our algorithms) can foster the study of compliance with a rich ontology of obligations, which has been neglected in the literature.

6.3 Summary

In conclusion, this thesis contributes techniques for inferencing with defeasible logic under different variants and has presented a new approach to the inference process such that more advanced, efficient algorithm can be devised. Our result show that, under the same model of complexity, our approach (significantly) outperform the present approaches in both reasoning time and memory utilization. Besides, we also developed an argumentation framework and operational model for distributed query processing with defeasible logic and speculative computation. Further, we also show ways of how (modal) defeasible reasoning can be extended to solve our daily problems.

However, as mentioned above, there are still many open research questions that need to be solved. We hope that the success of our techniques in defeasible reasoning can shed light on the same issue in other rule-based non-monotonic formalisms, and other (application) domains in the area of logic reasoning as well as knowledge representation as a whole.
Bibliography


[63] Pierangelo Dell’Acqua and L.M. Pereira. Enabling Agents to Update their Knowledge and to Prefer Progress in Artificial Intelligence. In P. Brazdil and A. Jorge, editors, 10th Portuguese Int. Conf. on Artificial Intelligence (EPIA01), LNAI 2258, pages 183–190, 2001.


[75] Johnson Fong, Ho-Pun Lam, Ricky Robinson, and Jadwiga Indulska. Defeasible preferences for intelligible pervasive applications to enhance eldercare. In *PerCom Workshops*, Lugano, Switzerland, 19-23 March 2012. IEEE.


Appendix A

Test theories

Description of scalable test theories generated from DTScale tool of Deimos (depicted from [185]).

A.1 Chain Theories

Chain theories chain(n) start with a fact $a_0$ and continue with a chain of defeasible rules of the form $a_{i-1} \Rightarrow a_i$. A proof of $+\partial a_n$ will use all of the rules and the fact.

$$\text{chain}(n) = \begin{cases} r_1 : a_0 \Rightarrow a_1 \\ r_2 : a_1 \Rightarrow a_2 \\ \vdots \\ r_n : a_{n-1} \Rightarrow a_n \end{cases}$$

A variant of chain(n) uses only strict rules.

A.2 Circle Theories

Circle theories circle(n) consists of $n$ defeasible rules of the form $a_i \Rightarrow a_{(i+1) \mod n}$. Any proof of $+\partial a_i$ will loop.

$$\text{circle}(n) = \begin{cases} r_0 : a_0 \Rightarrow a_1 \\ r_1 : a_1 \Rightarrow a_2 \\ \vdots \\ r_{n-1} : a_{n-1} \Rightarrow a_0 \end{cases}$$

A variant of circle(n) uses only strict rules.

A.3 Teams Theories

Teams theories teams(n) consists of conclusions $a_i$ which are supported by a team of two defeasible rules and attacked by another team of two defeasible rules. Priorities ensure that attacking rule is beaten by one of the supporting rules. The antecedents of these rules are in turn supported and attacked by cascades of teams of rules.

$$\text{teams}(n) = \text{block}(a_0, n)$$
where, if \( p \) is a literal, and \( r_1, \ldots, r_4 \) are new unique labels:

\[
\text{block}(p, 0) = \begin{cases} 
    r_1 : \Rightarrow p \\
    r_2 : \Rightarrow p \\
    r_3 : \Rightarrow \neg p \\
    r_4 : \Rightarrow \neg p \\
    r_1 > r_3 \\
    r_2 > r_4 
\end{cases}
\]

and, if \( n > 0, a_1, \ldots, a_4 \) are new unique literals, and \( r_1, \ldots, r_4 \) are new unique labels:

\[
\text{block}(p, n) = \begin{cases} 
    r_1 : a_1 \Rightarrow p \\
    r_2 : a_2 \Rightarrow p \\
    r_3 : a_3 \Rightarrow \neg p \\
    r_4 : a_4 \Rightarrow \neg p \\
    r_1 > r_3 \\
    r_2 > r_4 \\
    \text{block}(a_1, n-1) \\
    \text{block}(a_2, n-1) \\
    \text{block}(a_3, n-1) \\
    \text{block}(a_4, n-1) 
\end{cases}
\]

A proof of +\( \partial a_0 \) will use every rule and priorities.

### A.4 Tree Theories

In tree theories \( \text{tree}(n, k) \) \( a_0 \) is at the root of a \( k \) \- branching tree of depth \( n \) in which every literals occur once.

\[
\text{tree}(n, k) = \text{block}(a_0, n, k)
\]

where, if \( p \) is a literal, \( n > 0, r \) is a new unique label, and \( a_1, a_2, \ldots, a_k \) are new unique literals:

\[
\text{block}(p, n, k) = \begin{cases} 
    r : a_1, a_2, \ldots, a_k \Rightarrow p \\
    \text{block}(a_1, n-1, k) \\
    \text{block}(a_2, n-1, k) \\
    \vdots \\
    \text{block}(a_k, n-1, k)
\end{cases}
\]

and:

\[
\text{block}(p, 0, k) = \{ p \}
\]

A proof of +\( \partial a_0 \) will use every rule and fact.

### A.5 Directed Acyclic Graph Theories

In directed acyclic graph theories \( \text{dag}(n, k) \), \( a_0 \) is at the root of a \( k \) \- branching tree of depth \( n \) in which every literals occur \( k \) times.
\[ \text{dag}(n, k) = \begin{cases} 
\quad \Rightarrow a_{kn+1} \\
\quad \Rightarrow a_{kn+2} \\
\quad \vdots \\
\quad \Rightarrow a_{kn+k} \\
\quad r_0 : \quad a_1, a_2, \ldots, a_k \Rightarrow a_0 \\
\quad r_1 : \quad a_2, a_3, \ldots, a_{k+1} \Rightarrow a_1 \\
\quad \vdots \\
\quad r_{nk} : \quad a_{nk+1}, a_{nk+2}, \ldots, a_{nk+k} \Rightarrow a_{nk} 
\end{cases} \]

A proof of \( +\partial a_0 \) will use every rule and fact.
Appendix B
Proofs

Theorem 3.16. Let $D = (\emptyset, R, >)$ be a defeasible theory (in regular form), and $r \in R_{w}[q]$ with number of weaker rules equals to zero. Let $D' = (\emptyset, R \setminus \{r\}, >')$ be the reduct of $D$ with response to $r$, denoted by $\text{reduct}(D)$, where $>'$ is defined as:

$$> \setminus \{s > r\} : \exists s \in R_{ad}[-q], A(s) = \emptyset \text{ and } s > r$$

Then $D \equiv D'$.

In addition, $-\partial q$ can be derived if $R[q] = \emptyset$ after the removal of $r$.

Proof. To recap (Definition 3.13), a superiority rule chain is a superiority relation hierarchy such that, apart from the first and last element of the chain, there exists a superiority relation between rules $r_k$ and $r_{k+1}$:

$$r_1 > r_2 > \cdots > r_n$$

where $n$ is the length of the chain, and $C(r_k) = -C(r_{k+1}) \forall 1 \leq k < n$.

And the following two lemmas follow directly from the definition of the derivation conditions.

Lemma B.1. Let $P$ be a derivation in a defeasible theory $D$, $q$ a literal, and $d \in \{\partial, \delta\}$. If $+\Delta \in P$ then $P \& (+dq)$ is a derivation in $D$. So $+\Delta(D) \subseteq +d(D)$.

Lemma B.2. Let $P$ be a derivation in a defeasible theory $D$, $q$ a literal, and $d \in \{\partial, \delta\}$. If $-dq \in P$ then $-\Delta q \in P$. So $-d(D) \subseteq -\Delta(D)$.

The meaning of Lemma B.1 is that once we have established that a literal is positively provable we can remove it from the body of rules without affecting the set of conclusions we can derive from the theory. Similarly, Lemma B.2 states that we can safely remove rules from a theory when one of the elements in the body of the rules is negatively provable.

We proof by induction on the length of derivations $P$ in $D$ and contradiction.

Induction base ($n = 1$). Suppose the length of a proof $P$ is 1. The only line in $P$, $P(1)$ is either $+\Delta q$ or $-\Delta q$ and the proposition holds trivially.

Inductive step ($n > 1$). We consider only the cases of defeasible conclusions, since, as $D$ and $\text{reduct}(D)$ have the same definite conclusions.

Suppose the proposition hold for all derivations of length $\leq n$. Let $P$ be a derivation of length $n + 1$, and $q$ be a tagged literal in the language $\Sigma$. We have divided the proof into two cases. We first prove that if $D \vdash q$ then $\text{reduct}(D) \vdash q$, and the other direction, namely: if $\text{reduct}(D) \vdash q$ then $D \vdash q$.

Suppose $r \in R[q]$ and $r \notin R'$, $s \in R[-q]$ such that $\forall a_s \in A(s), +\partial a_s \in P(1..n)$ and $s > r$, and $\{t \in R_{ad}[-q] : r > t\} = \emptyset$. 

IV
Case $\Rightarrow$. Let $W = \{w : w \in R[q]\}$, $S = \{s : s \in R[\neg q]\}$, $W' = \{w' : w' \in R'[q]\}$ and $S' = \{s' : s' \in R'[\neg q]\}$.

Case $D \vdash +\partial q \Rightarrow D' \vdash +\partial q$. This means $r$ is defeated and is overruled by $s$ irrespective to whether $r$ is applicable or not. Let $r^* \in R_{ad}[q]$. Then if $r^* = r$ is the only rule in $R[q]$, by induction hypothesis there exists a derivation $P$ in $D$ such that $P(k) = +\Delta q$, thus we can extend $P$ with $+\partial q$.

Next, if $r^* \neq r$, we assume on the contrary that $D' \not\vdash +\partial q$ while $D \vdash +\partial q$. Then according to clause (2.3.2) of the definition of $+\partial$, $D \vdash +\partial q$ iff for every applicable rule $s^* \in S$, $\exists w \in W$ s.t. $\forall a_w \in A(w)$, $+\partial a_w \in P(1..n)$ and $w > s^*$.

However, $D' \not\vdash +\partial q$ means that $\exists s' \in S'$ s.t. $\forall a_{s'} \in A(s')$, $+\partial a_{s'} \not\in P(1..n)$ and for every applicable rule $w' \in W'$, $w' \neq s'$, which gives us the desire contradictions since by definition $R' = R \setminus \{r\}$, implying that $W' = W \setminus \{r\}$ and $S' = S$.

Since $r$ does not belongs to $R'$ and the derivation does not depend on it, the derivation is also a derivation in $D'$.

Case $D \vdash -\partial q \Rightarrow D' \vdash -\partial q$. Let $r^* \in R_{ad}[q]$. If $r^* = r$ is the only rule in $R_{ad}[q]$, then $r$ is defeated and the proposition is trivially true.

Next, if $r^* \neq r$, by inductive hypothesis there exists a derivation $P$ in $D$ such that $P(k) = +\Delta \neg q$ and $\forall w \in W$ either $\exists a_w \in A(w)$, $-\partial a_w \in P(1..n)$ or there exists an applicable rule $s^* \in S$ such that $w \neq s^*$, thus we can extend $P$ with $-\partial q$. Since $r$ does not in $R'$ and the derivation does not depend on it, the derivation is also a derivation in $D'$.

Case $D \vdash +\partial q \Rightarrow D' \vdash +\partial q$. This case is trivial as the conditions satisfied are either there exists a derivation $P$ in $D$ such that $P(k) = +\Delta q$ and $\forall w \in W$ either $\exists a_w \in A(w)$, $-\partial a_w \in P(1..n)$ or there exists an applicable rule $s^* \in S$ such that $w \neq s^*$.

In either case, removing $r$ from $R$ does not affect the derivation. Thus the derivation is also a derivation in $D'$.

Case $D \vdash -\partial q \Rightarrow D' \vdash -\partial q$. By inductive hypothesis, $D \vdash -\partial q$ implied that there exists a derivation $P$ in $D$ such that $P(k) = +\Delta q$ and either

1. $\forall s^* \in S, \exists a_{s^*} \in A(s^*)$ such that $-\partial a_{s^*} \in P(1..n)$; or
2. $\exists w^* \in W$ such that $\forall a_{w^*} \in A(w^*)$, $+\partial a_{w^*} \in P(1..n)$ and $\forall s^* \in S$ either $\exists a_{s^*} \in A(s^*)$, $-\partial a_{s^*} \in P(1..n)$ or $s^* \neq w^*$.

The first case is obviously cannot be satisfied according to the definition.

So, for the second case, we assume on the contrary that $w^* = r$. Then by definition, we have $s \in S$ such that $\forall a_s \in A(s)$, $+\partial a_s \in P(1..n)$ and $s > r$, which contradicts the condition above saying that $s \neq r$. Thus our assumption is false and $w^* \neq r$. That is, by inductive hypothesis, $\exists w \in W, w \neq r$ and $w \neq s$ such that $\forall a_w \in A(w)$, $+\partial a_w \in P(1..n)$ which falsify the conclusion derived by $s$, and consequently extending $P$ with $-\partial q$. So $r$ is redundant in this case as the derivation does not depend on, implying that the derivation is also a derivation in $D'$.

Case $\Leftarrow$.

The proof is by induction on the length of derivations.

**Inductive Base** For the inductive base we notice that the only two possibilities for derivations of length one is when we have either (i) $+\Delta l$ justified since there is a rule $r$ in $R_s[l]$.

---

1Here the condition where $r$ is the weakest rule in a superiority rule chain is important. Consider a situation where $r$ is not the weakest rule in a superiority rule chain, i.e., $\exists s^* \in S$ s.t. $r > s^*$ and was defeated in $D$. However, removing $r$ from $R$ may results in activating $s^*$ in $D'$, which leads to undesired conclusions.
such that $A(r) = \emptyset$ or (ii) $-\Delta l$ because $R_s[l] = \emptyset$. By construction, $R_s = R'_s$, and defeasible rules and the superiority relation do not play any role in the conditions for definite provability. Thus we have established the inductive base for both directions of the proof.

**Inductive Step (if $D' \vdash \# l$ then $D \vdash \# l$).** Here the inductive hypothesis is that the property holds for proofs of length up to $n$.

Case $\Delta$. The cases of definite provability follow immediately from the inductive hypothesis and the remarks about the set of strict rules being the same in $D'$ and $D$, and the nature of the conditions for $+\Delta$ and $-\Delta$.

For $\pm \partial l$, first we have to distinguish two cases: (i) the first case when $l$ is neither $q$ nor $\sim q$, then $R[l] = R'[l]$. Thus, we can immediately use the inductive hypothesis to obtain the desired results.

For the second case, we have to analyse four sub-cases, namely: $P(n+1) = +\partial q$, $P(n+1) = +\partial \sim q$, $P(n+1) = -\partial q$, and $P(n+1) = -\partial \sim q$. Case: if $D' \vdash +\partial q$ then $D \vdash +\partial q$. The difference between $D'$ and $D$ is that in $D$ we have an extra rule, $r^*$, in $R_d[q]$, this means that in the derivation of $q$ in $D'$ we do not use anywhere $r^*$, and then, given the inductive hypothesis, a derivation of $q$ in $D'$ is also a derivation of $q$ in $D$.

Case: if $D' \vdash +\partial q$ then $D \vdash +\partial q$ In this case we have to show that given the conditions of the theorem on $r^*$, it is not possible to use $r^*$ to prevent a derivation of $\sim q$. Again, by construction we know that $R[\sim q] = R'[\sim q]$ and we have already proved that $D$ and $D'$ are equivalent as far as provability of definite literals is concerned. This means that we can concentrate on clause 2.3 of the proof conditions for $+\partial$. However, we know that there is a rule $s$ such that $s \succ r^*$, $s \in R[\sim q]$, and $A(s) = \emptyset$. This means that, trivially $s$ satisfies the condition that all the elements in its body are provable, and thus clause 2.3.2 is satisfied, which make the whole condition for proving $+\partial \sim q$ satisfied. Thus we have a proof of $+\partial \sim q$ in $D$.

Case: if $D' \vdash \sim q$ then $D \vdash \sim q$ Suppose it does not hold. Thus we have a derivation $P$ in $D'$ satisfying the conditions but we do not have a derivation of it in $D$. Since we work with regular theories, then we restrict ourselves to clause 2 of the proof conditions for $-\partial$.

This means that in $D$ the following conditions is satisfied:

\begin{align*}
(2.1) \exists r \in R_{sd}[q] & \text{ such that } \forall a_r \in A(r), -\partial a_r \notin P(1..m) \text{ and } \\
(2.2) \forall s \in R[\sim q] & \text{ either } \\
(1) \exists a_s \in A(s) & \text{ such that } +\partial a_s \in P(1..m) \text{ or } \\
(2) \exists t \in R[q] & \text{ such that } \\
(1) \forall a_t \in A(t) & -\partial a_t \notin P(1..m) \text{ and } \\
(2) t > s. 
\end{align*}

Lemma 5.36 (Monotonicity). The sequences of sets of arguments $J_i^p$ and $R_i^p$ are monotonically increasing.

**Proof.** We prove this lemma by induction on $i$. The induction base is trivial in both cases since both $J_0^p = \emptyset$ and $R_0^p = \emptyset$ which thus $J_0^p \subseteq J_1^p$ and $R_0^p \subseteq R_1^p$.

VI
By definition strict local argument are acceptable to every set of arguments, thus they are in every $J^n_P$.

Let $A$ be an argument in $J^n_P$ and $B$ be an argument in $Args_P$ defeating $A$. By definition, $B$ is undercut by $J_{n-1}^P$. That is, for every argumentation line $ALC_i(B)$ with head $B$, $\exists q \in C$ and an argument $D \in J_{n-1}^P$ s.t. $D$ defeat a proper subargument of $B$ or an argument in $ALC_i(B) \setminus B$ at $q$. By inductive hypothesis, $J_{n-1}^P \subseteq J_n^P$, $D$ is supported by $J_n^P$. Consequently, $B$ is undercut by $J_n^P$. Since $A$ is an argument in $J_n^P$, by definition $A$ is supported by $J_{n-1}^P$, and by inductive hypothesis $A$ is supported by $J_n^P$. Therefore, $A$ is acceptable w.r.t. $J_n^P$. Thus, $A \in J_{n+1}^P$.

Next we consider the sequence of rejected arguments $R^n_P$. Let $A$ be an argument in $J_n^P$. Then, according to definition, $A$ is not a strict local argument and at least one of the following conditions hold: (a) a proper subargument $A'$ of $A$ is in $R_{n-1}^P$, (b) for every argumentation line $ALC_i(A)$ with head $A$ there exists an argument $A' \in ALC_i(A) \setminus A$ s.t. either a subargument of $A'$ is in $R_{n-1}^P$. For either case, by inductive hypothesis, $R_{n-1}^P \subseteq R_n^P$, hence $A' \in R_n^P$, and $A \in R_{n+1}^P$. (c) a proper subargument of $A$ or an argument in $ALC_i(A) \setminus A$ is defeated by an argument supported by $JArgP$. In this case, $A \in R_i^P$ for every $i$, and therefore $A \in R_{n+1}^P$. □

Lemma 5.37 (Coherence). In a Defeasible Multi-Context System $P$:

- No argument is both justified and rejected.
- No literal is both justified and rejected.

Proof. Suppose there is an argument that is both justified and rejected. Let $n$ be the smallest index such that, for some argument $A$, $A \in RArgP(JArgP)$ and $J_n^P$. Since $A \in J_n^P$, it holds that: (a) $A$ is a strict local argument; or (b) $A$ is supported by $J_{n-1}^P$ and every arguments defeating $A$ is undercut by $J_{n-1}^P$. Since $A \in RArgP(JArgP)$, (a) does not hold, meaning that there should, at least, exists an argumentation line $ALC_i(A)$ with head $A$ such that for every subargument of $A$ or arguments in $ALC_i(A) \setminus A$, $A'$, it holds that $A' \in J_{n-1}^P \subseteq J_n^P$, and every argument $B$ that attacks $A'$ is undercut by $J_{n-1}^P$. That is, there exists an argument $D$, supported by $J_{n-1}^P$, that attacks a proper subargument of $B$ or an arguments in $ALC_i(B) \setminus B$, namely $B'$.

However, since $B'$ is supported by $JArgP$, $D$ is undercut by $JArgP$, that is, there exists an arguments $E$, supported by $JArgP$, that attacks a proper subargument of $D$ or an arguments in $ALC_i(D)$, namely $D'$. $D' \in J_{n-1}^P$ since $D'$ is a proper argument of an argument supported by $J_{n-1}^P$. Moreover, $D'$ is rejected, since it is attacked by an argument $E$ that is supported by $JArgP$. But this contradicts the assumed minimality of $n$. Hence the original supposition is false, and no argument is both justified and rejected.

The second part follows easily from the first: if $p$ is justified there is an argument $A$ for $p$ in $JArgC_i$. From the first part, $A \in ArgsP - RArgsP(JArgP)$. Thus if $p$ is justified, it is not rejected. □

Lemma 5.38. If the set of justified arguments in $P$, $JArgP$, contains two arguments with conflicting conclusions, then both arguments are strict local arguments.
Proof. Let the two arguments be $A$ and $B$. Suppose $B$ is a strict local argument. Then, for a to be acceptable w.r.t. any $S$, $A$ must be a strict local argument (since $B$ attacks $A$, and $B$ cannot be attacked or undercut, because it is strict local argument). Thus, by symmetry, either $A$ and $B$ are both strict local arguments, or they are both defeasible local or mapping arguments. Suppose that both are defeasible local or mapping arguments, and $B$ defeats $A$. Then $A$ must be rejected because it is defeated by an argument supported by $J\text{Ar}g_{S}^{F}$ and is justified by assumption. If we assume that $A$ defeat $B$, then $B$ will become both justified and rejected. Thus, both cases, by Lemma 5.37, are of impossible. That is, no two non-strict local justified arguments have conflicting conclusions.

Proposition 5.42 (Termination). For a multi-context system $P = \{P_{1}, \ldots, P_{n}\}$ where each $P_{i} = (id, gr_{i}, C_{i}, \Delta_{i}, T_{i})$ is a peer in $P$, the inference process is guaranteed to terminate in finite time returning one of the values: true, false and undefined as an answer for the queried literal.

Proof. At each call, the iterative step in the process reduction phase checks for the local answers for all literals in their bodies and all strict rules with head $p_{i}$ (or $\sim p_{i}$). By definition, all rules in $C_{i}$ are finite. Since $V_{i}$ (the vocabulary used by $C_{i}$) is a finite set of literals, each rule contains a finite set of literals in its body. Therefore, the step in checking non-askable literals include a finite number of operations. Since we assumed that there is no loops in a context theory, each such iterative step would be for a different literal in $V_{i}$. Consequently, the local literals assignment process terminates in finite time and returning either true or false as a local answer for the queried literal, or undefined if $C_{i}$ does not contain any rules with head $p_{i}$ nor $\sim p_{i}$.

Besides, the iterative step will also checks the answers for all literals in the bodies of rules with head $p_{i}$ or $\sim p_{i}$. By definition, all such rules are defined by $C_{i}$ and are finite in number. Since each foreign added to the goal set through the literal assignment process is in vocabulary $V_{j}$ of a context group $C_{j}$, where $i \neq j$, and that each vocabulary is a finite set of literals, each such rules contains a finite set of literals in their body. And at each step, the structure that keeps track of the history of the query (hist) is augmented with the set of askedable literals in $\{GS \setminus q_{j}\}$, where $q_{j}$ belongs to the vocabulary $V_{j}$ of a context group $C_{j}$ and $q_{j}$ is not already in the hist. As the number of external context is finite, the total number of vocabulary in each group is a finite set of literals, the total number of queries issue is bounded by the total number of literals $V = \bigcup V_{i}$ in $P$. That is, the process reduction phase will be terminated in finite time.

Since total number of queries issue is bounded above by the total number of literals in the system, the answers received by the answer arrival phase are also upper bounded. Consequently, the processes generated due to answer revisions are limited. Hence, the answer arrival phase will also be terminated in finite time.

That is, since the process reduction phase and answer arrival phase terminate in finite time, the inference procedure will terminate in finite time. By definition, it is trivial to verify that one of the values: true, false and undefined is returned as an answer for $p_{i}$ upon termination.

Proposition 5.43 (Soundness and Completeness). For a multi-context system $P = \{P_{1}, \ldots, P_{n}\}$ where each $P_{i} = (id, gr_{i}, C_{i}, \Delta_{i}, T_{i})$ is a peer in $P$, and a literal $p_{i} \in C_{i}$, the operational model returns:

1. $CT_{p_{i}} = +\varnothing$ iff $p_{i}$ is justified in $P$.
2. $CT_{p_{i}} = -\varnothing$ iff $p_{i}$ is rejected in $P$. 

VIII
3. \( CT_{p_i} = \text{cycle} \iff p_i \text{ appears in a cycle such that literals in the cycles are depending on each other.} \)

4. \( CT_{p_i} = \text{undefined} \iff p_i \text{ is neither justified or rejected in } P. \)

**Proof.** (\( \Rightarrow \)) We prove by induction on the length of the argumentation line \( AL_{p_i} \) for proving \( p_i \).

**Induction base.** Suppose the length of argumentation line \( AL_{p_i} \) be 1 to case 1 and 2, and 2 to case 4.

**Case 1** If the inference procedure returns \( Ans_{p_i} = \text{true} \) in one call, this means that either (a) there is a strict local argument \( A \) for \( p_i \) in \( Args_P \). Hence \( A \in Args_P \) and \( p_i \) is justified; or (b) there is a local defeasible rule \( r_i \) in \( C_i \) s.t. \( A(r_i) = \emptyset \) and there is no applicable rule with head \( \sim p_i \) in \( C_i \). Therefore, there is an argument \( A \) for \( p_i \) in \( Args_P \) with root \( p_i \), which contains only rule \( r_i \), and with no argument attacking \( A \). Since \( A \) has no proper subarguments and it is not attacked by any argument, \( A \in Args_P \); therefore \( p_i \) is justified.

**Case 2** If the inference call returns \( Ans_{p_i} = \text{false} \) in one call, this means that there is no strict local argument for \( p_i \) in \( Args_P \) and either (a) there exists an local strict argument \( B \) for \( \sim p_i \) in \( Args_P \) which is by definition supported by \( Args_P \), and therefore \( p_i \) is rejected by \( Args_P \); or (b) there is a local defeasible rule \( s_i \) in \( C_i \) with head \( \sim p_i \) such that \( A(s_i) = \emptyset \). Therefore, there is an argument \( B \) for \( \sim p_i \) in \( Args_P \), with root \( \sim p_i \), which contains only rule \( s_i \). For \( B \) it holds that it has no proper subarguments - it is supported and not undercut by \( JArgs^P \). Therefore it defeats any non-strict local arguments for \( p_i \). Since there is no strict local argument for \( p_i \) in \( Args_P \), every argument for \( p_i \) is defeated by \( B \); thus \( p_i \) is rejected by \( JArgs^P \).

**Case 4** Since we assume that there is no loops in the local context theories, there are no rules s.t. the literal in their head also belongs to the body of the rule. Hence, for the inference procedure return \( Ans_{p_i} = \text{undefined} \), the following conditions hold: (a) there is no strict local argument for \( p_i \) in \( Args_P \), (b) there is no rule with head \( \sim p_i \) in \( P \) which means that there is no argument in \( Args_P \) attacking the arguments for \( p_i \) at their root, (c) there is only one rule \( r_i \) with head \( p_i \) in \( P \) and with only one askable literal \( p_j \in C_j \) in its body, (d) (at the same time) only one rule \( r_j \) head \( p_j \) in \( P \) such that \( p_j \) is the only askable literal in its body; (e) there is no rule with head \( \sim p_j \) in \( P \), and (f) \( p_j \notin \Delta_i \) and \( p_i \notin \Delta_j \). Hence, the only argument \( A \) for \( p_i \) can be obtained using \( r_i \) and the only argument \( B \) for \( p_j \) can be obtained using \( r_j \), are neither justified by \( JArgs^P \) nor rejected by \( JArgs^P \) (since there are not attacking argument). Therefore, \( p_i \) is neither justified nor rejected by \( JArgs^P \).

**Inductive step.** Suppose that the proposition hold for all arguments of length \( \leq n \), where \( n > 1 \) to case 1 and 2, and \( n > 2 \) to case 4. Let \( P \) be a derivation of length \( n + 1 \).

**Case 1** For the inference procedure returns \( Ans_{p_i} = \text{true} \) in \( n + 1 \) calls, the following conditions hold:

(a) there is a rule \( r_i \) with head \( p_i \) in \( C_i \) s.t. for all literals \( \alpha \) in its body it holds that \( Ans_{\alpha} = \text{true} \) is returned by the procedure in at most \( n \) call. That is, by inductive hypothesis, for every literal \( \alpha \), there is an argument \( A_{\alpha} \) with conclusion \( \alpha \) in \( JArgs^P \). Therefore, for every \( A_{\alpha} \), it holds that either \( A_{\alpha} \) is a local argument, or its head is in an argumentation line \( AL_{\alpha} \) s.t. every argument in \( AL_{\alpha} \) is in \( JArgs^P \). Using argument \( A_{\alpha} \), the argumentation line \( AL_{\alpha} \) and the rule \( r_i \), we can build an argument \( A \) for \( p_i \) and an
argumentation line $AL_{p_i}$ with head $p_i$ s.t. every proper subarguments of $A$ and every argument in $AL_{p_i} - A$ are in $JArgs^P$. Consequently, $A$ is supported by $JArgs^P$.

(b) there is no strict local argument for $\sim p_i$ in $Argsp$.

c) for all rules $s_i$ with head $\sim p_i$ in $C_i$ either (i) there is a literal $b$ in the body of $s_i$ for which the inference procedure returns $Ans_b = false$ in $n$ calls. By inductive hypothesis, $b$ is rejected by $JArgs^P$, which means that every argument for $b$ is defeated by an argument supported $JArgs^P$. Hence for every argument $B$ using $s_i$ in $Argsp$ is undercut by $JArgs^P$; or (ii) $\forall b \in A(s_i)$ the inference procedure returns either $true$ or undefined as an answer for $b$ in at most $n$ calls and $p_i \in CR_{new}$ after the belief update process, meaning that there is an argument $A$ for $p_i$ in $Argsp$, which uses rule $r_i$ and applicable rules to support $p_i$, and has a rank $\Sigma_{C_i}(p_i)$ higher than that of $\sim p_i$ ($\Sigma_{C_i}(\sim p_i)$). Consequently for every argument $B$ for $\sim p_i$ using $s_i$ in $Argsp$ will be blocked and does not defeat $A$ at $p_i$. Suppose that an argument $B$ for $\sim p_i$ in $Argsp$ uses a rules $s_i$ that is not blocked. Then by inductive hypothesis, for some literal $b$ in $B$, it holds that $b$ is rejected; hence $B$ is undercut by $JArgs^P$.

So, overall speaking, using the justified argumentation line for foreign literals in the body of $r_i$, we can obtain an argumentation for $p_i$, which is supported by $JArgs^P$, and every argument defeating $A$ is either undercut by $JArgs^P$ or blocked by the preference operator w.r.t. the rank $\Sigma_{C_i}(p_i)$ of $p_i$. That is, $p_i$ is justified in $P$.

(Case 2) For the inference procedure returns $Ans_{p_i} = false$ in $n + 1$ calls, the following conditions hold: (a) there is no strict local argument for $p_i$ in $Argsp$; and (b) for every rules $r_i$ with head $p_i$ in $C_i$ either (b1) there is a literal $a$ in the body of $r_i$ s.t. the inference procedure return $Ans_a = false$ in at most $n$ calls. By inductive hypothesis, $a$ is rejected. Every argument $A$ using $r_i$ is defeated by an argument supported by $JArgs^P$ if $a \in V_i$, or every argumentation line $AL_a$ with head $a$ is defeated by an argument supported by $JArgs^P$ if otherwise. In any of the two cases, the arguments using $r_i$ is rejected by $JArgs^P$; or (b2) there is a rule $s_i$ with head $\sim p_i$ in $C_i$ s.t. the inference procedure returns $Ans_{s,p_i} = true$ for any literal $b$ in its body, and for every literal $a$ in the body of $r_i$, the inference procedure returns $Ans_a = true$ or undefined as the answer for $a$ in at most $n$ calls, and the rank $\Sigma_{C_i}(p_i)$ not greater than $\Sigma_{C_i}(\sim p_i)$. Then by definition of belief state and inductive hypothesis, we can build an argument $B$ for $\sim p_s t. B$ is supported by $JArgs^P$ and has lower or equal rank than any argument $A$ for $p_i$ that uses unblocked rules and rule $r_i$; therefore $B$ is defeated by such argument for $p_i$.

Consider now the arguments for $p_i$ in $Argsp$ that use at least one rules that are not unblocked. As above, we can prove that these arguments are defeated by an argument supported by $JArgs^P$.

Therefore, for every argument $A$ for $p_i$ it holds that either $A$ or an argument in every argumentation line with head $A$ is defeated by $JArgs^P$; therefore $p_i$ is rejected by $JArgs^P$.

(Case 4) For the inference procedure returns $Ans_{p_i} = undefined$ in $n + 1$ calls, the following conditions must hold:

(a) there exists no local argument for $p_i$ or $\sim p_i$ in $Argsp$.

(b) $p_i \notin \Delta_i$ and $\sim p_i \notin \Delta_i$.

(c) for all rules with head $p_i$ in $P$ either (c1) there is a literal $a \in A(r_i)$ s.t. the inference procedure returns $Ans_a = undefined$ as an answer for $a$ in no more than $n$ calls; or (c2) $\forall a \in A(r_i)$ the inference procedure returns $Ans_a = true$ as an answer for $a$ in no
more than n calls, but there is rule $s_i \in C_i$ with head $\neg p_i$ and for every literals $b \in A(s_i)$ s.t. the inference procedure returns either $A_{sb} = true$ or undefined as an answer in no more than n calls and $\Sigma_{C_i}(p_i) = \Sigma_{C_i}(\neg p_i)$. For case described in (c1), using inductive hypothesis, $a$ is not justified. Consequently there is no argument for $p_i$ in $Args_P$ that is supported by $J_{Args_P}$.

For case described in (c2), by inductive hypothesis, for every arguments $A$ in $Args_P$ that uses $r_i$ and other applicable rules in $C_i$, there is an argument $B$ in $Args_P$ that uses $s_i$ and other unblocked rules in $C_i$ s.t. $\Sigma_{C_i}(\neg p_i) \geq \Sigma_{C_i}(p_i)$ holds, meaning that $B$ defeats $A$ at $p_i$. Consequently it can be proved that there is an argument $B'$ in $Args_P$, which also uses $s_i$, and has lower rank than $B$ w.r.t. $C_i$, and an argumentation line $AL_{B'}$ with head $B'$ s.t. no subarguments of $B$ or arguments of $AL_{B'}$ is defeated by an argument supported by $J_{Args_P}$. Therefore $B'$ is not undercut by $J_{Args_P}$ and defeats any non-strict arguments with applicable rules with $p_i$ in their head. Since there is no strict local argument for $p_i$ and for all arguments for $p_i$ there is no applicable rules w.r.t. $P$, it is easy to verify that they are not supported by $J_{Args_P}$. Thus we reach a conclusion that, for every arguments $A$ for $p_i$ in $Args_P$, $A$ is either not supported by $J_{Args_P}$, or are attacked by an argument in $Args_P$ which is not undercut by $J_{Args_P}$. Therefore $p_i$ is not justified.

(d) for all rules $s_i \in C_i$ with head $\neg p_i$ either (d1) there is a literal $b \in A(s_i)$ s.t. the inference procedure either returns $A_{sb} = false$ or undefined in no more than n calls; or (d2) for all $b$, the inference procedure returns $A_{sb} = true$ in no more than n calls but there is a rule $r_i \in C_i$ with head $p_i$ s.t. $\forall a \in A(r_i)$ the inference procedure returns either $A_{sa} = true$ or undefined in no more than n calls, and $\Sigma_{C_i}(p_i) > \Sigma_{C_i}(\neg p_i)$. In the same way as before, we can reach to a conclusion that there is an argument $A$ in $Args_P$, which uses rule $r_i$ and other applicable rules, and an argumentation line $AL_A$ with head $A$ s.t. there is no argument $B$ that is supported by $J_{Args_P}$ and defeats an argument in $AL_A$. Therefore $p_i$ is not rejected by $J_{Args_P}$.

($\Leftarrow$)

(Case 1) We use induction on the stage of acceptability with conclusions $p_i$ in $Args_P$.

Induction base. By definition, an argument $A$ for $p_i$ is acceptable if (a) $A$ is a strict local argument, i.e., $A_{spar} = true$; or (b) $A$ is a defeasible local argument in $Args_P$ that is supported by $J_{0}^{P}$ and every argument that defeat $A$ in $Args_P$ is undercut by $J_{0}^{P}$. Since $A$ is supported by $J_{0}^{P}$, there is at least one rule $r_i \in C_i$ with head $p_i$ and with empty body. Suppose on the contrary that there is another rule $s_i$ with head $\neg p_i$ in $C_i$ s.t. for all literals $b$ in its body, the inference procedure returns $A_{sb} = true$ or undefined and the rank $\Sigma_{C_i}(\neg p_i) > \Sigma_{C_i}(p_i)$. Then, for all arguments for $p_i$ and rule $r_i$, there is an argument $B'$ that uses rule $s_i$ and unblocked rules, which has higher rank than $A$ in $C_i$. However, since $A$ is a local argument in $C_i$, it is not possible. Therefore, for every rule $s_i$ with head $p_i$ in $C_i$, either (c) there is a literal $b$ in the body of $s_i$ s.t. the inference procedure returns $A_{sb} = false$ or (d) $\Sigma_{C_i}(p_i) > \Sigma_{C_i}(\neg p_i)$, which will eventually return true as an answer for $p_i$.

Inductive step. Suppose that $A$ is an argument for $p_i$ in $Args_P$ that is acceptable w.r.t. $J_{n+1}^{P}$. Again, by definition, (a) $A$ is a strict local argument for $p_i$, i.e., $A_{spar} = true$; or (b) $A$ is a defeasible local argument in $Args_P$ that is supported by $J_{n+1}^{P}$ and for every argument defeating $A$ is undercut by an argument in $J_{n+1}^{P}$. That $A$ is supported by $J_{n+1}^{P}$ meaning that every proper subarguments of $A$ is acceptable w.r.t. to $J_{n}^{P}$ and there is an argumentation line $AL_A$ with head $A$ s.t. every arguments in $AL_A$ is acceptable w.r.t. $J_{n}^{P}$.
That is, by inductive hypothesis, there exists a rule $r_i$ in $C_i$ with head $p_i$ s.t. for every literal $a \in A(r_i)$ the inference procedure returns $\text{Ans}_a = \text{true}$.

Suppose that there exists an argument $B$ in $\text{Args}_P$ defeats $A$. Then by definition, $B$ is undercut by $J^P_{n+1}$. That is, for every argumentation line $AL_B$ with head $B$, there exists an argument $D$ in $\text{Args}_P$ supported by $J^P_n$ s.t. $D$ defeats a proper subarguments of $B$ or an argument in $AL_B - B$ at $q_j$. So, by inductive hypothesis, either (a) there exists a rule $t_j \in C_i$ with head $q_j$ s.t. $\forall t \in A(t_j)$ the inference procedure returns $\text{Ans}_t = \text{true}$. Then it holds that $\Sigma_{C_i}(D) > \Sigma_{C_i}(B')$ where $B'$ is either a proper subarguments of $B$ or an argument in the argumentation line $AL_B$; or (b) for all rules $s_j \in C_i$ with head $q_j$, $\exists s \in A(s_j)$ s.t. $\text{Ans}_s = \text{false}$.

Consider now a rule $s_i \in C_i$ with head $\sim p_i$, which is contained as an argument in $\text{Args}_P$ and does not defeat $A$, and suppose that $\forall b \in A(s_i)$, the inference procedure returns either $\text{Ans}_b = \text{true}$ or undefined, and $\Sigma_{C_i}(p_i) \neq \Sigma_{C_i}(\sim p_i)$. Then, we can verify by the first part of the proposition that there is an argument $B$ and an argumentation line $AL_B$ with head $B$ s.t. no argument in $AL_B$ is defeated by an argument supported by $J\text{Args}_P$, and for every arguments using any unblocked rules and $r_i$ to support $p_i$, $\Sigma_{C_i}(\sim p_i) \neq \Sigma_{C_i}(p_i)$. That is, $p_i$ is not justified, which contradicts our original supposition. Therefore, $\forall s_i \in C_i$ with head $\sim p_i$, which is contained as an argument that $B$ does not defeat $A$, either $\exists b \in A(s_i)$ s.t. the inference procedure returns $\text{Ans}_b = \text{false}$ or $\Sigma_{C_i}(\sim p_i) \neq \Sigma_{C_i}(p_i)$.

Overall, an argument $A$ for $p_i$ that is acceptable w.r.t. $J^P_{n+1}$ if there is a rule $r_i$ with head $p_i \in C_i$ s.t. $\forall a \in A(r_i)$ the inference process returns $\text{Ans}_a = \text{true}$, and for every rule $s_i$ with head $\sim p_i$ either (a) $\exists s \in A(s_i)$ s.t. the inference process returns $\text{Ans}_s = \text{false}$; or (b) $\Sigma_{C_i}(p_i) > \Sigma_{C_i}(\sim p_i)$ In either case, the inference procedure will compute $\text{Ans}_{p_i} = \text{true}$ and $\text{Ans}_{\sim p_i} = \text{false}$ or undefined, and will eventually return $\text{true}$ as an answer for $p_i$.

**Case 2** Assume on the contrary that a literal $p_i$ is rejected by $J\text{Args}_P$ and the inference procedure returns either $\text{true}$ or undefined as an answer in $n$ calls. By definition, either (a) there is a strict local argument for $p_i$ in $\text{Args}_P$, which leads to the conclusion that $p_i$ is justified (Contradiction), or (b) there is a rule $r_i$ with head $p_i \in C_i$ s.t. $\forall a \in A(r_i)$ the inference procedure returns either $\text{true}$ or undefined as an answer for $a$, and for all rules $s_i$ with head $\sim p_i$ either (a) $\exists s \in A(s_i)$ s.t. the inference process returns $\text{Ans}_s = \text{false}$ or (b) $\Sigma_{C_i}(p_i) > \Sigma_{C_i}(\sim p_i)$ or is blocked due to the rank in the agent’s belief state. Then, by the first part of the proposition, this implies that there is an argument $A$ for $p_i$ in $\text{Args}_P$ and an argumentation line $AL_A$ with head $A$ s.t. no argument in $AL_A - A$ and no proper subarguments of $A$ is defeated by an argument supported by $J\text{Args}_P$. And for every argument $B$ for $\sim p_i$ in $\text{Args}_P$, either $B$ is not supported by $J\text{Args}_P$, or there is argument $D$ for $p_i$ in $\text{Args}_P$ and argumentation line $AL_D$ with head $D$ s.t. no proper subarguments of $D$ is defeated by an argument supported by $\text{Args}_P$ and $\Sigma_{C_i}(p_i) > \Sigma_{C_i}(\sim p_i)$. However, this leads to a conclusion that $p_i$ is not rejected, which contradicts our original proposition. Therefore, the inference procedure will return $\text{Ans}_{p_i} = \text{false}$ as an answer for $p_i$.

**Case 4** It is trivial to prove using the first part of the theorem. For a literal $p_i$ which is neither justified or rejected by $J\text{Args}_P$, suppose that $\text{Ans}_{p_i} = \text{true}$, then $p_i$ is justified by $J\text{Args}_P$ (contradiction). Suppose on the contrary that $\text{Ans}_{p_i} = \text{false}$, then $p_i$ is rejected by $J\text{Args}_P$ (contradiction). Therefore, for $p_i$ the inference procedure will return $\text{Ans}_{p_i} = \text{undefined}$. □
Appendix C

SPINdle De defeasible Theory Language

SPINdle uses a simple language (called DFL) in which literals, propositions and their negations can be represented in textual form. The following is a summary description the syntax accepted by the current implementation.

C.1 Whitespace and Comments

Any amount of whitespace is permitted before and after any symbol. All text after the “#” sign are considered as comments and will be treated as whitespace.

C.2 Atoms and Literals

Atom

An atom is a predicate symbol (the name of the atom) that is optionally followed by a parenthesized list of terms.

Sample atoms: foo(X), a, abc, foo

Basic Literal

A basic literal is either an atom a or its negation \( \neg a \).

The classical negation \( \neg a \) of an atom \( a \) is denoted by a minus sign (\( - \)) that is immediately before the atom name, i.e.: \( \neg a \).

The latest version of SPINdle extends defeasible theories by permitting arguments in literals, where arguments can be any text string (constant) and are enclosed in parentheses separated by a comma. For example: \( p(a, b, c) \).

Modalised Literal

A modalised literal is of the form

\( [\Box]a \)

where \( \Box \) is the modal operator representing mental states of the literal.

Sample literals: \( [INT]foo(X) \), \( [OBL]\neg smoke \), \( [PER]leave(X) \)
### C.3 Facts, Rules and Defeaters

The following is the basic form of rule in DFL:

\[ Rx[\Box] : a_1, a_2, \ldots, a_i, -b_1, -b_2, \ldots, -b_j \triangleright c_1, c_2, \ldots, c_k \]

where \( Rx \) is the rule label, \( \Box \) is the modal operator of the rule, \( a_1, a_2, \ldots, a_i, -b_1, -b_2, \ldots, -b_j \) and \( c_1, c_2, \ldots, c_k \) are the body and head (modalised) literals of the rule respectively, and \( \triangleright \) is the rule type symbol (Table C.1).

A rule label will be generated by the theory parser if the rule label is missing in the original content. While a rule can have no body literals, there should be at least one literal in the head, or an exception will be thrown while parsing the theory.

Besides, it is also important to note that both facts, strict rules and defeaters can have only **ONE** head literal while defeasible rules is the only rule type that can have multiple literals in the head. In addition, note also that there should be **NO** body literals for facts. If a fact can only be derived under some constraints, it should therefore be represented using a strict rule instead.

### C.4 Superiority relations

A superiority relation is represented by putting a “\( \triangleright \)” symbol between two rule labels. For example: \( r1 \triangleright r2 \), where \( r1 \) and \( r2 \) are the rule labels of the two rules with conflicting (head) literal(s).

### C.5 Conclusions

After running SPINdle, various conclusions of a defeasible theory \( T \) can be printed on the screen and/or stored in the local file system. A conclusion of \( T \) is a tagged literal and can have one of the following four forms:

+\( Dq \) This means that \( q \) is definitely provable in \( T \). That is, it is a fact, or can be proved using only strict rules.

\( -Dq \) This means that we have proved that \( q \) is not definitely provable in \( T \).

+\( dq \) This means that \( q \) is defeasibly provable in \( T \).

\( -dq \) This means that we have proved that \( q \) is not defeasibly provable in \( T \).
An example

Example C.1. Consider the following defeasible theory $D$, written in DFL.

\begin{align*}
r1: & \quad \rightarrow quaker \\
r2: & \quad \rightarrow \neg \text{republican} \\
r3: & \quad \text{quaker} \rightarrow \text{pacifist} \\
r4: & \quad \text{republican} \rightarrow \neg \text{pacifist} \\
r5: & \quad \text{republican} \rightarrow \text{footballfan} \\
r6: & \quad \text{pacifist} \rightarrow \text{antimilitary} \\
r7: & \quad \text{footballfan} \rightarrow \neg \text{antimilitary}
\end{align*}

The superiority relation is empty

and the followings are the conclusions derived with ambiguity blocking:

\begin{align*}
+ D & \quad \text{quaker}(X) \\
+ D & \quad \text{republican}(X) \\
- D & \quad \text{antimilitary}(X) \\
- D & \quad \neg \text{antimilitary}(X) \\
- D & \quad \text{footballfan}(X) \\
- D & \quad \text{pacifist}(X) \\
- D & \quad \neg \text{pacifist}(X) \\
+ d & \quad \neg \text{antimilitary}(X) \\
+ d & \quad \text{footballfan}(X) \\
+ d & \quad \text{quaker}(X) \\
+ d & \quad \text{republican}(X) \\
- d & \quad \text{antimilitary}(X) \\
- d & \quad \text{pacifist}(X) \\
- d & \quad \neg \text{pacifist}(X)
\end{align*}
Appendix D

XML schema for SPINdle defeasible theory

<?xml version="1.0" encoding="UTF-8"?>
  <xsd:element name="theory" type="ctTheory"></xsd:element>
  <xsd:complexType name="ctTheory">
    <xsd:sequence>
      <xsd:element name="fact" type="ctFact" maxOccurs="unbounded" minOccurs="0"></xsd:element>
      <xsd:element name="rule" type="ctRule" maxOccurs="unbounded" minOccurs="0"></xsd:element>
      <xsd:element name="superiority" type="ctSuperiority" maxOccurs="unbounded" minOccurs="0"></xsd:element>
      <xsd:element name="coversion" type="ctConversion" maxOccurs="unbounded" minOccurs="0"></xsd:element>
      <xsd:element name="conflict" type="ctConflict" maxOccurs="unbounded" minOccurs="0"></xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="ctLiteral">
    <xsd:sequence>
      <xsd:choice maxOccurs="1" minOccurs="0">
        <xsd:element name="mode" type="stMode"></xsd:element>
        <xsd:element name="not" type="ctNegMode"></xsd:element>
      </xsd:choice>
      <xsd:choice maxOccurs="1" minOccurs="0">
        <xsd:element name="atom" type="stAtom"></xsd:element>
        <xsd:element name="not" type="ctAtom"></xsd:element>
      </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="mode" type="xsd:string"></xsd:attribute>
  </xsd:complexType>
  <xsd:complexType name="ctLiteralList">
    <xsd:choice>
      <xsd:element name="literal" type="ctLiteral" minOccurs="1" maxOccurs="1"></xsd:element>
      <xsd:element name="literal" type="ctLiteral" minOccurs="1" maxOccurs="1"></xsd:element>
    </xsd:choice>
  </xsd:complexType>
</xsd:schema>
<xsd:element name="and" type="ctAndLiterals" maxOccurs="1" minOccurs="1"></xsd:element>
</xsd:choice>
</xsd:complexType>
<xsd:complexType name="ctRule">
  <xsd:sequence>
    <xsd:element name="head" type="ctLiteralList" maxOccurs="1"
      minOccurs="1"></xsd:element>
    <xsd:element name="body" type="ctLiteralList"
      maxOccurs="1" minOccurs="0"></xsd:element>
    <xsd:choice maxOccurs="1" minOccurs="0">
      <xsd:element name="mode" type="stMode"></xsd:element>
      <xsd:element name="not" type="ctNegMode"></xsd:element>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="label" type="xsd:string"></xsd:attribute>
  <xsd:attribute name="strength" type="xsd:string"></xsd:attribute>
</xsd:complexType>
<xsd:complexType name="ctAndLiterals">
  <xsd:sequence>
    <xsd:element name="literal" type="ctLiteral" maxOccurs="unbounded"
      minOccurs="1"></xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ctConversion">
  <xsd:sequence>
    <xsd:element name="from" type="xsd:string" maxOccurs="1"
      minOccurs="1"></xsd:element>
    <xsd:element name="to" type="xsd:string" maxOccurs="unbounded"
      minOccurs="1"></xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ctConflict">
  <xsd:sequence>
    <xsd:element name="conflictMode" type="xsd:string" maxOccurs="1"
      minOccurs="1"></xsd:element>
    <xsd:element name="conflictWith" type="xsd:string"
      maxOccurs="unbounded" minOccurs="1"></xsd:element>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="ctSuperiority">
  <xsd:attribute name="superior" type="xsd:string"></xsd:attribute>
  <xsd:attribute name="inferior" type="xsd:string"></xsd:attribute>
</xsd:complexType>
<xsd:complexType name="ctFact">
  <xsd:sequence>
    <xsd:element name="literal" type="ctLiteral"></xsd:element>
  </xsd:sequence>
  <xsd:attribute name="label" type="xsd:string"></xsd:attribute>
  <xsd:attribute name="mode" type="xsd:string"></xsd:attribute>
</xsd:complexType>

XVII
<xsd:simpleType name="stMode">
  <xsd:restriction base="xsd:string"></xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="ctNegMode">
  <xsd:sequence>
    <xsd:element name="mode" type="stMode"></xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="stAtom">
  <xsd:restriction base="xsd:string"></xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="ctAtom">
  <xsd:sequence>
    <xsd:element name="atom" type="stAtom" maxOccurs="1" minOccurs="1"></xsd:element>
  </xsd:sequence>
</xsd:complexType>
</xsd:/schema>
Appendix E

Experimental results

<table>
<thead>
<tr>
<th>$n$</th>
<th>$chain(n)$</th>
<th>$circle(n)$</th>
<th>$dag(n,k)$</th>
<th>$tree(n,k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPINdle</td>
<td>Delores</td>
<td>SPINdle</td>
<td>Delores</td>
</tr>
<tr>
<td>500</td>
<td>0.144</td>
<td>0.029</td>
<td>0.064</td>
<td>0.029</td>
</tr>
<tr>
<td>1000</td>
<td>0.243</td>
<td>0.047</td>
<td>0.127</td>
<td>0.048</td>
</tr>
<tr>
<td>1500</td>
<td>0.347</td>
<td>0.063</td>
<td>0.163</td>
<td>0.064</td>
</tr>
<tr>
<td>2000</td>
<td>0.402</td>
<td>0.081</td>
<td>0.192</td>
<td>0.084</td>
</tr>
<tr>
<td>2500</td>
<td>0.623</td>
<td>0.105</td>
<td>0.377</td>
<td>0.104</td>
</tr>
<tr>
<td>3000</td>
<td>0.648</td>
<td>0.123</td>
<td>0.382</td>
<td>0.123</td>
</tr>
<tr>
<td>3500</td>
<td>0.729</td>
<td>0.141</td>
<td>0.432</td>
<td>0.140</td>
</tr>
<tr>
<td>4000</td>
<td>0.754</td>
<td>0.160</td>
<td>0.450</td>
<td>0.161</td>
</tr>
<tr>
<td>4500</td>
<td>0.819</td>
<td>0.180</td>
<td>0.493</td>
<td>0.180</td>
</tr>
<tr>
<td>5000</td>
<td>0.890</td>
<td>0.200</td>
<td>0.574</td>
<td>0.201</td>
</tr>
<tr>
<td>5500</td>
<td>0.941</td>
<td>0.220</td>
<td>0.563</td>
<td>0.220</td>
</tr>
<tr>
<td>6000</td>
<td>0.975</td>
<td>0.240</td>
<td>0.566</td>
<td>0.240</td>
</tr>
<tr>
<td>6500</td>
<td>1.062</td>
<td>0.260</td>
<td>0.593</td>
<td>0.260</td>
</tr>
<tr>
<td>7000</td>
<td>1.109</td>
<td>0.280</td>
<td>0.596</td>
<td>0.280</td>
</tr>
<tr>
<td>7500</td>
<td>1.168</td>
<td>0.300</td>
<td>0.647</td>
<td>0.300</td>
</tr>
<tr>
<td>8000</td>
<td>1.242</td>
<td>0.320</td>
<td>0.655</td>
<td>0.320</td>
</tr>
<tr>
<td>8500</td>
<td>1.351</td>
<td>0.340</td>
<td>0.658</td>
<td>0.340</td>
</tr>
<tr>
<td>9000</td>
<td>1.499</td>
<td>0.360</td>
<td>0.665</td>
<td>0.360</td>
</tr>
<tr>
<td>9500</td>
<td>1.569</td>
<td>0.380</td>
<td>0.721</td>
<td>0.380</td>
</tr>
<tr>
<td>10000</td>
<td>1.648</td>
<td>0.400</td>
<td>0.781</td>
<td>0.400</td>
</tr>
</tbody>
</table>

Table E.1: Reasoning time for Theories with Undisputed Inferences (with $k = 3$)
<table>
<thead>
<tr>
<th>$n$</th>
<th>$\text{teams}(n)$</th>
<th>$\text{supTest}(n)$</th>
<th>$n$</th>
<th>$\text{SPIndle}$</th>
<th>$\text{Delores}$</th>
<th>$n$</th>
<th>$\text{SPIndle}$</th>
<th>$\text{Delores}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.017</td>
<td>0.010</td>
<td>500</td>
<td>0.257</td>
<td>0.121</td>
<td>500</td>
<td>0.257</td>
<td>0.121</td>
</tr>
<tr>
<td>2</td>
<td>0.036</td>
<td>0.015</td>
<td>1000</td>
<td>0.611</td>
<td>0.250</td>
<td>1000</td>
<td>0.611</td>
<td>0.250</td>
</tr>
<tr>
<td>3</td>
<td>0.084</td>
<td>0.040</td>
<td>1500</td>
<td>0.782</td>
<td>0.380</td>
<td>1500</td>
<td>0.782</td>
<td>0.380</td>
</tr>
<tr>
<td>4</td>
<td>0.232</td>
<td>0.170</td>
<td>2000</td>
<td>0.864</td>
<td>0.515</td>
<td>2000</td>
<td>0.864</td>
<td>0.515</td>
</tr>
<tr>
<td>5</td>
<td>0.831</td>
<td>0.762</td>
<td>2500</td>
<td>1.073</td>
<td>0.651</td>
<td>2500</td>
<td>1.073</td>
<td>0.651</td>
</tr>
<tr>
<td>6</td>
<td>2.777</td>
<td>4.347</td>
<td>3000</td>
<td>1.173</td>
<td>0.797</td>
<td>3000</td>
<td>1.173</td>
<td>0.797</td>
</tr>
<tr>
<td>7</td>
<td>24.601</td>
<td>81.311</td>
<td>3500</td>
<td>1.265</td>
<td>0.942</td>
<td>3500</td>
<td>1.265</td>
<td>0.942</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4000</td>
<td>1.384</td>
<td>1.089</td>
<td>4000</td>
<td>1.384</td>
<td>1.089</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4500</td>
<td>1.669</td>
<td>1.241</td>
<td>4500</td>
<td>1.669</td>
<td>1.241</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5000</td>
<td>1.702</td>
<td>1.399</td>
<td>5000</td>
<td>1.702</td>
<td>1.399</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5500</td>
<td>1.809</td>
<td>1.557</td>
<td>5500</td>
<td>1.809</td>
<td>1.557</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6000</td>
<td>1.867</td>
<td>1.723</td>
<td>6000</td>
<td>1.867</td>
<td>1.723</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6500</td>
<td>2.154</td>
<td>1.889</td>
<td>6500</td>
<td>2.154</td>
<td>1.889</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7000</td>
<td>2.279</td>
<td>2.060</td>
<td>7000</td>
<td>2.279</td>
<td>2.060</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7500</td>
<td>2.731</td>
<td>2.233</td>
<td>7500</td>
<td>2.731</td>
<td>2.233</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8000</td>
<td>2.800</td>
<td>2.415</td>
<td>8000</td>
<td>2.800</td>
<td>2.415</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8500</td>
<td>2.788</td>
<td>2.597</td>
<td>8500</td>
<td>2.788</td>
<td>2.597</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9000</td>
<td>2.853</td>
<td>2.786</td>
<td>9000</td>
<td>2.853</td>
<td>2.786</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9500</td>
<td>2.943</td>
<td>2.976</td>
<td>9500</td>
<td>2.943</td>
<td>2.976</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10000</td>
<td>3.222</td>
<td>3.199</td>
<td>10000</td>
<td>3.222</td>
<td>3.199</td>
</tr>
</tbody>
</table>

Table E.2: Reasoning time for Theories with Disputed Inferences
Appendix F

UAV Navigation Theory

# uav simulator - theory

# rules about the vehicle
# [contextRule_Rule_00001] >> isEmergencyVehicle
# [contextRule_Rule_00002] >> isExpensiveVehicle
# [contextRule_Rule_00003] >> currentMove_EAST(X)

# contextual rules in each direction
# [contextRule_Rule_00004] -> hasVehicle_EAST(X)
# [contextRule_Rule_00005] -> unmovableVehicle_EAST(X)
# [contextRule_Rule_00006] -> trafficJam_EAST(X)
# [contextRule_Rule_00007] -> possibleCollision_EAST(X)
# [contextRule_Rule_00008] -> emergencyVehicleComing_EAST(X)
# [contextRule_Rule_00009] -> expensiveVehicleComing_EAST(X)
# [contextRule_Rule_00010] -> -directPath_EAST(X)
# [contextRule_Rule_00011] -> -reversePath_EAST(X)

# result literals
# moveTo_EAST, moveTo_SOUTH,
# moveTo_WEST, moveTo_NORTH,
# moveReverseTo_EAST, moveReverseTo_SOUTH,
# moveReverseTo_WEST, moveReverseTo_NORTH

# vehicle and context setting

# VEH01: =>-withPriority
# VEH11: isEmergencyVehicle => withPriority
# VEH11>VEH01

# give way to emergency vehicle if the vehicle itself is not an emergency vehicle
# if both vehicles are emergency vehicle, negotiation between vehicles is needed
GW01: => -giveWay_EAST
GW02: => -giveWay_SOUTH
GW03: => -giveWay_WEST
GW04: => -giveWay_NORTH
EM01: emergencyVehicleComing_EAST => giveWay_EAST, negotiatePathWithEmergencyVehicle_EAST
EM02: emergencyVehicleComing_SOUTH => giveWay_SOUTH, negotiatePathWithEmergencyVehicle_SOUTH
EM03: emergencyVehicleComing_WEST => giveWay_WEST, negotiatePathWithEmergencyVehicle_WEST
EM04: emergencyVehicleComing_NORTH => giveWay_NORTH, negotiatePathWithEmergencyVehicle_NORTH
PRI01: withPriority => -giveWay_EAST
PRI02: withPriority => -giveWay_SOUTH
PRI03: withPriority => -giveWay_WEST
PRI04: withPriority => -giveWay_NORTH
# assume all travel directions are not busy
BD01: \(-\text{busyDirection}\_\text{EAST}\)
BD02: \(-\text{busyDirection}\_\text{SOUTH}\)
BD03: \(-\text{busyDirection}\_\text{WEST}\)
BD04: \(-\text{busyDirection}\_\text{NORTH}\)
TJ11: \(\text{trafficJam}\_\text{EAST} \Rightarrow \text{busyDirection}\_\text{EAST}\)
TJ12: \(\text{trafficJam}\_\text{SOUTH} \Rightarrow \text{busyDirection}\_\text{SOUTH}\)
TJ13: \(\text{trafficJam}\_\text{WEST} \Rightarrow \text{busyDirection}\_\text{WEST}\)
TJ14: \(\text{trafficJam}\_\text{NORTH} \Rightarrow \text{busyDirection}\_\text{NORTH}\)
TJ11>BD01
TJ12>BD02
TJ13>BD03
TJ14>BD04

# assume that there is no vehicle coming in all directions
# if there exist vehicle(s) coming, assuming this direction is traffic jammed
VC01: \(-\text{vehicleComing}\_\text{EAST}\)
VC02: \(-\text{vehicleComing}\_\text{SOUTH}\)
VC03: \(-\text{vehicleComing}\_\text{WEST}\)
VC04: \(-\text{vehicleComing}\_\text{NORTH}\)
HV01: \(\text{hasVehicle}\_\text{EAST} \Rightarrow \text{vehicleComing}\_\text{EAST}\)
HV02: \(\text{hasVehicle}\_\text{SOUTH} \Rightarrow \text{vehicleComing}\_\text{SOUTH}\)
HV03: \(\text{hasVehicle}\_\text{WEST} \Rightarrow \text{vehicleComing}\_\text{WEST}\)
HV04: \(\text{hasVehicle}\_\text{NORTH} \Rightarrow \text{vehicleComing}\_\text{NORTH}\)
vehicleComing\_\text{EAST} \Rightarrow \text{trafficJam}\_\text{EAST}
vehicleComing\_\text{SOUTH} \Rightarrow \text{trafficJam}\_\text{SOUTH}
vehicleComing\_\text{WEST} \Rightarrow \text{trafficJam}\_\text{WEST}
vehicleComing\_\text{NORTH} \Rightarrow \text{trafficJam}\_\text{NORTH}
HV01>VC01
HV02>VC02
HV03>VC03
HV04>VC04

# assumed that there exist direct/reverse paths available in all directions
PATH01: \(\Rightarrow \text{pathTo}\_\text{EAST}\)
PATH02: \(\Rightarrow \text{pathTo}\_\text{SOUTH}\)
PATH03: \(\Rightarrow \text{pathTo}\_\text{WEST}\)
PATH04: \(\Rightarrow \text{pathTo}\_\text{NORTH}\)
REV\_PATH01: \(\Rightarrow \text{reversePathTo}\_\text{EAST}\)
REV\_PATH02: \(\Rightarrow \text{reversePathTo}\_\text{SOUTH}\)
REV\_PATH03: \(\Rightarrow \text{reversePathTo}\_\text{WEST}\)
REV\_PATH04: \(\Rightarrow \text{reversePathTo}\_\text{NORTH}\)

# override the rule if no direct/reverse path exist at a particular direction
PATH21: \(-\text{directPath}\_\text{EAST} \Rightarrow -\text{pathTo}\_\text{EAST}\)
PATH22: \(-\text{directPath}\_\text{SOUTH} \Rightarrow -\text{pathTo}\_\text{SOUTH}\)
PATH23: \(-\text{directPath}\_\text{WEST} \Rightarrow -\text{pathTo}\_\text{WEST}\)
PATH24: \(-\text{directPath}\_\text{NORTH} \Rightarrow -\text{pathTo}\_\text{NORTH}\)
PATH21>PATH01
PATH22>PATH02

XXII
REV_PATH21: -reversePath_EAST => -reversePathTo_EAST
REV_PATH22: -reversePath_SOUTH => -reversePathTo_SOUTH
REV_PATH23: -reversePath_WEST => -reversePathTo_WEST
REV_PATH24: -reversePath_NORTH => -reversePathTo_NORTH
REV_PATH21>REV_PATH01
REV_PATH22>REV_PATH02
REV_PATH23>REV_PATH03
REV_PATH24>REV_PATH04

FFT01: pathTo_EAST, -giveWay_EAST => canTravelTo_EAST
FFT02: pathTo_SOUTH, -giveWay_SOUTH => canTravelTo_SOUTH
FFT03: pathTo_WEST, -giveWay_WEST => canTravelTo_WEST
FFT04: pathTo_NORTH, -giveWay_NORTH => canTravelTo_NORTH

FFT11: reversePathTo_EAST, -giveWay_EAST => canTravelReverseTo_EAST
FFT12: reversePathTo_SOUTH, -giveWay_SOUTH => canTravelReverseTo_SOUTH
FFT13: reversePathTo_WEST, -giveWay_WEST => canTravelReverseTo_WEST
FFT14: reversePathTo_NORTH, -giveWay_NORTH => canTravelReverseTo_NORTH

UMV21: unmovableVehicle_EAST => -canTravelTo_EAST
UMV22: unmovableVehicle_SOUTH => -canTravelTo_SOUTH
UMV23: unmovableVehicle_WEST => -canTravelTo_WEST
UMV24: unmovableVehicle_NORTH => -canTravelTo_NORTH
UMV21>FFT01
UMV22>FFT02
UMV23>FFT03
UMV24>FFT04

PC21: possibleCollision_EAST => -canTravelTo_EAST
PC22: possibleCollision_SOUTH => -canTravelTo_SOUTH
PC23: possibleCollision_WEST => -canTravelTo_WEST
PC24: possibleCollision_NORTH => -canTravelTo_NORTH
PC21>FFT01
PC22>FFT02
PC23>FFT03
PC24>FFT04

UMV31: unmovableVehicle_EAST => -canTravelReverseTo_EAST
UMV32: unmovableVehicle_SOUTH => -canTravelReverseTo_SOUTH
UMV33: unmovableVehicle_WEST => -canTravelReverseTo_WEST
UMV34: unmovableVehicle_NORTH => -canTravelReverseTo_NORTH
UMV31>FFT11
UMV32>FFT12
UMV33>FFT13
UMV34>FFT14

PC31: possibleCollision_EAST => -canTravelReverseTo_EAST
PC32: possibleCollision_SOUTH => -canTravelReverseTo_SOUTH
PC33: possibleCollision_WEST => -canTravelReverseTo_WEST
PC34: possibleCollision_NORTH => -canTravelReverseTo_NORTH
PC31>FFT01
PC32>FFT02
PC33>FFT03
PC34>FFT04

GW11: giveWay_EAST, busyDirection_EAST => -canTravelTo_EAST
GW12: giveWay_SOUTH, busyDirection_SOUTH => -canTravelTo_SOUTH
GW13: giveWay_WEST, busyDirection_WEST => -canTravelTo_WEST

XXIII
GW14: giveWay_NORTH, busyDirection_NORTH => ¬canTravelTo_NORTH
GW21: giveWay_EAST, busyDirection_EAST => ¬canTravelReverseTo_EAST
GW22: giveWay_SOUTH, busyDirection_SOUTH => ¬canTravelReverseTo_SOUTH
GW23: giveWay_WEST, busyDirection_WEST => ¬canTravelReverseTo_WEST
GW24: giveWay_NORTH, busyDirection_NORTH => ¬canTravelReverseTo_NORTH
GW11>FFT01
GW12>FFT02
GW13>FFT03
GW14>FFT04
GW21>FFT11
GW23>FFT12
GW23>FFT13
GW24>FFT14

canTravelTo_EAST => moveTo_EAST
canTravelTo_SOUTH => moveTo_SOUTH
canTravelTo_WEST => moveTo_WEST
canTravelTo_NORTH => moveTo_NORTH

canTravelReverseTo_EAST => moveReverseTo_EAST
canTravelReverseTo_SOUTH => moveReverseTo_SOUTH
canTravelReverseTo_WEST => moveReverseTo_WEST
canTravelReverseTo_NORTH => moveReverseTo_NORTH

freeToTravel_EAST => moveTo_EAST
freeToTravel_SOUTH => moveTo_SOUTH
freeToTravel_WEST => moveTo_WEST
freeToTravel_NORTH => moveTo_NORTH

XXIV