Abstract

This dissertation proposes two novel approaches for extending AgentSpeak with qualitative uncertainty reasoning by integrating two dynamic variants of epistemic logic. These extensions address a crucial gap in the literature where qualitative approaches to uncertainty are seldom integrated into agent-oriented programming languages due to various challenges related to methodology, implementation, and computational complexity.

The extensions provide various symbolic constructs that enable the modelling and reasoning of belief uncertainty. The significance of qualitative uncertainty reasoning is illustrated through a simple Minesweeper scenario and two complex uncertainty challenges from the 2019 Multi-Agent Programming Contest: uncertain navigation and agent identification. Given the ability to express qualitative uncertainty, we equip the agent with a more robust and effective way to plan and act under uncertainty.

An in-depth evaluation of the performance and scalability of the proposed AgentSpeak extensions is provided, with a heavy focus on examining their impact on the agent’s time-sensitive reasoning cycle. The results show that these extensions provide a tractable and computationally feasible approach to extending AgentSpeak with the ability to manage the statics and dynamics of uncertainty.
Acknowledgements

This dissertation is dedicated to my love, Sandra.

A constant source of support and inspiration in every way imaginable, you have always given me the strength and determination to persevere and overcome any challenges that have arisen. I love you and will forever be grateful for the time and experiences we have and will endure together.

To Mrs. Morley, I cannot thank you enough for the support you have given to both Sandra and I. Your belief in our potential and your constant push for us to aim higher and strive for greater things will forever impact our lives. It was through the support, encouragement, and inspiration of you and Sandra that I gained the confidence and strength to pursue further education. I love you and will miss you dearly. You will forever be cherished, and your influence will continue to guide Sandra and I as we navigate the next phases of our lives.

I would also like to express my gratitude to Professor Esfandiari, whose support, guidance, and encouragement have been invaluable. Your expertise and wisdom have played a crucial role in my academic and personal growth, and I am truly grateful for the opportunity to learn from and work alongside you.

I am grateful to all who supported me during this journey, as you have helped to form the foundation of this dissertation.
Contents

Abstract i

Acknowledgements ii

List of Tables x

List of Figures xi

List of Code Listings xiii

Glossary xv

1 Introduction 1

1.1 Main Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.2 Thesis Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2 Background & State of the Art 8

2.1 Background: 2019 Multi-Agent Programming Contest (MAPC) . . . 9

2.1.1 Map Definition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.1.2 Environment: Perceptions and Actions . . . . . . . . . . . . . . 10

2.1.3 Challenges: Agent Navigation and Identification . . . . . . . 12
2.2 Background: AgentSpeak: A Logical Language for BDI Agents .......................... 13
  2.2.1 Syntax and Semantics ................................................. 14
    2.2.1.1 Beliefs and Rules ........................................... 15
    2.2.1.2 Goals, Events, and Plans .................................. 15
  2.2.2 Reasoning Cycle and Operational Semantics ............................. 17

2.3 State of the Art: Uncertainty Reasoning in AgentSpeak ....................... 21
  2.3.1 Existing Uncertainty Extensions ................................... 22
    2.3.1.1 Quantitative Extensions .................................... 22
    2.3.1.2 Qualitative Extensions ..................................... 24
  2.3.2 Extending AgentSpeak ............................................. 26
    2.3.2.1 Extending the Language .................................... 28
    2.3.2.2 Extending the Reasoning Cycle .............................. 29
    2.3.2.3 Model Creation ............................................... 30
    2.3.2.4 Model Updating .............................................. 31
    2.3.2.5 Model Querying ............................................... 32
    2.3.2.6 Computational Complexity ................................... 33
  2.3.3 Key Findings .......................................................... 34

2.4 Background: Epistemic Logic For Qualitative Reasoning About Uncertainty ................................................. 36
  2.4.1 Epistemic Logic ..................................................... 37
    2.4.1.1 Semantics .................................................... 37
  2.4.2 Public Announcement Logic (PAL) .................................. 39
  2.4.3 Dynamic Epistemic Logic (DEL) ................................... 41
  2.4.4 Computational Complexity ....................................... 43
# Methodology

3.1 Introduction and Structure ........................................ 45

3.1.1 Uncertain Navigation in AgentSpeak .......................... 46

3.1.2 Chapter Structure: An Agile Approach ......................... 50

3.2 PAL-AgentSpeak: An Extension for Monotonic Belief Change .... 51

3.2.1 Propositional vs. First-Order Epistemic Reasoners ............. 52

3.2.1.1 Propositionalization ...................................... 52

3.2.2 Model Creation ................................................ 54

3.2.2.1 Extended Semantics: Ranges .............................. 55

3.2.2.2 Extended Semantics: Range Constraints .................... 55

3.2.2.3 Generating an Epistemic Model ........................... 58

3.2.3 Model Updating ................................................. 60

3.2.3.1 Monotonic Belief Change .................................. 62

3.2.3.2 Non-Monotonic Belief Change ...................... 63

3.2.3.3 Changing Possibilities ................................... 66

3.2.3.4 Generating Belief Events ................................. 67

3.2.4 Model Querying ................................................. 69

3.2.4.1 Querying Possibilities ............................... 69

3.2.4.2 Evaluation of Standard Belief Queries .................... 70

3.2.4.3 PAL-AgentSpeak Belief Querying ..................... 71

3.2.5 Operational Semantics ........................................ 77

3.2.5.1 Model Creation ........................................... 78

3.2.5.2 Model Updating ........................................... 78

3.2.5.3 Model Querying ........................................... 80

3.2.6 Concluding Remarks .......................................... 82
3.3 DEL-AgentSpeak: An Extension for Event Uncertainty and Non-
Monotonic Change ........................................... 84

3.3.1 DEL Event Models in DEL-AgentSpeak ................. 85

3.3.2 Transition from PAL-AgentSpeak ...................... 88

3.3.3 Model Creation ........................................... 90
  3.3.3.1 Operational Semantics ............................... 91

3.3.4 Model Updating ........................................... 92
  3.3.4.1 Default DEL Event Models ......................... 92
  3.3.4.2 Custom Event Models: “On” Plans ................. 94
  3.3.4.3 Operational Semantics ............................... 96

3.3.5 Model Querying ........................................... 98
  3.3.5.1 Operational Semantics ............................... 99

3.3.6 Concluding Remarks .................................... 100

4 Implementation ............................................. 104

4.1 Extending Hintikka’s World .................................. 105
  4.1.1 Standard Model Implementation ......................... 106
    4.1.1.1 Complexity of Standard Models ................... 108
  4.1.2 Compact Model Implementation ......................... 109
    4.1.2.1 Complexity of Compact Models ................... 110
  4.1.3 Performance Benefits of Compact Models ................ 111

4.1.4 Providing an Epistemic Reasoning API ................... 112
  4.1.4.1 Model Creation ....................................... 113
  4.1.4.2 Model Updates ....................................... 114
  4.1.4.3 Model Queries ....................................... 115

4.2 Implementing PAL-AgentSpeak and DEL-AgentSpeak .......... 116
4.2.1 Obtaining Rule-Free Consequences ........................................ 116
  4.2.1.1 Standard Logical Consequences .................................... 117
  4.2.1.2 Rewriting Rule-based Logical Consequences ....................... 118
4.2.2 Model Creation ................................................................. 119
  4.2.2.1 Complexity ............................................................... 120
4.2.3 Model Updating ................................................................. 121
  4.2.3.1 Complexity ............................................................... 123
4.2.4 Model Querying ................................................................. 123
  4.2.4.1 Complexity ............................................................... 124
4.2.5 Concluding Remarks .......................................................... 125

5 Application ................................................................................. 126
  5.1 MAPC: Navigating Under Uncertainty ....................................... 127
    5.1.1 Environment ................................................................. 127
      5.1.1.1 Perceptions and Actions ........................................... 128
    5.1.2 The DEL-AgentSpeak Program ......................................... 129
      5.1.2.1 Program Execution .................................................. 129
  5.2 MAPC: Identifying Agents for Collaboration .............................. 133
    5.2.1 Environment ................................................................. 133
    5.2.2 The Standard AgentSpeak Program ................................... 135
    5.2.3 The DEL-AgentSpeak Program ......................................... 137
      5.2.3.1 Program Execution .................................................. 138
  5.3 Minesweeper ........................................................................... 140
    5.3.1 Environment ................................................................. 141
      5.3.1.1 Perceptions and Actions ........................................... 141
      5.3.1.2 Initial State ........................................................... 142
5.3.2 The PAL-AgentSpeak Program .................................. 142
5.3.2.1 Program Execution ....................................... 144

6 Evaluation .......................................................... 148

6.1 Introduction and Structure ........................................ 148
  6.1.1 Introductory Remarks ....................................... 149
6.2 Model Creation .................................................... 150
  6.2.1 Evaluation Methodology ..................................... 151
    6.2.1.1 Parameters ............................................. 151
    6.2.1.2 Metrics ............................................... 151
    6.2.1.3 Methodology .......................................... 152
  6.2.2 Model Creation Results ..................................... 152
6.3 Model Updating ................................................... 154
  6.3.1 Evaluation Methodology ..................................... 154
    6.3.1.1 Parameters ............................................. 155
    6.3.1.2 Metrics ............................................... 155
    6.3.1.3 Methodology .......................................... 155
  6.3.2 Model Updating Results ..................................... 156
6.4 Model Querying ................................................... 157
  6.4.1 Evaluation Methodology ..................................... 158
    6.4.1.1 Parameters ............................................. 158
    6.4.1.2 Metrics ............................................... 158
    6.4.1.3 Methodology .......................................... 159
  6.4.2 Model Querying Results ..................................... 159
6.5 Concluding Remarks: Feasibility ................................. 160
  6.5.1 Final Remarks ............................................... 162
List of Tables

4.1 The API endpoints and their mapped functions. . . . . . . . . . . . . 113

B.1 Comparison between original and compact model representations, given
|W| worlds. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 186
B.2 Model creation results. . . . . . . . . . . . . . . . . . . . . . . . . 187
B.3 Model update results. . . . . . . . . . . . . . . . . . . . . . . . . . . 188
B.4 Model query results. . . . . . . . . . . . . . . . . . . . . . . . . . . 189
List of Figures

2.1 Agent Alice’s initial situation. .............................................. 11
2.2 An overview of the Jason reasoning cycle, adapted from [1]. .......... 18
2.3 A sample semantic rule. ...................................................... 30
2.4 The semantic rule for belief addition in AgentSpeak(PL). ............ 31
2.5 The semantic rule for determining applicable plans using the Bayesian network in AgentSpeak(PL). ............................................. 32
2.6 The semantic rule for evaluating test goals in TEAgentSpeak. ...... 33
2.7 The agent’s updated situation. .............................................. 40
2.8 After moving right, Alice considers (2,1) and (3,1) possible. ....... 43
3.1 Proof (by contradiction): $\neg\Diamond_B \ell$ is equivalent to $\Box_B \neg\ell$. ............. 67
3.2 Proof (by contradiction): given a belief rule $\Box(p \rightarrow q)$, certain belief (left) or possibility (right) of $q$ infer $p$ is certain (left) or possible (right). Labels ($w$) and ($v$) denote the formulas modelled by worlds in an arbitrary epistemic model. .................................................. 75
3.3 Proof (by contradiction): $\Box_B p$ is equivalent to $\Box_B\Box_B p$ under an S5 system. ................................................................. 77
3.4 The semantic rule for belief addition of a ranged literal in PAL-AgentSpeak. 79
## List of Code Listings

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Certain navigation in Jason.</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>The desirable navigation program. Modifications for uncertainty reasoning are shown in blue font.</td>
<td>47</td>
</tr>
<tr>
<td>3.2</td>
<td>Sample beliefs and rules.</td>
<td>57</td>
</tr>
<tr>
<td>3.3</td>
<td>Add \texttt{obs} to the epistemic model, so that we may filter possible locations by obstacles.</td>
<td>61</td>
</tr>
<tr>
<td>3.4</td>
<td>The workaround for moving right in PAL-AgentSpeak.</td>
<td>65</td>
</tr>
<tr>
<td>3.5</td>
<td>The PAL-AgentSpeak navigation program, excluding movement.</td>
<td>83</td>
</tr>
<tr>
<td>3.6</td>
<td>The “on” plans for the navigation agent.</td>
<td>86</td>
</tr>
<tr>
<td>3.7</td>
<td>From Listing 2.1, the standard AgentSpeak program for certain navigation.</td>
<td>88</td>
</tr>
<tr>
<td>3.8</td>
<td>The DEL-AgentSpeak navigation program.</td>
<td>102</td>
</tr>
<tr>
<td>4.1</td>
<td>Creating a standard S5 epistemic model.</td>
<td>107</td>
</tr>
<tr>
<td>4.2</td>
<td>Creating a standard S5 event model.</td>
<td>107</td>
</tr>
<tr>
<td>4.3</td>
<td>Model checking a standard epistemic model.</td>
<td>108</td>
</tr>
<tr>
<td>4.4</td>
<td>Apply a standard event model to a standard epistemic model.</td>
<td>108</td>
</tr>
<tr>
<td>4.5</td>
<td>Creating a compact S5 epistemic model.</td>
<td>109</td>
</tr>
<tr>
<td>4.6</td>
<td>Creating a compact S5 event model.</td>
<td>110</td>
</tr>
<tr>
<td>4.7</td>
<td>Model checking a compact epistemic model.</td>
<td>110</td>
</tr>
</tbody>
</table>
Glossary

AAMAS . . . . . . . . . . . . . . . . . Autonomous Agents and Multiagent Systems

AI . . . . . . . . . . . . . . . . . . . Artificial Intelligence

BDI . . . . . . . . . . . . . . . . . . . Belief-Desire-Intention

DEL . . . . . . . . . . . . . . . . . . Dynamic Epistemic Logic

HW . . . . . . . . . . . . . . . . . . Hintikka’s World

MAPC . . . . . . . . . . . . . . . . . Multi-Agent Programming Contest

PAL . . . . . . . . . . . . . . . . . . Public Announcement Logic

SAT . . . . . . . . . . . . . . . . . Satisfiability
Chapter 1

Introduction

In the field of artificial intelligence (AI), agents are software programs which act autonomously based on their perceptions of the environment [2]. Rational agents are those who make *optimal* action decisions [3]. The *belief-desire-intention* (BDI) model is a logical approach to modelling rational agents, where rationality is formally defined on the basis of agent (B)eliefs, (D)esires, and (I)ntentions.

In 1991, Rao and Georgeff [4] developed the BDI model for rational agency based on Michael Bratman’s philosophical model for understanding human rationality [5]. BDI agents are commonly used because they can handle complex dynamic environments by balancing reactive and proactive (goal-driven) behaviour [6]. BDI has been used to model agent behaviour in many different fields, such as: (i) geographical information systems, specifically for hospital site selection [7]; (ii) Short-term forecasting of economic markets [8]; (iii) air traffic management [9]; and (iv) fault detection, specifically used on the NASA space shuttle [10].

Several agent-oriented programming languages have been created based on the BDI model. A notable language is *AgentSpeak*, an abstract language introduced by
Rao [11]. Jason [1] is an agent-oriented platform that realizes the abstract language of AgentSpeak, and has recently gained considerable attention in the field of agent development [12]. We use the terms AgentSpeak and Jason interchangeably throughout this thesis but will explicitly differentiate when necessary.

Due to their ability to operate in highly dynamic environments, it is common for agent developers to place high expectations on the capabilities provided by BDI-based languages such as AgentSpeak. This typically requires developers to create ad hoc integrations with other AI capabilities, such as the ability to represent and reason about belief uncertainty, the automatic generation of plans to describe how agents achieve their goals, computer vision and natural language processing for interfacing with the world and humans, etc. [13, 6].

However, despite the seemingly-obvious integration of BDI with other forms of AI, BDI-based languages and platforms lack standardized approaches for most of these integrations. This is likely due to the fact that it is difficult for these languages to provide a standardized approach that is simultaneously simplistic and idiomatic to express, generalizable to any domain, and computationally efficient [13]. Regardless, the burden of integrating these approaches is placed on the developer. This indicates a larger issue for BDI-based languages: one of the main reasons why BDI is not adopted in industry is due to its sole focus on rationality and beliefs, desires, and intentions, and its lack of focus on integrated mechanisms to develop higher intelligence agents [13].

We aim to further the field of intelligent agents, specifically BDI agents, by integrating the ability to reason about belief uncertainty with the BDI-based language AgentSpeak. The proposed integration aims to extend AgentSpeak in an idiomatic manner by providing semantics that allow the agent to describe uncertainty in the
language, and to allow for symbolic constructs which enable possibilistic reasoning about uncertainty. This enables the agent to explicitly reason about uncertainty by distinguishing between certainty, possibility, and impossibility, ultimately allowing it to plan and act more effectively. This is achieved by integrating AgentSpeak with a modal logic for belief and knowledge known as epistemic logic [14]. In the coming sections, we enumerate the main contributions and structure of the thesis.

1.1 Main Contributions

We enumerate the major high-level contributions made by this thesis.

- This thesis conducts an analysis of the state-of-the-art approaches to uncertainty reasoning in AgentSpeak and identifies the limitations of existing extensions in the literature. To address the gap in the literature, we present two AgentSpeak extensions: PAL-AgentSpeak and DEL-AgentSpeak, which are based on two dynamic forms of epistemic logic: Public Announcement Logic (PAL) and Dynamic Epistemic Logic (DEL) [14]. The extensions provide possibilistic reasoning about uncertainty and vary in their level of expressiveness for capturing change.

- This thesis addresses the integration of an epistemic reasoner, Hintikka’s World [15], with the widely-used AgentSpeak development platform, Jason, in the context of implementing PAL- and DEL-AgentSpeak. Additionally, we present various enhancements to Hintikka’s World, such as optimized implementations tailored to the specific reasoning subclass required by PAL-/DEL-AgentSpeak, and the provision of the reasoner as a service by extending it with an application programming interface (API). Note that Jason and Hintikka’s World are merely
implementation choices, and that the works presented in this paper are applicable to all AgentSpeak derivatives and implementations, and in fact, are applicable to agent-oriented programming languages as a whole.

- This thesis demonstrates the utility of PAL- and DEL-AgentSpeak through the application of the extensions to various domains, including navigation and identification from the 2019 Multi-Agent Programming Contest (MAPC), and Minesweeper. This thesis evaluates the performance and scalability of PAL- and DEL-AgentSpeak through experiments that assess its worst-case impact on the reasoning cycle. We utilize constraints imposed by the 2019 MAPC to highlight the tractability and feasibility of these extensions in practice.

The following publications resulted from the research that occurred as part of this thesis:

- A detailed account of our original participation in the 2019 MAPC was published in [16]. We highlight some of the challenges faced while developing our AgentSpeak agents. The publication offers comprehensive insights into these challenges, including several ad hoc approaches and examples that highlight the complexity of reasoning about uncertainty without native or external support for uncertainty reasoning. The 2019 MAPC challenges serve as motivation for incorporating qualitative uncertainty reasoning in AgentSpeak, as proposed by this thesis.

- PAL-AgentSpeak was published at Autonomous Agents and Multiagent Systems (AAMAS) 2022 [17] which presented a novel extension for uncertainty reasoning to the field of agent systems, and applied the extension to the 2019 MAPC navigation example. This publication detailed:
The syntax and assigned informal semantics required for a qualitative uncertainty AgentSpeak extension enabled via PAL.

The corresponding integration with a PAL reasoner (Hintikka’s World). Additionally, an evaluation of scalability to indicate the performance impact of PAL-AgentSpeak.

Various optimizations to Hintikka’s World for our subset of models and reasoning capabilities were also presented and evaluated.

Various shortcomings in expressability due to the use of PAL, some suggested ad hoc workarounds, and proposing the use of DEL instead of PAL as future work. These are addressed by the following publication.

- DEL-AgentSpeak will be published at AAMAS 2023 [18]. This publication proposes a more expressive form of change compared to PAL-AgentSpeak, which, on top of PAL-AgentSpeak, include:
  
  - Additional syntactic components for change (named “on” plans)
  
  - Various necessary formalities, such as the formal semantics of the extension. These formalities help to solidify the soundness and generalizability of the contribution.
  
  - Integration with the DEL reasoner in Hintikka’s World.
  
  - A demonstration of how the shortcomings of PAL-AgentSpeak are addressed with the additional expressability provided by DEL-AgentSpeak.

These papers constitute most of the methodological, application, and evaluation work presented in this thesis.

In the following section, we present the outline of the thesis.
1.2 Thesis Outline

The outline of this thesis is provided as follows:

- Chapter 2 (Background & State of the Art) introduces the prerequisite content to understand this thesis. It includes an overview of the 2019 MAPC and its related uncertainty challenges, an explanation of the AgentSpeak language and its operation, an examination of the existing limitations in approaches and extensions for uncertainty reasoning in AgentSpeak, and a background on epistemic logic. Furthermore, it also introduces two dynamic variants of epistemic logic, namely public announcement logic (PAL) and dynamic epistemic logic (DEL), to handle changes in uncertainty.

- Chapter 3 (Methodology) outlines the methodology proposed for two Agent-Speak extensions that integrates with epistemic logic using an existing epistemic reasoner with support for PAL and DEL. The chapter formalizes two Agent-Speak extensions: one based on PAL (PAL-AgentSpeak) and the other based on DEL (DEL-AgentSpeak), allowing developers to choose between different levels of expressiveness. Finally, the chapter concludes with a discussion on the limitations and developer recommendations for choosing between the two presented extensions.

- Chapter 4 (Implementation) presents the most crucial implementation details for integrating PAL-AgentSpeak and DEL-AgentSpeak with Jason. The contributions of this section include the implementation of an optimized representation for uncertainty models in Hintikka’s World and the implementation details for both PAL-/DEL-AgentSpeak, including how they interact with Hintikka’s World.
We also provide analyses for the time and space complexity for all algorithms used by the implementation.

- Chapter 5 (Application) provides application domains with varying magnitudes of uncertainty, and appropriately applies PAL-/DEL-AgentSpeak to these domains to model and reason about uncertainty. Two of these applications are inspired by some of the most interesting and prevalent challenges faced in the 2019 MAPC; namely, agent navigation and identification. To show that we are not limited to the MAPC, we also apply the extensions to the classic game of Minesweeper. This chapter presents the PAL-/DEL-AgentSpeak program listings for each domain and provides a program and model trace that solidify the reader’s understanding of both extensions.

- Chapter 6 (Evaluation) provides experiments and evaluation results for the implementation of PAL-/DEL-AgentSpeak. The chapter looks at how the operations of the extensions scale given various parameters. The operations are evaluated with respect to computation time and memory. In the chapter, comments on the feasibility of the extensions in practice will be provided using the context and time constraints used in the 2019 MAPC.

- Chapter 7 (Conclusion) provides an overview of the chapters and highlights the main contributions of the thesis. The chapter will provide a brief overview of the contribution limitations and present various possible avenues for future work.
Chapter 2

Background & State of the Art

This chapter serves as a foundation for understanding the subsequent chapters of the thesis by introducing several key concepts. These include the 2019 Multi-Agent Programming Contest (MAPC) and its relevant uncertainty challenges, an overview of the AgentSpeak language including its syntax, semantics, and operation, a review of existing approaches and limitations for reasoning about belief uncertainty in AgentSpeak, and a background on epistemic logic and two dynamic variants used for representing and reasoning about uncertainty. These concepts are used to extend AgentSpeak with symbolic reasoning about uncertainty, which will be presented in the following chapter.
2.1 Background: 2019 Multi-Agent Programming Contest (MAPC)

The Multi-Agent Programming Contest (MAPC)\(^1\) is a yearly contest that brings together teams of agent developers to compete in a simulated environment. Every year, the contest proposes a different set of maps, objectives, and challenges for the competing teams.

By participating in the 2019 MAPC contest, we encountered various challenges during development that brought forth the pitfalls of programming agents in AgentSpeak, in particular, its inability to naturally reason about qualitative possibilistic uncertainty. A full account of our ad hoc approach to the 2019 contest is provided in [16]. It was made clear by the contest that AgentSpeak (and its lack of appropriate uncertainty extensions) did not allow us to represent or reason about our uncertainty in the way that we required.

In this section, we introduce the reader to the relevant aspects of the 2019 MAPC. We conclude the section with a discussion on some of the challenges incurred throughout the contest, specifically *navigation* and *agent identification*. These challenges will be used as running examples throughout the thesis.

2.1.1 Map Definition

The 2019 MAPC puts a heavy emphasis on promoting communication and collaboration between agents of the same team. The scenario requires the agents to explore the map in order to gather blocks and to coordinate with their teammates to assemble a configuration of blocks as required by each task. Submission of tasks before their

\(^1\)https://multiagentcontest.org/
deadline result in an allocation of points to the team. In the 2019 MAPC, time advances in discrete simulation steps. At each step, agents receive their respective perceptions and have 4 seconds to deliberate on a single action. If the 4 second deadline has passed, the agent misses the opportunity to perform an action.

One of the challenges posed by the 2019 MAPC is navigation due to the partial-observability of the environment. The agent is unaware of its actual location on the map and must infer its location based on a map definition and relative perceptions. The map definition is introduced in Example 2.1.1.

**Example 2.1.1.** An agent, Alice, knows the grid map (see Figure 2.1) and is given perceptions of surrounding obstacles, but is uncertain of their actual location. Despite this uncertainty, *Alice* must find a way to navigate to the *goal cell*. Alice has access to an action *move*, which updates their location and surrounding perceptions accordingly. We model this problem with the following literals:

- **loc**(*X*, *Y*): *Alice* is at location (*X*, *Y*)
- **obs**(*D*): an obstacle in direction *D* is perceived by *Alice* (among *down*, *up*, *left*, *right*)
- **dir**(*D*, *goal*): *D* is a direction along the shortest path from *Alice’s* current location to the nearest goal cell.

In our example *Alice* starts at location (1,1), receives the perception **obs**(*down*), and considers the shortest-path directions: **dir**(left, goal) and **dir**(right, goal). Figure 2.1 highlights *Alice’s* location (1,1) using a red dashed box.

### 2.1.2 Environment: Perceptions and Actions

At each simulation step, the agent is provided with various perception literals that describe its local environment. Given *D* ∈ {up, down, left, right}:
Figure 2.1: Agent Alice’s initial situation.

- \(\text{\textsf{obs}}(D)\) if an obstacle is perceived in direction \(D\), \(\neg\text{\textsf{obs}}(D)\) otherwise.

- \(\text{\textsf{moved}}(D)\) confirms the success of \(\text{\textsf{move}}(D)\) if it was performed in the previous simulation step.

In the case of Example 2.1.1, the agent is provided with the perceptions: \(\{\text{\textsf{obs}}(\text{down}), \neg\text{\textsf{obs}}(\text{up}), \neg\text{\textsf{obs}}(\text{left}), \neg\text{\textsf{obs}}(\text{right})\}\) upon sensing the environment. Note that no \(\text{\textsf{moved}}\) perceptions are given since the agent has not yet performed the \(\text{\textsf{move}}\) action.

The agent also has access to an action: \(\text{\textsf{move}}(D)\), which changes the agents location according to direction \(D\). This action succeeds if there are no obstacles in direction \(D\) and if the resulting location is within the bounds of the 5x5 map.

Example 2.1.2. After perceiving the initial situation, the agent performs \(\text{\textsf{move}}(\text{right})\). The success of this action moves the agent to location (2,1), and results in the following perceptions: \(\{\text{\textsf{obs}}(\text{down}), \neg\text{\textsf{obs}}(\text{up}), \neg\text{\textsf{obs}}(\text{left}), \neg\text{\textsf{obs}}(\text{right}), \text{\textsf{moved}}(\text{right})\}\). Note that the obstacle perceptions are identical to location (1,1), but the agent also receives confirmation that it has moved right.

The following section presents a brief overview of the challenges encountered in the 2019 MAPC and how we propose to address them. This will provide the necessary context and motivation for the remainder of the thesis.
2.1.3 Challenges: Agent Navigation and Identification

In order to compete in the 2019 MAPC, agents must work together in order to build task requirements. Agents have a partial view of their environment, where neither their absolute location nor their teammate identities can be perceived. This introduces difficulties in two critical components: 1) reliable navigation and 2) uniquely identifying friendly agents for collaboration. Without a mechanism to reliably reason about the uncertainty that clouds these components, the agent is not able to reliably navigate the map nor collaborate with friendly agents.

Our ad hoc approach to the 2019 MAPC [16] was hindered by the lack of appropriate extensions for uncertainty reasoning in AgentSpeak. Consequently, agents had to rely on a strategy of moving randomly and waiting for complete and reliable certainty about their local environment before attempting to navigate and identify teammates, resulting in a large number of wasted simulation steps.

Given the 2019 MAPC domain is symbolic in nature and lacks a probabilistic distribution, a qualitative approach to uncertainty reasoning would have been ideal. A qualitative approach such as possibilistic reasoning would provide us with the level of expressivity we required to reason about our uncertainty, while also being generalizable to other challenges and application domains. Possibilistic reasoning enables us to determine which locations and navigation directions, as well as agent identities, are deemed possible given our past and current perceptions, and therefore which actions may still be taken despite the current uncertainty. For example, if the agent is uncertain about its location, it may be able to navigate reliably if among its possible locations there is one common direction to get to its goal.

These challenges will be revisited in more detail throughout the thesis. Chapter 3 provides additional motivation for the representation and reasoning of uncertainty by
illustrating the navigation challenge through an idealized extension of the AgentSpeak language. In Chapter 5, we apply the proposed AgentSpeak extension to both the navigation and identification challenges, as well as a different program altogether, Minesweeper, to demonstrate the generalizability of our contributions.

In the next section, we introduce the reader to AgentSpeak, the language we propose to extend in this thesis to support reasoning with uncertainty.

2.2 Background: AgentSpeak: A Logical Language for BDI Agents

The Belief-Desire-Intention (BDI) agent model [4] is a practical reasoning model based on Michael Bratman’s BDI philosophical model [5]. The model aims to represent an agent’s current mental state and attitude using three main components: beliefs, desires, and intentions.

- Beliefs refer to the information that an agent believes to be true about its environment and represent the agent’s current mental state.

- Desires are used to represent the state an agent wishes to achieve, which can be an environmental state or an internal state of mind. Current desires are also known as goals.

- Intentions are goals that an agent chooses to fulfill through the execution of plans, which describe a sequence of sub-goals and actions.

- Events are generated by external or internal factors, such as changes in the agent’s environment or mindset (i.e., the addition or deletion of beliefs, goals, etc.).
AgentSpeak, presented by Rao in [11], is an abstract agent-oriented programming language which conforms to the principles of BDI. AgentSpeak syntax is heavily inspired by Prolog, using predicates and literals to describe components such as beliefs, goals, plans, and events. AgentSpeak formally describes the operational semantics of its reasoning cycle. There are several implementations of AgentSpeak, but Jason is the most prominently used [12]. Jason [1] is a Java-based interpreter for AgentSpeak that offers a comprehensive architecture for developing agents. The contributions made throughout this thesis are applicable to the wider scope of AgentSpeak, but we use the terms AgentSpeak and Jason interchangeably and make note when it is necessary to distinguish between the two.

2.2.1 Syntax and Semantics

In AgentSpeak, literals have the form $\ell(t_1, \ldots, t_n)$ with lower-case functor $\ell$ and $n \geq 0$ terms. Variable terms syntactically start with a capital letter; e.g., $\ell(Var_1, t_2)$ has one variable term $Var_1$. A most-general unifier (MGU) $\theta$ provides the most general substitution for variables that allow two terms to be equal, e.g., we say $\ell_1(t_1) = \ell_1(V_1)\theta$ when $\theta = \{V_1/t_1\}$. Throughout the dissertation, we conflate the term MGU with the substitution of variables in literals, as is done in [1].

A literal is considered ground when a MGU $\theta$ exists that unifies all variable terms with a ground value. Throughout the thesis, we use the symbol $\ell$ as a general representation of literals; we assume $\ell$ is unground unless explicitly stated. Due to Jason’s open-world assumption and three-valued logic, we must consider that literals may be strongly-negated $\sim\ell(\ldots)$. Weak negation is provided by the operator $\text{not}(\ell)$, however, this operator is only used in belief entailment.
2.2.1.1 Beliefs and Rules

In AgentSpeak, a belief base stores the agent’s set of belief literals and rules referred to as explicit and implicit beliefs, respectively. Beliefs in AgentSpeak are expressed using ground literals: $\ell$. Beliefs are queried using a conjunction of $n \geq 1$ literals which may be unground, strongly-, or weakly-negated: $\ell_1 \land \cdots \land \ell_n$. We also make use of belief rules as provided by Jason’s extended AgentSpeak syntax. A rule $\ell_H :\neg \varphi_B$ with literal $\ell_H$ and conjunction $\varphi_B$ state that if the rule body $\varphi_B$ is entailed by the agent’s beliefs, that the rule head $\ell_H$ should also be implicitly entailed.

Definition 2.2.1 (Belief Entailment). Given a conjunction $\varphi = \ell_1 \land \cdots \land \ell_n$ and MGU $\theta$, we say that $\varphi$ is entailed by a belief base $B$: $B \models \varphi_\theta$, iff for: 1) all non-weakly negated literals $\ell_i$ there exists either: a) an explicit belief $\ell_b \in B$ such that $\ell_i \theta = \ell_b$, or b) an implicit belief $(\ell_H :\neg \varphi_B) \in B$ such that $\ell_i \theta = \ell_H \theta_2$ where $B \models \varphi_B \theta_2$, and 2) for all weakly-negated literals $\neg \ell_j$: $B \not\models \ell_j \theta$.

In place of literals, Jason also allows for the use of internal actions which have the form $.ia(\ldots)$; a Java function corresponding to $.ia(\ldots)$ is used to determine entailment. This allows the agent to utilize algorithms written in Java from within its program. We will introduce specific internal actions as they are used throughout the thesis.

2.2.1.2 Goals, Events, and Plans

The language has two types of goals: achievement goals $!\ell_A$, which aim to achieve a state described by a ground literal $\ell_A$, and test goals $?\ell_T$, which test the beliefs for the condition described by a ground literal $\ell_T$. Events $+\varphi$ and $-\varphi$ respectively represent the addition or deletion of some belief or goal, such that $\varphi = \ell!/\ell/?\ell$. Note
that weakly-negated literals are solely expressed during belief entailment and are not permitted in the description literals of goals or events.

Plans have the form $te: c \leftarrow b$, where $te$ is a potentially-unground event trigger, $c$ (if provided) is a belief conjunction known as the plan context or precondition, and $b$ (if provided) is the plan body composed of a sequence of grounded sub-goals ($!/?\ell$), actions ($\ell$), or belief additions or deletions $+/-\ell$. Given a current event $e$, we say that the plan is relevant if given some MGU $\theta_R$, $te\theta_R = e$. Given a relevant plan with $\theta_R$, the plan is considered applicable for execution with MGU $\theta_A$ if the context $c$ is entailed by the current belief base $B$: $B \models c\theta_R\theta_A$.

Events and plans are stored by the agent’s event set and plan library, respectively. An AgentSpeak program is defined by a developer with the following sequence: 1) initial beliefs and rules, 2) initial goals, and 3) plan definitions. All syntactic components are terminated with a ‘.’, with the exception of non-terminating plan body sequences which use ‘;’. The following example provides an AgentSpeak program for the MAPC navigation challenge (without uncertainty).

**Example 2.2.1** (Navigation Example). We introduce a standard AgentSpeak program for reasoning about certain navigation.

In this listing, the agent believes its true location is $loc(1,1)$ (Line 1). The rules on Line 3 infer $\text{dir}$ and $\text{obs}$ beliefs from the current location of the agent. We declare $!\text{nav}$ as the agent’s initial goal on Line 7. The navigation plan (Line 9) handles the addition of the initial $!\text{nav}$ goal, which navigate based on the shortest path direction $\text{dir}$, and recursively introduces the goal $!\text{nav}$. The plan on Line 14 handles the event where a new belief perception $\text{moved}(right)$ is added to the belief base, denoting successful movement of locations — this plan revises location beliefs accordingly.
loc(1,1).

// Rules for dir/obs that relate to location
obs(down) :- loc(1,1) | loc(2,1).
dir(right, goal) :- loc(1,1) | loc(2,1).
...

// etc. for all obs/dir :- loc(1,1)

! nav.

// Plan for navigation
+! nav : dir(D, goal)
<- move(D);
! nav.

// Belief plans for changing locations (?)
+ moved(right) : loc(X, Y)
<- -loc(X, Y);
+ loc(X + 1, Y).

// ... similar plans for moved(left), etc.

### 2.2.2 Reasoning Cycle and Operational Semantics

The operational semantics of AgentSpeak describe a reasoning cycle that allows the agent to find relevant plans using the agent’s plan library and based on belief and goal events, determine plan applicability using the belief base, and execute the plan bodies so that the agent may interact with the environment and change its internal state. Jason implements the operational semantics, staying consistent with the operational semantics defined by the abstract language of AgentSpeak, but also integrating the agent with various other architectural components — for example, Jason allows the developer to provide an environment implementation which adds external perceptions to the belief base and performs actions requested during the execution of plans.

The complete reasoning cycle is shown in Figure 2.2, which is a figure directly adapted from the Jason book ([1]). We will now enumerate the steps involved in each iteration of the reasoning cycle. The numbering of each of these steps corresponds to the numbered components in the reasoning cycle diagram shown in Figure 2.2.
Steps marked with an asterisk (*) are omitted as they are not relevant to this thesis; however, these steps are still listed here for the sake of completeness.

![Diagram of Jason reasoning cycle](image)

Figure 2.2: An overview of the Jason reasoning cycle, adapted from [1].

1. **Perceiving the Environment**: the agent perceives its environment, obtaining a set of literals representing perceptions. Perceptions are then sent to the belief update function and added to the belief base.

2. **Updating the Belief Base (buf)**: the belief update function (buf) processes perceptions. By default, all old perceptions are removed from, and new perceptions are added to, the belief base.

3. (***) Checking for Mail
4. (*) Socially Acceptable Messages

5. **Selecting an Event**: the agent selects a single event from the set of events through the event selection function. By default, this function obtains the first available event in the event set.

6. **Obtaining Relevant Plans**: using the plan library, find all plans with triggers that match the selected event; these are known as relevant plans.

7. **Obtaining Applicable Plans**: every relevant plan whose context is a logical consequence of the belief base is considered an applicable plan.

8. **Selecting an Applicable Plan**: the agent selects one applicable plan to handle its previously-selected event. By default, the first applicable plan is chosen. The selected applicable plan is stored in the agent’s *set of intentions*, an architectural component that stores new and existing intentions that have not completed execution.

9. **Selecting an Intention**: the agent chooses one intention from the set of intentions to execute. By default, a ‘round-robin’ approach is used to ensure fairness.

10. **Executing the Selected Intention**: in each reasoning cycle, the Jason agent executes one step in the selected intention’s plan body. As defined earlier, this may be an: action, sub-goal, belief addition or removal, or an internal action. Once all steps in a plan body have been executed, the intention is removed from the set of intentions. The execution of each type of step is described briefly below:
(a) **Action**: an action allows the agent to interact with its external environment.

(b) **Sub-goal**: a sub-goal is introduced to the set of events and the current intention is suspended until the sub-goal is selected and completed in a future reasoning cycle.

(c) **Belief Addition/Deletions**: the agent updates the beliefs in its belief base through belief additions and deletions.

(d) **Internal Action**: the internal action’s corresponding Java implementation is invoked.

The overview of the reasoning cycle’s execution, as provided, should be sufficient to understand the execution of the Jason program listings presented in this thesis.

This section provided the necessary background for understanding AgentSpeak, a popular agent-oriented language for developing BDI-based agents, and the corresponding Jason platform. In this section, we introduced the reader to the syntax and semantics of AgentSpeak programs and the agent’s reasoning cycle. In the next section, we explore uncertainty reasoning in AgentSpeak. We identify the current shortcomings of the language with respect to uncertainty reasoning, and we explore the current state of the art with respect to AgentSpeak extensions for uncertainty reasoning.
2.3 State of the Art: Uncertainty Reasoning in AgentSpeak

Uncertainty is a common and complex issue that arises in many aspects of agent development and behaviour. As agents operate in dynamic and unpredictable environments, they must contend with incomplete information, unpredictable outcomes, and unexpected events. The ability to represent, manage, and reason about uncertainty is therefore a critical skill for agents, as it can impact their effectiveness and performance in achieving their goals.

In the context of BDI-based programming languages, agents may experience uncertainty related to their beliefs, desires, and intentions. Beliefs model the agent’s mental state, including its understanding of the environment; thus, it is paramount that belief uncertainty is represented and can be reasoned about appropriately. In this section, we will dive into the current state of AgentSpeak extensions that can model and reason about belief uncertainty. To begin with, Example 2.3.1 illustrates the use of the standard AgentSpeak language for modelling and reasoning about uncertainty.

**Example 2.3.1.** By default, Jason extends the standard AgentSpeak syntax with the ability to attach meta-information to literals using *annotations*. Annotations may be used by agents or language extensions to assign additional semantic meaning to certain literals. For example, the Jason book [1] annotates beliefs with degrees of certainty, so a belief $\ell$ with a degree of certainty of 0.5 is expressed as $\ell[\text{deg}(0.5)]$.

Although the above annotation example demonstrates the ability to handle uncertainty, the developer is still obligated to manage uncertainty in an ad hoc fashion using standard syntax and semantics. Instead, the language should impose syntactic conventions and provide automated mechanisms for representing and handling uncer-
tainty; this allows the developer to focus on expressing the relevant components of the domain while relying on the language to perform the heavy lifting of managing uncertainty. In this regard, this section examines different AgentSpeak extensions that have been proposed in the literature.

Briefly, we discuss the topic of belief revision, which is often seen in the literature as a way of reasoning about uncertainty. Belief revision [19] is the process of revising beliefs based on new evidence or environmental changes. Belief revision on its own is unsatisfactory for representing and reasoning about uncertainty, as it requires complete and accurate information and does not capture varying degrees of certainty. Additional interventions are needed to adequately represent and reason about uncertainty; for example, the belief revision approach presented in [20] relies on the annotation definitions of Example 2.3.1, where beliefs are revised according to their annotated degrees of certainty.

The following subsections will explore the state of the art with respect to Agent-Speak extensions that reason about uncertainty, and the methodologies that these extensions use to extend the language of AgentSpeak.

2.3.1 Existing Uncertainty Extensions

In the following subsections, we explore the state of the art with respect to quantitative and qualitative extensions to BDI-based languages such as AgentSpeak.

2.3.1.1 Quantitative Extensions

Quantitative methods provide precise methodologies to model and reason about uncertainty. They are typically most suitable when the domain requires numerical representations for uncertainty. The vast majority of AgentSpeak extensions for
modelling and reasoning uncertainty are based on quantitative methods such as probability (a measure of likelihood) and plausibility (degree of belief). In the case of some AgentSpeak extensions [21, 22], both probabilistic and plausibilistic interpretations of belief uncertainty are provided.

AgentSpeak extensions for probabilistic uncertainty rely on methodologies such as: Bayesian networks and conditional probabilities for general probabilistic reasoning [23, 24, 25] and continuous real-time domains such as supervisory control and data acquisition (SCADA) [23, 26], particle filtering [27] and partially-observable Markov decision processes (POMDP) [28, 29] for sensor fusion (combining sensor data to increase accuracy) and state estimation, and probabilistic theory of mind [30] for reasoning about deception by modelling the minds of external agents [30, 31].

Whereas probabilities are meant to measure likelihood, plausibility measures instead interpret quantitative values as degrees of belief and are typically used to represent preferences among beliefs [22]. The AgentSpeak extensions [22, 21, 26, 32] use methodologies such as Dempster-Shafer theory [33] and possibility theory [34] to model numerical degrees of belief. Dempster-Shafer theory allows for the description of belief uncertainty through belief functions rather than probabilistic distributions, whereas possibility theory assigns degrees of possibility or plausibility to individual beliefs. Similar to the previous probabilistic methodologies, plausibility measures provide a different interpretation of numerical values but can also be used for continuous domains such as SCADA [26].

Adjacent to the notion of probability and plausibility is the many-valued fuzzy logic, which is a less-precise quantitative methodology for representing and reasoning about vagueness. Extensions such as [35, 36] integrate the BDI logic and the AgentSpeak
programming language with fuzzy logic to reason about uncertainty in cyber-physical systems and internet of things interoperability [36].

In Section 2.3.2 we provide an in-depth analysis of the methodology used by two notable quantitative extensions AgentSpeak(PL) [25] and TEAgentSpeak [22].

2.3.1.2 Qualitative Extensions

Qualitative methods for modelling and reasoning about uncertainty allow for symbolic representations of uncertainty; these approaches typically rely on relative orderings or symbolic logic. Qualitative representations tend to be less precise than quantitative approaches, but they integrate well with the syntax of symbolic languages such as AgentSpeak and can still be used to model quantitative domains (albeit less precisely than their quantitative counterparts). As we shall see, the recurring theme for qualitative extensions is that they are used in theory (i.e. to extend the BDI logic) but fail to be integrated with practical languages such as AgentSpeak.

Modal Approaches  Traditionally, one of the simplest approaches to reasoning about uncertainty is provided by modal logic and its possible world semantics [37]. Given a set of possible worlds (states), semantics are assigned to modalities for necessity □ and possibility ◇, allowing the agent to reason about uncertainty by distinguishing between something that is certain, possible, or neither. We formally introduce modal logic in Section 2.4.

The BDI paradigm, upon which AgentSpeak is founded, is specified using modal logic based on possible world semantics. Beliefs, desires, and intentions are specified using modalities [4]. BDI also uses modalities to reason about past, present, and future beliefs, desires, and intentions. In the transition from theory to practice, AgentSpeak
conforms to the belief, desire, and intention modalities using set-based representations. Although the use of set-based representations allows for a computationally efficient implementation [11, 21], its separation from the modal logic removes the inherent ability to reason about possible uncertainty [38, 21].

As we have lightly discussed above, modal logic and the underlying possible world semantics are one approach to symbolically representing uncertainty. In certain domains, the agent may need to distinguish between varying levels of belief certainty — various works [39, 40, 41, 42] extend the BDI logic with graded modalities that allow for this. Unlike quantitative extensions, modal-based extensions exist solely as theoretical extensions of BDI and do not extend the practical language of AgentSpeak as their integration is associated with various methodological, implementation, and computational complexity challenges [43, 21].

An attempt has been made to introduce a theoretical modal-based extension to AgentSpeak. Modal-based BDI extensions, as described in [39, 40], use multiple belief modalities to represent levels of uncertainty. For example, $B_A \ell$ represents absolute belief in $\ell$, $B_U \ell$ represents $\ell$ is usually believed, and so on. The G-Jason extension [44] was inspired by these modal-based extensions but replaces modalities with quantitative values representing degrees of belief (similar to the standard Jason annotation approach from Example 2.3.1). The reason for this change was to simplify the computational complexity of the approach, as implementing the semantics for a modal-based approach is difficult and computationally complex [21, 44]. However, as a result of this simplification, the approach loses the benefits of a modal-based approach to uncertainty.
Preference and Partial-Order Semantics  Partial orderings, on the other hand, can be used to rank a set of possible worlds based on the relative desirability of each state — sometimes referred to as a plausibility model. Plausibility models can be used to provide semantics for belief preferences, typically used for achieving belief revision and non-monotonic default reasoning [37]. The AgentSpeak extensions [22, 21] integrate with plausibility models to provide belief revision and reasoning about uncertainty. Partial orderings are difficult to compactly represent and update within an AgentSpeak program [22]. As we see in the respective extensions, quantitative weightings of beliefs and worlds are used to compactly represent the partial ordering of possible belief states, thus removing the qualitative status of the extension.

Despite various extensions for uncertainty reasoning in the BDI logic, there is a lack of qualitative extensions for uncertainty reasoning in the derived programming language AgentSpeak. Given the various quantitative extensions in the literature, the next section will explore the methodologies used by these extensions to extend the AgentSpeak language for uncertainty reasoning.

2.3.2 Extending AgentSpeak

Broadly speaking, extensions for reasoning about uncertainty typically rely on an external underlying model or reasoning engine to represent and reason about uncertainty. We can identify a pattern among these extensions that allows us to classify them according to three fundamental operations: model creation, model updating, and model querying. Model creation pertains to the establishment of an initial model of uncertainty. Model updating involves modifying the model of uncertainty to reflect changes in belief uncertainty, while model querying involves utilizing the underlying model or reasoning engine to inquire about the current belief uncertainty. In order to
allow for the creation, updating, and querying of a model of uncertainty, extensions must modify the AgentSpeak syntax, assigned semantics, and operational semantics (i.e., reasoning cycle).

We explore two previously-discussed AgentSpeak extensions for uncertainty: AgentSpeak(PL) [25] and TEAgentSpeak [22], which are quantitative extensions for reasoning about uncertainty that, respectively, use a Bayesian network and plausibility model as models for uncertainty. These particular extensions were selected because they presented a complete and formal account of their extended language and operational semantics. This allows us to examine how they augment the AgentSpeak language, including syntax, assigned semantics, and operational semantics.

Lastly, since there are no qualitative AgentSpeak extensions for reasoning about uncertainty in the literature, we explore a qualitative extension that integrates with a related field, defeasible reasoning. Defeasible reasoning is commonly used in argumentation, conflict resolution, and legal reasoning. In [45], an extension is presented that integrates AgentSpeak with defeasible reasoning, where the symbolic definitions of AgentSpeak are assigned semantics. However, as the extension does not specify its corresponding operational semantics, we limit our examination to its extension of syntax and assigned semantics.

We begin by examining how these extensions alter the syntax and semantics of the AgentSpeak language to enable the integration of the extension by inferring the necessary information for creation, updating, and querying.
2.3.2.1 Extending the Language

In terms of syntax and assigned semantics, quantitative approaches extend the language of AgentSpeak in a similar way. They extend the AgentSpeak syntax to allow the agent to specify the numerical probability or plausibility of beliefs.

In AgentSpeak(PL), a new operator was introduced that allowed the provision of probabilities. Given a belief $\ell$, AgentSpeak(PL) allows the agent to assign a probability of 50% as follows: $\%\ell = 0.5$. Similarly, TEAgentSpeak provides their own operator for assigning a plausibility of 0.5 to belief $\ell$: $\ast(\ell, 0.5)$. Both extensions allow for the use of their respective operators within the description of initial agent beliefs, the description of plan bodies to update belief uncertainty during execution, and within the description of plan contexts and test goals to query uncertainty.

Qualitative extensions such as the one for defeasible reasoning [45] rely on standard syntax with extended semantics assigned to reserved literals to infer information about the domain. In defeasible logic, there is a notion of facts, strict rules, defeasible rules, etc. The meaning behind each of these terms is outside the scope of this thesis and are thus not introduced; however, we are interested in how these different components are represented syntactically using AgentSpeak.

The proposed extension maps standard beliefs to facts, while special predicates of the form $\text{strict\_rule}(\text{Head},\text{Body})$ and $\text{defeasible\_rule}(\text{Head},\text{Body})$ represent strict and defeasible rules, respectively, with each rule having the form $\text{Body} \rightarrow \text{Head}$. The symbolic nature of defeasible reasoning rules aligns with the symbolic syntax of AgentSpeak, making the integration seamless.

In general, quantitative approaches require distributions that represent probabilities or plausibilities. On the other hand, qualitative approaches can model any domain as they do not require numerical distributions but come at the cost of imprecision.
Reasoning with numerical values can be cumbersome and prone to errors and typically involves the selection of arbitrary numerical values to express levels of uncertainty. Qualitative extensions, which rely primarily on the assignment of additional semantics to the language, facilitate the idiomatic use of syntactic constructs to represent uncertainty in AgentSpeak.

2.3.2.2 Extending the Reasoning Cycle

Operational semantics are critical to formalizing the extensions made to the Agent-Speak reasoning cycle, providing clarity, reproducibility, and generalizability of the extension. We explore the operational semantics proposed by AgentSpeak(PL) [25] and TEAgentSpeak [22, 21]. To do so, we first formally introduce the operational semantics of standard AgentSpeak.

**Definition 2.3.1 (Operational Semantics of AgentSpeak).** Operational semantics are presented as semantic rules that define the transition relations between AgentSpeak configurations. We use a compacted form of Jason’s configuration (for a full definition, see [1]) \( \langle ag, C, T, S \rangle \), where:

- **ag** (agent state) is a tuple \( \langle B, P, \ldots \rangle \):
  - Belief base \( ag_B \) holds the agent’s beliefs and rules and is initialized with initial beliefs and rules defined by the program.
  - Plan library \( ag_P \) holds the agent’s set of plan definitions.

- **C** (circumstances) is a tuple \( \langle I, E, \ldots \rangle \):
  - \( C_I \) is the current set of intentions \( \{ i, i', \ldots \} \) where \( i \) is an execution stack containing a trace of plans.
\( C_E \) is a set of event tuples \((te, \iota)\) with an event \(te\) raised by intention \(\iota\).

- \( T \) (transition system) is a tuple \(\langle \iota, R, Ap, \ldots \rangle\):
  - \( T_\iota \) is the currently selected intention
  - \( T_R \) and \( T_{Ap} \) contain relevant and applicable plans as tuples \((p, \theta)\), with plan \(p\) and MGU \(\theta\).

- \( S \) (current step) is one of the standard AgentSpeak steps, including those most relevant to this thesis: \texttt{ApplPl} (obtain applicable plans), \texttt{SelAppl} (select applicable plan), \texttt{SelInt} (select intention), \texttt{ExecInt} (execute intention), and \texttt{ClrInt} (clear intention).

Since we will be extending the standard semantic rules of AgentSpeak, any modifications to the rules will be boxed. Semantic rules have the form shown in Figure 2.3, labelled (label), with \(p_n\) premises, and a transition conclusion between AgentSpeak configurations \(c\) to \(c'\).

\[
\frac{p_1 \ldots p_n}{c \rightarrow c'} \quad \text{(label)}
\]

Figure 2.3: A sample semantic rule.

Following this, we discuss how the uncertainty extensions AgentSpeak(PL) and TEAgentSpeak present their extended operational semantics according to their respective model creation, updating, and querying operations.

### 2.3.2.3 Model Creation

The model creation phase occurs during the initialization of the agent’s configuration, preceding any transitional semantic rules. In the AgentSpeak(PL) program, initial
beliefs and probabilities are used to initialize a probabilistic Bayesian network stored as the extended agent component $ag_{PBN}$ within the initial AgentSpeak(PL) configuration. In the case of TEAgentSpeak, the belief base $ag_B$ is replaced with $ag_G$ a weighted plausibility model named the global uncertain belief set (GUB), which also is initialized given initial belief plausibilities defined by the agent.

2.3.2.4 Model Updating

Both AgentSpeak(PL) and TEAgentSpeak provide similar semantics for the standard AgentSpeak belief addition $+\ell$ and deletion $-\ell$ operators, where $+\ell$ equates to complete certainty: $\%\ell = 1$ or $*(\ell,1)$, and $-\ell$ equates to complete uncertainty: $\%\ell = 0$ or $*(\ell,0)$. Otherwise, the agent may directly specify updates to uncertainty using the forms $\%\ell = -$ or $*(\ell,-)$ in place of the standard addition/deletion operations.

Both extensions override the semantic rules for ExecInt to allow for the appropriate updating of the underlying model of uncertainty. To demonstrate, the semantic rule in Figure 2.4 represents one of the model update rules from AgentSpeak(PL) [25]. In this case, a standard belief addition results in the update of the Bayesian network component $ag_{PBN}$ via the function $RecalcBN(\ldots)$. A similar semantic rule is used for belief deletion and for their extended probabilistic update operator $\%$.

$$\begin{align*}
T_i = i[head \leftarrow +b; h] \\
\langle ag, C, T, ExecInt \rangle \rightarrow \langle ag', C', T', Crlnt \rangle \quad (AddBel)
\end{align*}$$

where

$$\begin{align*}
ag'_B &= ag_B + b \\
\ldots \\
ag'_{PBN} &= RecalcBN(ag'_B, \ldots, ag_{PBN})
\end{align*}$$

Figure 2.4: The semantic rule for belief addition in AgentSpeak(PL).
The semantic rules for model updates in TEAgentSpeak follow a similar pattern, where the boxed line in Figure 2.4 is replaced with the respective model update function for updating TEAgentSpeak’s GUB component.

2.3.2.5 Model Querying

The model querying process involves the semantic connection between AgentSpeak’s component for belief querying and the respective models for uncertainty. During the AgentSpeak reasoning cycle, belief queries occur at two points: evaluating plan contexts (step AppPl) and the evaluation and execution of test goals (step ExecInt).

In AgentSpeak(PL), the agent is able to reason about possibilities in plan contexts and test goals via both standard AgentSpeak and extended probabilistic syntax, e.g., a context such as $\ell_1 \land %\ell_2 = 0.5$ queries whether $\ell_1$ is certain and that the probability of $\ell_2$ is 0.5. The two semantic rules in Figure 2.5 are used to compute the set of applicable plans $T_{Ap}$. The function call $AppPlans(\text{ag}_B,T_R)$ uses the standard AgentSpeak functionality for evaluating contexts based on any non-probabilistic belief queries (e.g., $\ell_1$) and is joined with the applicable plans from the function call $AppPlans'(\text{ag}_{PBN},T_R)$ which evaluates probabilistic beliefs (e.g., $\ell_2$) based on the Bayesian network $\text{ag}_{PBN}$.

\[
\frac{AppPlans(\text{ag}_B,T_R) \cup AppPlans'(\text{ag}_{PBN},T_R) \neq \{\}}{(ag,C,T,\text{Appl}) \rightarrow (ag,C,T',\text{SelAppl})}
\]

where $T'_{Ap} = AppPlans(\text{ag}_B,T_R) \cup AppPlans'(\text{ag}_{PBN},T_R)$

Figure 2.5: The semantic rule for determining applicable plans using the Bayesian network in AgentSpeak(PL).

We also present a semantic rule given by TEAgentSpeak in Figure 2.6, which demonstrates the evaluation of test goals using TEAgentSpeak’s GUB component $ag_G$. 

32
The semantic rule is simple; the GUB is used instead of the standard belief base to determine entailment of the test goal formula \( ?\ell \). These semantic rules then follow a similar fate to standard AgentSpeak: the remainder of the intention \( T_i \) is unified with test goal unifier \( \theta \), and the updated intention is added to the set of current intentions \( C_I \).

\[
T_i = i[\text{head} \leftarrow ?\ell; h] \quad \text{ag}_G \models \ell \theta \\
\langle ag, C, T, \text{Execlnt} \rangle \rightarrow \langle ag, C', T, \text{Clrlnt} \rangle \quad \text{(TestGl)}
\]

where \( T'_i = i[(\text{head} \leftarrow h)\theta] \)

\( C'_I = (C_I \setminus \{T_i\}) \cup T'_i \)

Figure 2.6: The semantic rule for evaluating test goals in TEAgentSpeak.

We conclude the section with important remarks on the computational complexities of these extensions.

**2.3.2.6 Computational Complexity**

One of the difficulties with uncertainty is the associated computational complexity due to the need to account for all possible situations [21, 22]. We briefly explore the complexities associated with AgentSpeak(PL) and TEAgentSpeak.

The worst-case time and space complexity of constructing a Bayesian network (i.e., a graph representation) is exponential in the number of variables, since the network may need to represent all possible joint probability distributions over the variables [46]. For AgentSpeak(PL), this means that model creation and updating both have exponential complexities due to their construction (and reconstruction) of a Bayesian network. Querying the Bayesian network can be done in polynomial time [46].
TEAgentSpeak uses its GUB to represent various plausible states. Despite TEAgentSpeak attempting to divide the beliefs in the GUB for tractability, the approach is time and space exponential in the worst case [22]. Model updating can be performed in logarithmic time, whereas querying (that is, obtaining the uncertainty of a given belief) in TEAgentSpeak is NP-complete [22].

In an agent setting, it is crucial to consider computational complexity while also weighing the benefits of reasoning about uncertainty. During agent initialization, the model creation process takes place, whereas model updating and querying operations occur much more frequently throughout the reasoning cycle. Despite the inherent complexity associated with uncertainty, it might be advantageous to minimize the complexity of model updates and queries by shifting it to model creation, in order to enable rapid and responsive agent behaviour.

### 2.3.3 Key Findings

We summarize the section with various key findings:

- There are numerous quantitative extension implementations available for representing and reasoning about uncertainty in AgentSpeak. However, in some domains, they may be needlessly precise or unfeasible because of the difficulty in obtaining quantitative distributions. Additionally, when integrating with the language of AgentSpeak, quantitative approaches tend to be less idiomatic to express due to the use of numerical values in comparison to qualitative counterparts [38].

- Conversely, while there exists various theoretical qualitative extensions for uncertainty, they have not yet been integrated with practical languages such
as AgentSpeak. Qualitative approaches therefore lack an implementation due to various methodological and implementation challenges, and computationally complex properties of their proposed logical dialects [21, 38].

- In general, extensions for uncertainty make use of an external model or reasoner to provide representations for and reasoning about uncertainty. This is done through three main operations: 1) model creation, 2) updating, and 3) querying, which, respectively, integrate into: 1) the initialization of the agent configuration, 2) the belief addition and deletion operators, and 3) the computation of the set of applicable plans and evaluation of test goals.

- Uncertainty is computationally complex from both a time and space complexity. In the absolute worst-case scenario, the time and space complexity of existing quantitative approaches are exponential; however, trade-offs can be made with respect to whether the time complexity occurs during creation, updating, or querying. For agents, it is typically better to optimize model updating and querying due to the frequency of these operations.

Our proposal is to integrate a modal approach to reasoning about uncertainty with AgentSpeak. This symbolic approach is expected to integrate well with the AgentSpeak language, offering a more natural, albeit less precise, method for dealing with uncertainty. Although modal logic has been dismissed in the past as a suitable approach to uncertainty reasoning due to the computational complexity of the modal logic dialects that are typically integrated with BDI [21, 38], we propose the use of single-agent PAL and DEL which provide tractable computational complexities [47], while allowing for a much more idiomatic integration with AgentSpeak in comparison to its quantitative counterparts. In the next section, we introduce epistemic logic and
its dynamic variants, PAL and DEL, which employ modalities to provide a purely symbolic approach to reasoning about the statics and dynamics of uncertainty.

2.4 Background: Epistemic Logic For Qualitative Reasoning About Uncertainty

Epistemic logic, as described in [14], utilizes modal logic to assign interpretations of knowledge/belief to standard modalities. The modalities $\Box_B$ and $\Diamond_B$ signify necessary (certain) and possible beliefs, respectively. In this section, we introduce the syntax and semantics of epistemic logic. Given that standard epistemic logic does not provide constructs for change, we introduce two dynamic variants of epistemic logic: public announcement logic (PAL) and dynamic epistemic logic (DEL). In this thesis, we utilize a modal logic reasoner based on propositions rather than one based on first-order logic.

Although first-order formalisms for PAL and DEL have been developed [48], they are associated with a higher computational complexity to provide their higher level of expressiveness, which can result in more complex and time-consuming reasoning tasks [48]. Agents must be able to make reactive decisions; to do so, it is critical that their reasoning processes are fast and responsive. As such, this thesis will be based on the simpler proposition-based formalism. It is worth noting that higher-order formalisms are beyond the scope of this thesis but may be investigated as future work. From this point forward, any reference to epistemic logic implies its propositional form.
2.4.1 Epistemic Logic

Given a finite non-empty set of atomic propositions $P$, the syntax for a single-agent epistemic formula $\varphi$ is as follows [14]:

$$\varphi ::= p \mid \varphi \land \varphi \mid \neg \varphi \mid \Box_B \varphi$$

$p \in P$

A disjunction $\varphi_1 \lor \varphi_2$ is equivalent to $\neg (\neg \varphi_1 \land \neg \varphi_2)$, and the possibility modality $\Diamond_B \varphi$ is equivalent to the dual of certain belief $\neg \Box_B \neg \varphi$. The modalities $\Box_B \varphi$ and $\Diamond_B \varphi$ are read as “the agent believes $\varphi$” and “the agent considers $\varphi$ to be possible”, respectively.

2.4.1.1 Semantics

The semantics of entailment are provided by possible world semantics, also referred to as an epistemic model [14]. Given a set of atomic propositions $P$, an $S5$ compact epistemic model is defined as the triple $M = (W, V)$, where:

- $W$ is a finite set of worlds,
- $V : W \to 2^P$ is a valuation function

$W$ represents the possible worlds and $V$ is a valuation that maps each world to its propositional state. The formalism above is a compact representation of an epistemic model. It is common to also see an indistinguishability relation $R \subseteq W^2$ which captures the indistinguishability (and thus uncertainty) of the possible worlds. For the thesis, we solely adopt the $S5$ system of epistemic models which implicitly models $R = W^2$.

The $S5$ system provides the strongest notion of necessity, and in our case, it implies that all worlds are indistinguishable from one another. From a philosophical
perspective, S5 systems are meant to model knowledge, a stronger alternative to belief [14], but provides various desirable properties and optimizations for use with agents in practice, as we will see throughout Chapters 3 and 4. Additionally, this will not interfere with AgentSpeak’s interpretation of beliefs.

Given an epistemic model $M$ and a reference world $w \in W$ named the pointed world, entailment of an epistemic formula $\varphi$ is represented as $(M, w) \models \varphi$ where [14]:

- $(M, w) \models p$ iff $p \in V(w)$
- $(M, w) \models \neg \varphi$ iff $(M, w) \not\models \varphi$
- $(M, w) \models (\varphi_1 \land \varphi_2)$ iff $(M, w) \models \varphi_1$ and $(M, w) \models \varphi_2$
- $(M, w) \models \Box_B \varphi$ iff for all $w' \in W$: $(M, w') \models \varphi$

Given that belief formulae are evaluated independently of a pointed world (due to our compact representation of the epistemic model), we use the following short form to evaluate the belief of $\varphi$: $M \models \varphi$ iff for all $w' \in W$: $(M, w') \models \varphi$. We demonstrate epistemic logic using the MAPC navigation problem in the following example.

**Example 2.4.1 (Navigation in Epistemic Logic).** We create an epistemic model representing the initial uncertainty of the navigation agent. Each non-obstacle location on the map is considered a possible world. Each possible world then models the corresponding propositions for location $\text{loc}(x, y)$, obstacle perceptions $\text{obs}(d)$, and shortest-path navigation direction $\text{dir}(d, \text{goal})$:

- $W = \{ w_{xy} \mid (x, y) \in \{0, \ldots, 4\}^2 \setminus \{(1, 2), (2, 2)\} \}$,
- For all worlds $w_{xy} \in W$:
  - $\neg \text{loc}(x, y) \in V(w_{xy})$
\[- \text{obs}(d) \in V(w_{xy}) \text{ s. th. there is an obstacle in direction } d \text{ at } (x, y). \]

\[- \text{dir}(d, \text{goal}) \in V(w_{xy}) \text{ s. th. } d \text{ is the direction to travel among the shortest path from } (x, y) \text{ to the goal } (2,3). \]

In this epistemic model, we can reason about our uncertainty as follows:

- \(M \models \neg \Box_B \text{loc}(1,1)\): “I am uncertain that my location is (1,1)”
- \(M \models \Diamond_B \text{loc}(1,1)\): “I consider location (1,1) a possibility”
- \(M \models \Box_B \neg \text{loc}(1,2)\): “(1,2) holds an obstacle, I am certain it is not my location”
- \((M, w_{11}) \models \text{loc}(1,1) \land \text{dir}(\text{right, goal}) \land \text{obs}(\text{down})\): “The location (1,1) has an obstacle perception \text{obs}(\text{down}) \text{ and one of the shortest-path directions is } \text{dir}(\text{right, goal}).”

Epistemic logic provides the syntax and semantics for modelling and reasoning about uncertainty via modalities; however, it does not model change. We will now introduce public announcement logic, which extends epistemic logic with a dynamic operator for simple monotonic change.

### 2.4.2 Public Announcement Logic (PAL)

Public announcement logic (PAL) [14] extends the syntax and semantics of epistemic logic to include public announcement events. Given an epistemic formula \(\varphi\), a public announcement \([\varphi!]\) represents the monotonic revelation of \(\varphi\).

Semantically, a public announcement formula \([\varphi!]\) is applied to an epistemic model \(M = \langle W, V \rangle\) as \(M' = M \otimes [\varphi!]\), where:
• \( W' = \{ w \in W \mid (M, w) \models \varphi \} \)
• \( V'(w) = V(w) \).

The resultant model \( M' \) now only models worlds from \( M \) that hold the revelation \( \varphi \), such that \( M' \models \square_B \varphi \).

**Example 2.4.2** (Navigation: Observing Obstacles). In this example, we model the initial situation of the navigation problem in which the agent perceives its surrounding obstacles. The following perceptions are given: \( \text{obs}(\text{down}) \) and \( \neg \text{obs}(D) \) for \( D \in \{\text{up}, \text{left}, \text{right}\} \). These perceptions are encoded by the announcement: \([\text{obs}(\text{down}) \land \neg \text{obs}(\text{up}) \land \neg \text{obs}(\text{left}) \land \neg \text{obs}(\text{right})!]\). After applying this announcement to the initial epistemic model of Example 2.4.1, the resultant model only holds two possible worlds \((1,1)\) and \((2,1)\) as highlighted in red in Figure 2.7.

![Figure 2.7: The agent’s updated situation.](image)

There are two categories of change in the epistemic logic literature: *epistemic* and *ontic* change. Epistemic change refers to changes in an agent’s knowledge or beliefs about the world, such as acquiring new information, updating beliefs based on new evidence, or retracting previously held beliefs. Ontic change, on the other hand, refers to changes in the state of the world independent of an agent’s knowledge or beliefs.
about it. This can include physical changes in the environment, the actions of other agents, or other external events.

PAL is limited to expressing monotonic epistemic change; this translates into the fact that the number of possible worlds may only be reduced as a result of a public announcement. As a result, the agent can only become more certain as public announcements occur. Since PAL semantics are strictly eliminative, PAL maintains a reasonable polynomial complexity [47] but lacks the expressivity for ontic change and expansive changes (i.e., those where uncertainty grows over time) that may occur in more dynamic domains of uncertainty. In the case of our navigation example, PAL is not expressive enough to capture the change in knowledge that occurs due to the move action as a result of its corresponding ontic change. We now formally introduce DEL, which subsumes PAL with more expressive components for change.

2.4.3 Dynamic Epistemic Logic (DEL)

To model the dynamics of uncertainty, we use DEL event models \( \varepsilon = \langle E, pre, post \rangle \), where \( E \) is a finite set of events, \( pre(e) \) is an epistemic formula that represents the precondition for the event \( e \), and \( post(e) \) is a mapping that assigns a new truth value to each proposition \( p \in P \), based on an epistemic formula \( post(e, p) \). We say that \( post \) is trivial (\( post(e) = \emptyset \)) when \( post(e, p) = p \) for all \( p \). Similar to epistemic models, event models typically have an indistinguishability relation \( R \subseteq E^2 \). In this thesis, we also utilize S5 semantics for event models that allow us to implicitly maintain indistinguishability among all events \( R = E^2 \).

The application of an event model \( \varepsilon \) to an epistemic model \( M \) is a new epistemic model \( M' = M \otimes \varepsilon \) [14]. The updated model \( M' \) is defined as \( (W', V') \) via a product update where:
\[ W' = \{ (w, e) : (w, e) \in W \times E | M, w \models \text{pre}(e) \} \]

\[ V'((w, e)) = \{ p : p \in P | M, w \models \text{post}(e, p) \} \]

In the resultant model, the worlds in \( M' \) are pairs \((w, e)\) in which the precondition of \( e \) holds in \( w \) and in which we reassign the truth values of propositions according to \( \text{post} \).

For example, a public announcement \([\varphi!]\) is captured with an equivalent single-event event model: \( \langle \{e\}, \text{pre}(e) = \varphi, \text{post}(e) = \bot \rangle \). The process of applying a DEL event model to an epistemic model is called a product update, indicating that the size of the model can potentially increase multiplicatively to represent event uncertainty. The following example models the move action from the navigation example.

**Example 2.4.3 (Navigation: Moving Right).** The move action requires ontic change to describe the change in locations and thus requires the additional expressiveness provided by DEL. The DEL event model \( \varepsilon \to = (E, \text{pre}, \text{post}) \) models the action move(right), with:

- \( E = \{ e_{xy} : (x, y) \) is a non-obstacle cell\},
- \( \text{pre}(e_{xy}) = \text{loc}(x, y) \),
- if \( x \leq 3 \), and no obstacle at \((x + 1, y)\):
  - \( \text{post}(e_{xy}, \text{loc}(x, y)) = \bot \),
  - \( \text{post}(e_{xy}, \text{loc}(x + 1, y)) = \top \),
  - \( \text{post}(e_{xy}, \text{obs}(d)) = \top \) if there is an obstacle in direction \( d \) at \((x + 1, y)\), \( \bot \) otherwise.
  - \( \text{post}(e_{xy}, \text{dir}(d, \text{goal})) = \top \) if \( d \) is the the first direction to take in a shortest path from \((x, y)\) to the goal, or \( \bot \) otherwise.
• otherwise, $\text{post}(e_{xy}) = \emptyset$.

The event model effectively captures the change in $\text{loc}$, $\text{dir}$, and $\text{obs}$ that occurs due to the $\text{move}(\text{right})$ action. After applying the event model to the resultant model from Example 2.4.2, i.e., $M'' = M' \otimes \varepsilon_{\rightarrow}$, we obtain two possible locations $(2,1)$ and $(3,1)$ that are right-adjacent to the previous possible locations. This is shown in Figure 2.8.

![Figure 2.8: After moving right, Alice considers (2,1) and (3,1) possible.](image)

We now conclude the section on epistemic logic with some final remarks on the complexity of PAL and DEL.

### 2.4.4 Computational Complexity

In this thesis, we make use of the DEL (and implicitly PAL) reasoner Hintikka’s World [15, 47]. Hintikka’s World is an open-source pedagogical tool for representing and reasoning about uncertainty using propositional epistemic logic and its dynamic variants. Hintikka’s World utilizes the tableau method to model-check epistemic formulae. Model checking non-dynamic epistemic formulae has a linear complexity concerning the size of the epistemic model and formula. Although applying DEL event models also has a linear complexity with respect to the model and event model size,
product updates may cause DEL to grow the model size exponentially as more event models are applied. In contrast, PAL, which is more restrictive, may only maintain or shrink the model, and can guarantee a maximum model size, and thus complexity, over time. It is important to note that these complexity results (from [47]) are specific to the single-agent S5 epistemic logic utilized in this thesis.

This concludes the background and state of the art chapter. This chapter introduced the navigation domain which we use as a running example for reasoning about qualitative uncertainty. The syntax and semantics of AgentSpeak and its reasoning cycle were presented, along with an extensive review of the state of the art on AgentSpeak extensions for uncertainty. This review identified a clear gap in the literature and a need for a qualitative uncertainty extension. Epistemic logic and its dynamic variants, PAL and DEL, were introduced for qualitative reasoning about uncertainty statics and dynamics. In the next chapter, we formally extend AgentSpeak with PAL and DEL to provide an expressive and idiomatic extension for uncertainty reasoning.
Chapter 3

Methodology

3.1 Introduction and Structure

In the previous chapter, we introduced the relevant background for understanding the current challenges and approaches for reasoning about uncertainty in AgentSpeak. We noted the absence of any qualitative uncertainty approaches in the existing literature, despite their potential usefulness in symbolic agent-oriented languages such as AgentSpeak, and for reasoning about uncertainty in domains such as the 2019 MAPC.

This chapter presents two novel extensions to the AgentSpeak language, PAL-AgentSpeak and DEL-AgentSpeak, which respectively integrate with two dynamic forms of epistemic logic: PAL and DEL. This chapter details the formal methodology for extending AgentSpeak’s language and reasoning cycle, such that we provide the agent with two extensions for qualitative uncertainty reasoning which provide varying levels of expressiveness for change. Throughout this chapter, these extensions will be presented according to their model creation, updating, and querying phases, a pattern...
identified in the previous chapter among existing uncertainty extension methodologies. We start with a *desired* AgentSpeak program for the 2019 MAPC navigation challenge to present a desirable form of how uncertainty should be symbolically expressed in AgentSpeak.

### 3.1.1 Uncertain Navigation in AgentSpeak

In Example 3.1.1, we use the uncertain navigation challenge presented by the 2019 MAPC to motivate a *desirable* AgentSpeak program, highlighting the minimal language extensions necessary for qualitative uncertainty modelling and reasoning. Through these extensions, the agent has been equipped with the ability to idiomatically declare uncertainty about its location, and reason about which possible directions it must navigate to get to its goal; this is something that is not available in standard AgentSpeak, nor existing extensions in the literature. Our desirable AgentSpeak program aims for minimal syntactic input from the developer by allowing for a declarative specification of uncertainty. Doing so lessens the burden on the agent developer to write and maintain the program. Following the upcoming example, we discuss the role that each extended component plays in capturing the agent’s uncertainty.

**Example 3.1.1 (Uncertain Navigation).** Listing 3.1 provides the desirable AgentSpeak program for uncertainty reasoning and is adapted from the standard AgentSpeak program provided in Listing 2.1. Extended components are presented in blue font and act as placeholders for how we would like to extend AgentSpeak for symbolic uncertainty reasoning. We assign extended semantics to these components, which we now informally introduce to the reader.
In an AgentSpeak program, the agent’s belief state is defined by a set of beliefs and
rules that define its certainty. The navigation program shown previously in Listing
2.1 declared the certainty it had of its actual location: \texttt{loc(1,1)}. Under standard
AgentSpeak semantics, belief uncertainty is implicitly captured by the weak-negation
operator \texttt{not}, however, this is an elementary operator which does not allow for explicit
reasoning about uncertainty. To achieve this, AgentSpeak is extended with two
syntactic mechanisms, \textit{ranges} and \textit{constraints}, which allow the agent to explicitly
declare its uncertainty.

As the reader will soon see, we utilize custom literal operators such as “\texttt{range(\ldots)}”
to declare uncertainty in the program, as opposed to the standard approach of
extending the language using annotations (see Section 2.3.2). We use custom literal
operators as it reads better when writing uncertainty programs, but they may interfere with existing AgentSpeak programs that utilize this syntax for other purposes. In this case, the proposed methodology and implementation must be modified by the reader to utilize annotations instead.

**Definition 3.1.1** (Ranges). A “range” is a positive literal \( \text{range}(\ell) \) whose terminology we have chosen arbitrarily to represent the agent’s range of initial uncertainty. Semantically, an initial range held by the agent’s initial belief base declares unconstrained boolean uncertainty for any positive literal \( \ell \) such that the agent is not sure whether to initially believe \( \ell \) or \( \neg \ell \) – the inner positive literal \( \ell \) or its negation defined by any range will hereinafter be referred to as a **ranged literal**.

Line 1 of Listing 3.1 declares the range of uncertainty of \( \text{loc}(X, Y) \) from \((0,0)\) to \((4,4)\). In the listing, we make use of a built-in internal action \(.ran(X, S, E)\)\(^1\), which unifies \(X\) with each value between \(S\) and \(E\). In the listing, the ranged literal \( \text{loc}(\text{none}) \) is defined on Line 2 for the sake of ensuring there is exactly one location; the use of this literal to constrain ranged uncertainty will now be discussed in more detail.

**Definition 3.1.2** (Range Constraints). The agent may want to constrain the total boolean uncertainty captured by ranges. Given a range \( \text{range}(\ell) \), constraints will simply take the form of standard AgentSpeak beliefs and/or rules that describe the truth of \( \ell \). In the case of the navigation agent, it knows that exactly 1 location is true at any given time. This is represented by the following constraints expressed within the listing:

- Mutual exclusivity of locations is modelled by the constraint on Line 3.

\(^1\).\(\text{ran}(X, S, E)\) is an alias for Jason’s internal action \(.\text{range}(X, S, E)\), so that the reader does not get confused between ranged literal definitions and the \(.\text{range}\) action.
• We eliminate the case where no locations are true ($\text{loc}(\text{none})$) on Lines 4 and 7.

• Obstacle locations (1,2)/(2,2) are eliminated on Lines 5 and 6.

In order to reason about uncertainty, the language must also provide a symbolic way for querying varying levels of belief certainty. On Line 17 of Listing 3.1, the agent utilizes a symbolic operator $\text{poss}(\ell)$ to reason about the possibility of belief $\ell$. This allows the agent in Listing 3.1 to express plan contexts such as $\text{dir}(D, \text{goal}) \mid \text{poss}(\text{dir}(D, \text{goal}))$, where the agent reasons about the uncertainty it has about the direction it must travel. There is an implicit pattern in this context, namely that the agent prefers to act on certainty $\text{dir}(D, \text{goal})$ before acting on possibilities $\text{poss}(\text{dir}(D, \text{goal}))$; this is enabled by the short-circuiting of the $\mid$ operator. This becomes an emerging pattern when reasoning about uncertainty and is further explored when looking at other applications in Chapter 5.

While executing plans to achieve its goals, the agent performs belief addition and deletion operations to revise its belief state. These additions and deletions, as shown in the plan on Line 21 will affect the current state of uncertainty; in order to define their impact on the agent’s current uncertainty, these standard syntactic components are assigned additional semantics by the extensions presented throughout this chapter.

In the previous chapter’s state of the art section, we observed that AgentSpeak’s current extensions for uncertainty reasoning extend the reasoning cycle with three critical operations: model creation, model updates, and model queries. These extensions enable the creation of an uncertainty model that reflects the initial belief state, the updating of the uncertainty model based on changes in beliefs, and the querying of the uncertainty model.
This thesis proposes the use of epistemic logic and the dynamic variants PAL and DEL to qualitatively model and reason about uncertainty. Therefore, utilizing the extended constructs described above, the initial ranges and constraints create an initial epistemic model representing the agent’s initial uncertain belief state. When adding or removing beliefs, we update the epistemic model using the appropriate dynamic constructs provided by PAL/DEL, and lastly, we query the beliefs and possibilities (via \textit{poss}) by transforming the query into a modal formulae that can be evaluated using an epistemic model.

3.1.2 Chapter Structure: An Agile Approach

This chapter proposes two AgentSpeak extensions that provide two different levels of expressiveness which are applied to different domains and use cases. The first extension, PAL-AgentSpeak, uses the dynamic mechanisms of PAL to provide semantics for belief change, whereas the second extension, DEL-AgentSpeak, utilizes DEL. The proposed extensions integrate with Hintikka’s World, a reasoning engine that supports epistemic logic and its dynamic forms PAL and DEL. The use of Hintikka’s World is most relevant to the implementation of these extensions, and will be discussed in Chapter 4. Throughout the chapter, we utilize the MAPC navigation challenge and the desirable AgentSpeak program from Listing 3.1 as running examples to demonstrate both extensions.

PAL is a less expressive form of DEL and is ideal for simpler domains. We first introduce PAL-AgentSpeak in Section 3.2, its simplicity allows the reader to become familiar with the functions and semantic integrations required by both PAL- and DEL-AgentSpeak. Additionally, PAL-AgentSpeak maintains backwards compatibility
with standard AgentSpeak programs by only applying semantics to ranged literals, i.e., those that have been explicitly defined with uncertainty.

Building off of the PAL-AgentSpeak extension, DEL-AgentSpeak provides a much more expressive language but also requires additional syntactic components to infer dynamic components from the agent’s program. Section 3.3 presents the DEL-AgentSpeak extension. Unlike PAL-AgentSpeak, we do not limit assigned semantics to ranged literals; this is due to the various technicalities involved with the integration of DEL. As such, we also show that the semantics assigned by DEL-AgentSpeak to non-ranged literals maintain backwards compatibility with standard AgentSpeak programs.

Overall, this chapter formally presents PAL-AgentSpeak and DEL-AgentSpeak, two qualitative uncertainty extensions for AgentSpeak which have varying levels of expressiveness for change. At the end of this chapter, readers will have a clear understanding of both extensions and their applicability to different classifications of domains.

3.2 PAL-AgentSpeak: An Extension for Monotonic Belief Change

In this section, PAL-AgentSpeak is formally introduced. The extension relies on PAL semantics such that public announcement events are used to model belief change. Given its simplicity, PAL-AgentSpeak will provide trivial backward compatibility with standard AgentSpeak programs by ensuring that non-ranged literals are not impacted by its assigned semantics. This section will first introduce the formalities of PAL-AgentSpeak’s model creation, update, and querying operations. At the end of
this section, these operations are integrated into AgentSpeak’s reasoning cycle through
the formal definition of operational semantics. Before diving into PAL-AgentSpeak,
we briefly justify our usage of a proposition-based epistemic reasoner used for both
PAL- and DEL-AgentSpeak.

3.2.1 Propositional vs. First-Order Epistemic Reasoners

Although first-order formalisms for PAL and DEL exist, they bring an additional
computational cost to provide a higher level of expressiveness [48]. The level of
expressiveness provided by the first-order formalism is not necessary for the proposed
AgentSpeak extensions, thus we continue with a propositional reasoner given that
performance is a critical consideration for agent systems. Higher-order epistemic
reasoners are left outside the scope of the thesis and may be explored as future work.

The formal semantics assigned by PAL-AgentSpeak and DEL-AgentSpeak through-
out this chapter will thus involve a transformation from AgentSpeak syntax into
propositional (or modal) formulae. We use a process known as propositionaliza-
tion [49] to transform predicate-based formulae (i.e., AgentSpeak belief syntax) into
propositional formulae that can be interpreted by the reasoner.

3.2.1.1 Propositionalization

The process of propositionalization is used throughout the model creation, updating,
and querying processes of PAL- and DEL-AgentSpeak to transform AgentSpeak’s
syntax so that we can delegate to the epistemic reasoner. For example, during
model creation, propositionalization transforms ranged literals and constraints into
propositional formulae, which provide a propositional description of an initial epistemic
model. We will now define a propositionalization function for standard AgentSpeak formulae.

**Definition 3.2.1** (Propositionalization). Propositionalization of a literal conjunction \( \varphi \) is the process of converting \( \varphi \) so that all literals in the expression are converted to equivalent propositional symbols, a process that is only possible when \( \varphi \) is ground (i.e., there are no free variables). Let \( \ell \) be a unique propositional symbol for the atom \( \ell \), the propositionalization of a literal conjunction \( \varphi \) is defined as:

\[
pr(\ell) = \begin{cases} 
\bot & \text{if } \ell \text{ is unground} \\
\neg pr(a) & \text{if } \ell = \sim a \text{ (strongly-negated)} \\
\ell & \text{otherwise}
\end{cases}
\]

\[
pr(\neg(\varphi')) = \neg pr(\varphi') \text{ (weakly-negated)}
\]

\[
pr(\varphi_1 \land \ldots \varphi_n) = pr(\varphi_1) \land \ldots pr(\varphi_n)
\]

Note the equivalent propositionalization of strongly- and weakly-negated formulae, which we provide brief comments on.

**Propositionalization of a Three-Valued Logic** AgentSpeak’s three-valued logic uses strong- and weak-negation to model and query beliefs following both closed- and open-world assumptions. Since we rely on a closed-world assumption with epistemic logic, both strongly- and weakly-negated AgentSpeak operators (i.e., \( \sim \) and \( not \)) collapse into the propositional negation operator \( \neg \). From this, one may think that it would no longer be possible to provide a three-valued logic using a closed-world assumption. However, a three-valued logic can be provided using the modalities of a closed-world modal logic [50]. The transformation required to evaluate a three-valued...
logic using modalities will be discussed further during the model querying operation (Section 3.2.4).

**PAL-AgentSpeak Propositionalization**  In PAL-AgentSpeak, only ranged literals will be propositionalized to ensure backwards compatibility with standard AgentSpeak. In order to determine whether a literal is ranged, we need to obtain the set of all ranged literals defined by an agent. Given a belief base \((B)\) containing initial beliefs and rules, the following function obtains all ground range literals:

\[
ranges(B) = \{ \ell \theta, \neg \ell \theta \mid B \models range(\ell) \theta \}
\]

Given the context of a set of ranged literals \(R\), we define the range-only propositionalization function which simplifies the formula (by removing any non-range literals) before using the standard function \(pr\). Let \(simp(R, \varphi) = \varphi_S\) be a trivially-defined simplification function that removes literals in \(\varphi\) which do not appear in \(R\) and provide the resulting formula \(\varphi_S\).

\[
pr_{\langle R \rangle}(\varphi_1 \land \ldots \varphi_n) = pr(simp(R, \varphi_1)) \land \cdots \land pr(simp(R, \varphi_n))
\]

The function \(pr_{\langle R \rangle}(\varphi)\) returns a propositionally-equivalent form of the formula \(\varphi\) where all non-ranged literals have been omitted. We now present the relevant function definitions for the model creation process in PAL-AgentSpeak.

### 3.2.2 Model Creation

This section presents the model creation process, which involves the semantic transformation of ranges and constraints into their corresponding propositional formulae, and
the generation of an initial epistemic model which captures the uncertainty described by the transformed formulae.

### 3.2.2.1 Extended Semantics: Ranges

Range beliefs are assigned a special propositional meaning by our extension, allowing us to formally define uncertainty of a given literal. Essentially, a range \( \text{range}(\ell) \) represents the sentence \( \ell \lor \neg \ell \) – note that although this sentence represents a tautology it is necessary for generating an appropriate epistemic model. Given a range belief \( \text{range}(\ell) \) with positive ranged literal \( \ell \), we represent the ranged belief as the following propositional sentence:

\[
\text{pr}_\text{range}_{\langle R \rangle}(\ell) = \text{pr}_{\langle R \rangle}(\ell) \lor \neg \text{pr}_{\langle R \rangle}(\ell)
\]

### 3.2.2.2 Extended Semantics: Range Constraints

Range constraints will be assigned semantics that allow us to capture the conditions for which a ranged literal is explicitly true or false. When range constraints are expressed in the form of rules, e.g., \( \ell_H : \neg \ell_B \), the rule is effectively propositionalized as \( \ell_B \rightarrow \ell_H \). However, literals contained within the rule bodies may further depend on other rules to obtain their logical consequences; thus, the propositionalization of constraint rules requires special consideration.

A naive approach involves propositionalizing all logical consequences of rules existing in the belief base, ensuring that the reasoner has access to every propositional form of all belief rules during model creation. However, the logical consequences function is computationally expensive, requiring \( O(|B|^{|R|}) \) time and space in the worst-case scenario for \( B \) total belief queries used within \( R \) recursive rule consequences
Obtaining the propositionalization of all rule consequences would significantly and unnecessarily impact performance. Instead, a more elegant solution is to rewrite formulae such that they are rule-free and exclusively expressed in terms of explicit beliefs; this brings a similar time and space complexity but will ensure that the logical consequences function is not unnecessarily invoked.

**Rewriting Rule-based Formulae** To simplify rule handling, we use a transformational process called *rewriting*. When determining the logical consequences of formulae, we find all ground beliefs that satisfy a formula, including those implicitly used by rules. By rewriting the logical consequences of a formula so that it is expressed in terms of corresponding explicit beliefs, we avoid unnecessary propositionalization of all rules. Obtaining the rule-free rewritten forms of a formula has the same time and space complexity as the logical consequences function but avoids the propositionalization of unneeded rules.

Given belief base $B$, literal conjunction $\varphi$, and MGU $\theta$ we use the notation $B \models_\phi \varphi\theta$ to represent the logical consequences of $\varphi\theta$ with respect to $B$ enabled by the rule-free formula $\phi$.

**Example 3.2.1.** In Listing 3.2, we provide a belief base $B$ with one ground belief and two rules to help demonstrate the process of rewriting. The agent would like to query $\text{dir}(D, \text{goal})$ (where $D$ is a variable term to be unified) using this belief base. We say that $B \models_\phi \text{dir}(D, \text{goal})\theta$ given $\theta = \{D \leftarrow \text{right}\}$, i.e., that $\text{dir}(\text{right}, \text{goal})$ is a consequence of $B$ given the rule-free means for which $\text{dir}(\text{right}, \text{goal})$ is a consequence of $B$: $\phi = \text{not}(\text{not}(\text{loc}(1,1)))$ (or $\phi \equiv \text{loc}(1,1)$). Using this notation, we will now define a function that rewrites the logical consequences of formulae.
1 \text{loc}(1,1).
2 \text{obs(} \text{right}) : \neg \text{not loc}(1,1).
3 \text{dir(} \text{right, goal}) : \neg \text{not obs(} \text{right}).

Listing 3.2: Sample beliefs and rules.

**Definition 3.2.2** (Rewriting Formula Consequences). Given a belief base \(B\) and literal conjunction \(\varphi\) we obtain a set of equivalent rule-free forms that make \(\varphi\) a consequence of \(B\):

\[
\text{rw}_{\langle B \rangle}(\varphi) = \{ \phi \mid B \models_{\phi} \varphi \}
\]

This function will now be used to obtain all constraints expressed by the agent.

**Finding Range Constraints**  Given belief base \(B\) and \(R = \text{ranges}(B)\) containing ranged literals defined by the agent, we obtain a set of tuples \((\ell, \varphi)\) where \(\ell\) is a (positive or negated) ranged literal and \(\varphi\) is the rule-free condition for \(\ell\).

\[
\text{cons}(R, B) = \{ (\ell, \varphi) \mid \ell \in R \text{ and } \varphi \in \text{rw}_{\langle R \cup B \rangle}(\ell) \}
\]

Note that we conjoin \(R \cup B\) such that formulae can also be grounded with ranged literals. Given a pair \((\ell, \varphi)\) where \(\ell\) is a (positive or negated) ranged literal and \(\varphi\) is the truth condition for \(\ell\), we propositionalize the constraint as follows:

\[
\text{pr}_{\text{con}_{\langle R \rangle}}(\ell, \varphi) = \begin{cases} 
\text{pr}_{\langle R \rangle}(\ell) & \varphi = \top \\
\neg \text{pr}_{\langle R \rangle}(\ell) & \varphi = \bot \\
\text{pr}_{\langle R \rangle}(\varphi) \rightarrow \text{pr}_{\langle R \rangle}(\ell) & \text{otherwise}
\end{cases}
\]
In order to obtain all propositionalized ranges and constraints expressed by the agent, we introduce the following function.

**Collecting Propositional Constraints**  Given an initial belief base $B$ and set of ranged literals $R$, we obtain all propositional sentences that describe the agent’s initial state of uncertainty as follows:

$$all\_cons(B, R) = \{pr\_range_{R}(\ell) \mid \ell \in R\} \cup \{pr\_con_{R}(\ell, \phi) \mid (\ell, \phi) \in cons(R, B)\}$$

Through these propositional sentences, we are able to generate an initial model that captures the uncertainty described by the agent’s ranges and constraints.

### 3.2.2.3 Generating an Epistemic Model

Given a set of propositional sentences $S$ obtained via the relevant propositionalization of initial ranges and constraints, and a function $at(S)$ that collects all atomic propositional symbols used in $S$, we generate a set of possible worlds by obtaining all solutions that satisfy $S$ via the following function:

$$gen\_worlds(S) = \{w \in 2^{at(S)} \mid w \models \bigwedge_{s \in S} s\}$$

---

2In the case where no ranges or constraints are provided ($S = \emptyset$), a valid model containing a single empty world is formed.
We create a complete initial epistemic model \( gen_{\text{model}}(S) = \langle W, V \rangle \) from the sentences in \( S \) as follows:

\[
W = gen_{\text{worlds}}(S)
\]
\[
V(w) = w
\]

Due to the need to find all satisfiable solutions to the propositional sentences, the model generation process has a worst-case time and space complexity of \( O(2^{\text{at}(S)}) \). We further discuss the implications of the model generation process on practical usage and its relevant trade-offs in Chapter 6. The following example demonstrates the model creation process using our running navigation example.

**Example 3.2.2 (Model Creation: Navigation Program).** Using the uncertain navigation program provided in Listing 3.1, the ranges defined by the agent are provided as follows:

- \( \text{range}(\text{loc}(\text{none})) \), \( \text{range}(\text{loc}(0, 0)) \), . . . , \( \text{range}(\text{loc}(4, 4)) \)

These ranges are propositionalized (via \( pr_{\text{range}} \)) as follows:

- \( \neg\text{loc}(\text{none}) \lor \text{loc}(\text{none}) \)
- \( \neg\text{loc}(0, 0) \lor \text{loc}(0, 0), \ldots, \neg\text{loc}(4, 4) \lor \text{loc}(4, 4) \)

The agent’s constraints are propositionalized (via \( pr_{\text{con}} \)) as follows:

- \( \neg\text{loc}(0, 0) \land \cdots \land \neg\text{loc}(4, 4) \rightarrow \text{loc}(\text{none}) \)
- \( \text{loc}(0, 1) \rightarrow \neg\text{loc}(0, 0), \ldots, \text{loc}(4, 4) \rightarrow \neg\text{loc}(0, 0) \)
- . . .
• $\text{loc}(0,0) \rightarrow \neg \text{loc}(4,4)$, \ldots, $\text{loc}(4,3) \rightarrow \neg \text{loc}(4,4)$

• $\neg \text{loc}$(none), $\neg \text{loc}(1,2)$, $\neg \text{loc}(2,2)$

The resulting set of propositional sentences infer an initial epistemic model with the following set of worlds:

• $W = \{ \{ \text{loc}(0,0) \}, \{ \text{loc}(0,1) \}, \ldots, \{ \text{loc}(4,4) \} \}$

Due to the constraints we placed on the ranges each world valuation captures a unique location, additionally, no world valuations model location $(1,2)$ or $(2,2)$ due to the negated initial beliefs held by the agent. We say that initially the agent considers all locations possible except for $(1,2)$ and $(2,2)$.

This concludes the model creation process for PAL-AgentSpeak, which introduced the usage of ranges and constraints to describe and create an initial epistemic model. The functions and components will be integrated into the PAL-AgentSpeak operational semantics in Section 3.2.5. In the next section, we introduce the functions required for the model update process.

### 3.2.3 Model Updating

In AgentSpeak, belief updates occur as a result of changes in perceptions or explicit belief addition/deletion operations executed within the body of a plan. All changes to the belief base are represented as belief addition $+b$ or deletion $-b$ events. This section describes the extended semantics assigned by PAL-AgentSpeak to the belief addition and deletion events such that the uncertainty captured by the epistemic model gets updated accordingly. To preserve backwards compatibility, these semantics only apply to the addition or deletion of ranged literals. We bring forth our running example.
**Example 3.2.3.** The initial epistemic model created for the running example was shown in Example 3.2.2. After the epistemic model is created and the agent starts its execution, the agent immediately perceives its surrounding environment.

- The agent (1,1) perceives the obstacle at (1,2). The perception is represented as a sequence of belief addition: $+\text{obs}(\text{down})$ and $+\sim\text{obs}(\_)$ for all other directions.

Since the agent relies on obstacle perceptions to eliminate impossible locations, we must model $\text{obs}(\_)$ alongside their appropriate locations in the initial epistemic model. Listing 3.3 must therefore be appended to Listing 3.1, which retroactively includes the $\text{obs}$ ranges and constraints in the model creation process. The $\text{.member(Var, List)}$ internal action is used to unify $\text{Var}$ with each element in $\text{List}$.

```prolog
1 range(\text{obs}(D)) :- \text{.member}(D, [\text{up, down, left, right}]).
2 // ... Rule definitions for obs(_) :- loc(_).
3 \sim\text{obs}(D) :- \text{not}(\text{obs}(D)). // Closed World
```

Listing 3.3: Add $\text{obs}$ to the epistemic model, so that we may filter possible locations by obstacles.

In PAL-AgentSpeak, public announcements model belief changes for any ranged literals. Public announcements are monotonic operations for modelling epistemic change, and thus updates in PAL-AgentSpeak are limited to only supporting domains with monotonic epistemic change. These changes will be captured via the belief addition operator. Later, we discuss the implication of providing support for non-monotonic change via a belief deletion operator in PAL-AgentSpeak and why the deletion operator will not be supported for ranged literals.
3.2.3.1 Monotonic Belief Change

Monotonic belief change of a literal $\ell$ is modelled by applying a public announcement to the current model $M$: $M \otimes [\ell!]$. In PAL-AgentSpeak, this is invoked via a belief addition event or operation: $+\ell$. The resulting model $M'$ only maintains worlds that model $\ell$, thus modelling the addition of belief $\ell$: $M' \models \Box_B \ell$.

Example 3.2.4 (Belief Addition: Obstacles). We refer to the initial epistemic model created in Example 3.2.2, which has been modified to include the relevant obs ranges and constraints, as $M_I$. When the agent perceives the obstacle at its initial location, the belief additions $+\text{obs}(\text{down})$, $+\neg\text{obs}(\text{left})$, etc., occur. The corresponding announcement is applied to the model as follows:

$$M' = M_I \otimes [\text{obs}(\text{down})!] \otimes [\neg\text{obs}(\text{left})!] \otimes \ldots$$

The resulting model $M'$ is thus restricted to the following two worlds ($W'$):

- $W' = \{\{\text{loc}(1,1), \text{obs}(\text{down})\}, \{\text{loc}(2,1), \text{obs}(\text{down})\}\}$

The assignment of public announcement semantics to the belief addition operation allows for the easy specification of monotonic belief change in PAL-AgentSpeak. However, the running navigation example is a complex domain consisting of other scenarios that result in non-monotonic belief changes. For example, the action move($\text{right}$) produces ontic effects that change the agent’s location based on its movement, e.g., the agent moves from (1,1) to (2,1). This change is non-monotonic as it invalidates existing beliefs due to the agent’s change in situation. In the next subsection, we discuss non-monotonic changes in more detail.
3.2.3.2 Non-Monotonic Belief Change

Since PAL is restricted to monotonic epistemic change, PAL-AgentSpeak does not provide support for non-monotonic change; as such, belief deletions are implicitly not supported. However, for the sake of completeness, this section presents two explored workarounds which may allow us to capture non-monotonic change using PAL by: 1) reverting back to the initial epistemic model, and 2) regenerating the initial epistemic model. Due to the ad hoc nature of these workarounds, it is unclear whether these operations are sound. Rather, the ad hoc approaches exemplify the steps required to model a complex domain in PAL-AgentSpeak without a more expressive language extension such as DEL-AgentSpeak.

Workaround: Reverting Back to the Initial Model  One approach to combat the lack of support for non-monotonic change in PAL is to revert back to the original initial epistemic model after monotonic changes have been applied. Reverting back to the initial model to accommodate a single non-monotonic change, e.g., $-\ell$, would erase all monotonic changes applied the epistemic model not just those associated with $\ell$. To accommodate this, we can reapply any beliefs maintained after the deletion. This is demonstrated in the following example.

Example 3.2.5. Given an initial epistemic model $M$ with ranged literals \{$\ell_1, \ell_2$\} and an initial belief base $B$, we apply the sequence of belief changes to both the epistemic model and belief base: $+\ell_1, +\ell_2$.

- $M' = M \otimes [\ell_1!] \otimes [\ell_2!]$
- $B' = B + \ell_1 + \ell_2$
When a belief deletion \(-\ell_1\) occurs, we revert back to the initial epistemic model \(M\) and reapply all remaining beliefs \((B''')\) such that:

- \(B'' = B' - \ell_1 = \{\ell_2\}\)
- \(M'' = M \otimes [\ell_2!]\)

Reverting back to the initial epistemic model may be computationally feasible, since the initial model can be cached during the model creation process. Unfortunately, reverting back to the initial model does not allow for the removal of initial beliefs contained within the initial model, and is thus not a sound operation. In this case, we look at a more expensive process that involves the regeneration of the epistemic model.

**Workaround: Regenerating the Model** When a belief deletion occurs, a new initial epistemic model can be generated to accommodate the change in beliefs and the impact that it has on ranges and constraints. This approach is computationally infeasible due to the worst-case exponential time and space complexity associated with model creation; as such, it is not appropriate given the high frequency of update operations that occur with agents. In Chapter 6, we evaluate both the model creation and the DEL-AgentSpeak update operations. The DEL-AgentSpeak update operation is sufficiently expressive for non-monotonic operations and does so in a computationally feasible way, notably due to the lack of satisfiability (SAT) solving in the update process.

We briefly present how these workarounds may be used to model the \texttt{move(right)} action we discussed earlier, which had ontic effects that resulted in non-monotonic change.
Example 3.2.6 (Non-monotonic Change: Moving Locations). Recall Example 3.2.3. After the agent moves right from location (1,1), the following perception events occur: \{+moved(right), -obs(down), +obs(down), \ldots \}. Note that the deletion and addition of \(\text{obs}\) perceptions indicate a change in locations, despite there being no net effect on beliefs. In this case, the deletions trigger a reversion back to the initial model, but results in no net change since the obstacle perceptions are the same for \(\text{loc}(1,1)\) and \(\text{loc}(2,1)\). We require an additional mechanism to account for the event \(+\text{moved}(right)\) and the corresponding non-monotonic change of possible locations. To achieve this, we devise an ad hoc strategy.

Assume the existence of non-ranged beliefs which model all previous possible locations: \{\(prev\text{Loc}(1,1), prev\text{Loc}(2,1)\)\}; these happen to also be our current locations. Given we have moved right, Listing 3.4 presents a plan that manually eliminates any current possible locations that are not right-adjacent to a previous possible location.

```
1  + moved(right)
2    <- for(poss(loc(X,Y)) & not(prevLoc(X+1,Y)) { 
3      +~loc(X,Y);
4    }.
```

Listing 3.4: The workaround for moving right in PAL-AgentSpeak.

The listing utilizes the internal action \(\text{for}\) to iterate current possible locations provided via PAL-AgentSpeak’s \(\text{poss}\) operator formally introduced in Section 3.2.4. After the execution of this plan, the resulting epistemic model thus considers the possible location: \{\(\text{loc}(2,1)\)\}

This ad hoc strategy is counter-intuitive, as it places most of the burden on the developer to manage its own uncertainty, and instead should be managed by the extension. Additionally, in order to generalize this approach to all domains and non-monotonic changes, additional worlds must be generated at model creation time.
to anticipate all possible changes to uncertainty that may occur during execution; this is because public announcements only permit eliminative changes to the model. This becomes impractical for the agent developer since it is difficult and computationally infeasible to anticipate all events and effects at initialization time, but is a direct consequence of using PAL. Alternatively, the DEL-based AgentSpeak extension presented in Section 3.3 provides the expressability to effectively capture ontic and other non-monotonic changes without needing to anticipate them during model creation.

Using the workarounds presented in this section, non-monotonic change is captured through belief deletion semantics and ad hoc plans. Unfortunately, the workarounds presented are neither sound nor feasible. As such, PAL-AgentSpeak does not provide support for non-monotonic change, and thus does not provide a belief deletion operation for ranged literals. Instead, the more expressive DEL-AgentSpeak extension presented in Section 3.3 should be used.

This concludes the discussion of monotonic and non-monotonic change in PAL-AgentSpeak. Before continuing to the next section, we discuss a niche but important consideration that the reader may be wondering. If the agent can reason about its possibilities via \( \text{poss} \) as was shown in Listing 3.4, can it also perform \( +\text{poss}(\ell) \) and \( -\text{poss}(\ell) \) operations?

### 3.2.3.3 Changing Possibilities

As part of the PAL-AgentSpeak extension, the agent is able to query possibilities via a special literal \( \text{poss}(\ell) \) — this will be discussed in Section 3.2.4. As a result, the agent may think that it can add or remove possibilities via \( +\text{poss}(\ell) \) and \( -\text{poss}(\ell) \). Since a possibility is simply a modality derived from the agent’s possible worlds, a
Possibility addition $+\text{poss} (\ell)$ is not well-defined in terms of how it would impact the possible worlds to allow for the possibility of $\ell$.

On the other hand, deletion of a possibility $-\text{poss} (\ell)$ in PAL-AgentSpeak is not necessary as it can be sufficiently expressed via belief addition of the corresponding negation: $+\sim \ell$. In essence, elimination of a possibility ($\neg \Diamond_B p$) is logically equivalent to a negated belief ($\Box_B \sim p$). Figure 3.1 proves the equivalence of these statements, thus showing that it is not necessary to express the deletion of possibilities when negated beliefs can be added.

Figure 3.1: Proof (by contradiction): $\neg \Diamond_B \ell$ is equivalent to $\Box_B \sim \ell$.

Explicitly adding or removing possibilities are not possible or convenient semantic operations for the extension to provide and are thus not provided as part of the proposed AgentSpeak extension.

In the next subsection, we briefly discuss another topic for consideration. Belief event generation involves PAL-AgentSpeak’s integration of the current epistemic model with belief addition or deletion events in AgentSpeak.

### 3.2.3.4 Generating Belief Events

Belief addition or deletion events are generated to inform the agent of a change in beliefs. Belief events, like achievement goal and test goal events, allow the agent to plan and act accordingly. Under standard semantics, introducing belief events to the
agent is a trivial process; the events are simply generated as a result of explicit belief additions and deletions in the belief base. Changes to implicit beliefs (i.e., due to rule consequences) do not generate belief events due to the computational impact of obtaining rule consequences [1]. Consider the following example.

**Example 3.2.7.** Let $b$ be an explicit belief literal stored in the belief base and $r : - b$ be a implicit belief rule that depends on a belief $b$. Explicit belief additions or deletions of $b$ will generate the corresponding belief events: $+b$ and $-b$. Conversely, no belief events will be generated for the implicit belief $r$ despite changes to $b$.

Due to our integration with an epistemic reasoner, model checking is a process that is computationally complex. Similar to how implicit belief formulae do not generate belief events under standard AgentSpeak semantics, PAL-AgentSpeak (and DEL-AgentSpeak) will not generate belief events for belief or possibility changes obtained via model checking. The proposed AgentSpeak extensions simply maintain the original behaviour of AgentSpeak in this sense, only introducing belief events for explicit belief additions or deletions.

Before presenting PAL-AgentSpeak’s model querying component, we conclude this subsection with a brief remark on belief consistency.

**Belief Consistency** AgentSpeak permits inconsistent beliefs in the belief base for computational complexity purposes, delegating the task of belief consistency maintenance to the agent and its developer. This means that it is possible to simultaneously believe $\ell$ and its strongly-negated form $\neg \ell$. Due to the nature of epistemic logic, it would not be logically sound to simultaneously believe a proposition and its negation: $\Box_B (p \land \neg p)$. An epistemic model becomes invalid when it is created or updated with a logical contradiction as it will no longer provide valid results when
determining entailment of formulae. Following suit with AgentSpeak, PAL-AgentSpeak delegates the task of maintaining the beliefs and rules to the developer. Consistency must be strictly maintained for ranged literals as it is critical to the proper operation of the agent.

In the next section, PAL-AgentSpeak transforms belief and possibility queries into modal formulae, effectively delegating evaluation to the epistemic reasoner.

3.2.4 Model Querying

Once an epistemic model is available, we are able to take advantage of the epistemic modalities $\Box_B$ and $\Diamond_B$. Under standard AgentSpeak semantics, belief queries are literal expressions that query the belief base (e.g., plan contexts); however, these queries are only equipped to express certain belief (or lack thereof, with not). PAL-AgentSpeak extends the standard behaviour by: 1) enabling the agent to express the possibility of belief $\ell$ via a special form: $\text{poss}(\ell)$, and 2) redirecting the evaluation of queries containing ranged literals to the epistemic reasoner and non-ranged literals to the standard belief base.

3.2.4.1 Querying Possibilities

There are three syntactic components where an agent queries its beliefs:

1. Plan Contexts: a conjunction of literals that act as a precondition for the applicability of a plan.

2. Test Goals: test goals are single-literal queries invoked during the execution of a plan body.

3. Rules: a rule body is a conjunction of belief literals.
PAL-AgentSpeak provides the necessary semantics to allow plan contexts and test goals to reason about possibilities via the use of $poss(\ell)$ formulae — for example, we can express the context: $\ell_1 \in poss(\ell_2)$, and test goal: $?poss(\ell_3)$. On the other hand, belief rules will be used in tandem with the epistemic reasoner to infer $poss$ formulae from the implications expressed by belief rules — for example, we can use an existing belief rule $r :- \varphi$ to also infer possibilities: $poss(r) :- poss(\varphi)$. In order to allow for this, $poss$ can not be expressed within the rule body formulae $\varphi$. We will now briefly introduce the reader to how standard belief queries are evaluated in AgentSpeak.

3.2.4.2 Evaluation of Standard Belief Queries

Under standard AgentSpeak semantics, the evaluation of a belief query formula $\varphi_Q$ is represented by the following function given belief base $B$:

$$Bel_{(B)}(\varphi_Q) \equiv B \models \varphi_Q^{\theta}$$

Thus, we say that $\varphi_Q$ is a belief as long as it is a consequence of $B$ (w.r.t. MGU $\theta$). Implicitly, this definition also utilizes any available rules in the belief base to evaluate the consequences of any literals. For example, given a query literal $\ell_Q$ and the belief rule $\ell_Q :- \varphi$, we say that $Bel_{(B)}(\ell_Q)$ is implicitly true as long as $B \models \varphi$, i.e.: $Bel_{(B)}(\varphi) \rightarrow Bel_{(B)}(\ell_Q)$. We explicitly point out this implication as we must ensure consistent evaluation of rules when integrating with the epistemic reasoner.
3.2.4.3 PAL-AgentSpeak Belief Querying

We will now integrate the belief function $Bel$ with an epistemic reasoner — we start by integrating the evaluation of certain beliefs and later introduce the evaluation of possibilities.

Even though PAL-AgentSpeak relies on the epistemic reasoner to provide the evaluation of belief/possibility formulae, we still require the agent’s belief base $B$ and set of ranged literals $R$ to ground query formulae before they are propositionalized and evaluated by the reasoner. The PAL-AgentSpeak belief function will be denoted by $Bel_{(B,M,R)}(\varphi_Q)$, since the function’s evaluation of query $\varphi_Q$ is contextually dependant on $B$, $R$, and the agent’s current epistemic model $M$. A naive definition is provided as follows:

$$Bel_{(B,M,R)}(\varphi_Q) \equiv M \models B pr_{(R)}(\varphi_Q \theta) \text{ s. th. } (R \cup B) \models \varphi_Q \theta$$

This initial belief definition grounds the query $\varphi_Q$ by obtaining the logical consequences of ranged literals and the belief base (i.e., $R \cup B$) via MGU $\theta$. The expression $\varphi_Q \theta$ is ground and can now be propositionalized and evaluated by the epistemic reasoner. Since the propositionalization function filters out non-ranged literals, this definition preserves backwards compatibility — a query containing non-ranged literals will only be evaluated based on logical consequences with the belief base and will not be propositionalized for evaluation using the reasoner.

**Example 3.2.8.** Given belief base $B = \{dir(left), dir(right) :- loc(1,1)\}$, ranged literal set $R = \{loc(1,1)\}$, and an arbitrary epistemic model $M$, we evaluate the following queries:

- $Bel_{(B,M,R)}(loc(X,Y)) \equiv M \models B loc(1,1) \text{ s. th. } (R \cup B) \models loc(1,1)$
• $\text{Bel}_{(B,M,R)}(\text{dir(left)}) \equiv M \models \Box_B \top \text{ s. th. } (R \cup B) \models \text{dir(left)}$

• $\text{Bel}_{(B,M,R)}(\text{dir(right)}) \equiv M \models \Box_B \top \text{ s. th. } (R \cup B) \models \text{dir(right)}$

The first query $\text{loc}(X,Y)$ is grounded using the ranged literal $\text{loc}(1,1)$, propositionalized, and evaluated using the epistemic reasoner. The second query $\text{dir(left)}$ is a non-ranged literal which happens to be a consequence of $B$. Since this is a non-ranged query literal, the propositionalization function returns the tautology $\top$ so that the evaluation of the second query does not get impacted by the current epistemic model.

On the third query $\text{dir(right)}$, our naive definition fails to integrate the proper evaluation of rules. Ideally, $\text{dir(right)}$ should be true as a result of $M \models \Box_B \text{loc}(1,1)$ since the rule definition for $\text{dir(right)}$ depends on the ranged literal $\text{loc}(1,1)$. Instead, the naive definition presented will not evaluate rule consequences using the reasoner and will thus incorrectly say that $\text{dir(right)}$ is believed regardless of the epistemic model.

To address the improper integration of rules, we must evaluate the logical consequence formulae that lead to entailment of a query $\varphi_Q$ — this can be achieved via the rewriting function $\text{rw}$ (Definition 3.2.2) and leads us to the following belief function definition.

$$\text{Bel}_{(B,M,R)}(\varphi_Q) \equiv M \models \Box_B \text{pr}_{(R)}(\varphi) \text{ s. th. } \varphi \in \text{rw}_{(B\cup R)}(\varphi_Q)$$

**Example 3.2.9.** Continuing from the previous example, we obtain the correct evaluation of the following queries with the updated function definition:

• $\text{Bel}_{(B,M,R)}(\text{loc}(X,Y)) \equiv M \models \Box_B \text{loc}(1,1) \text{ s. th. } \text{loc}(1,1) \in \text{rw}_{(B\cup R)}(\text{loc}(X,Y))$

• $\text{Bel}_{(B,M,R)}(\text{dir(left)}) \equiv M \models \Box_B \top \text{ s. th. } \text{dir(left)} \in \text{rw}_{(B\cup R)}(\text{dir(left)})$
\( \bullet Bel_{\langle B,M,R \rangle}(\text{dir(right)}) \equiv M \models □_B \text{loc}(1,1) \text{ s. th. } \text{loc}(1,1) \in rw_{\langle B \cup R \rangle}(\text{dir(right)}) \)

As with the previous example, the first two queries are correctly evaluated by the updated belief function. The updated function now also ensures the third query \( \text{dir(right)} \) is evaluated based on the formulae that leads to its logical consequences, i.e., \( \text{loc}(1,1) \).

We further evolve the belief function to support possibility literals, i.e., \( \text{poss}(\ell) \) given some literal \( \ell \), in query formulae. The final form of the PAL-AgentSpeak belief function is provided as follows:

\[
Bel_{\langle B,M,R \rangle}(\varphi_Q) \equiv \begin{cases} 
M \models □_B \text{pr}_{\langle R \rangle}(\varphi) \text{ s. th. } \varphi \in rw_{\langle B \cup R \rangle}(\ell_Q) & \text{when } \varphi_Q = \ell_Q (\neq \text{poss}(\_)) \\
M \models ◇_B \text{pr}_{\langle R \rangle}(\varphi) \text{ s. th. } \varphi \in rw_{\langle B \cup R \rangle}(\ell_Q) & \text{when } \varphi_Q = \text{poss}(\ell_Q) \\
Bel_{\langle B,M,R \rangle}(\ell_1\theta) \land \cdots \land Bel_{\langle B,M,R \rangle}(\ell_n\theta) & \text{when } \varphi_Q = \ell_1 \land \cdots \land \ell_n
\end{cases}
\]

In this definition, we recursively evaluate the individual literals \( \ell_Q \) from the query conjunction \( \varphi_Q \) using their corresponding modalities. Note, when \( \ell_Q \) is a conjunction of multiple literals, a common MGU \( \theta \) must ground all beliefs. Recall from previous iterations of this belief function that the propositionalization function appropriately handles the evaluation of ranged/non-ranged literals.

**Example 3.2.10.** Continuing from our earlier examples, we evaluate the following queries using the final iteration of the PAL-AgentSpeak belief function. For demonstration purposes this example will introduce different belief queries, but note that all queries from the previous examples are still evaluated correctly.

\( \bullet \text{poss(} \text{loc}(X,Y)) \)

\[- M \models ◇_B \text{loc}(1,1) \text{ s. th. } \text{loc}(1,1) \in rw_{\langle B \cup R \rangle}(\text{loc}(X,Y)) \]
• \text{poss}(\text{dir}(\text{left}))

- \( M \models \lozenge_B \top \text{ s. th. } \text{dir}(\text{left}) \in rw_{(B \cup R)}(\text{dir}(\text{left})) \)

• \text{dir}(\text{left}) \land \text{poss}(\text{dir}(\text{right}))

- \( M \models \Box_B \top \text{ s. th. } \text{dir}(\text{left}) \in rw_{(B \cup R)}(\text{dir}(\text{left})) \), and
- \( M \models \lozenge_B \text{loc}(1,1) \text{ s. th. } \text{loc}(1,1) \in rw_{(B \cup R)}(\text{dir}(\text{right})) \)

The first query evaluates the possibility of \text{loc}(1,1) via modality \( \lozenge_B \). The second query attempts to query the possibility of a non-ranged belief base literal \text{dir}(\text{left}). There is no utility in expressing the second query because evaluation of this non-ranged literal does not rely on the epistemic reasoner – as long as \text{dir}(\text{left}) is explicitly held in the belief base, it will always be possible. The last query demonstrates a conjunction containing two non-ranged literal queries: \text{dir}(\text{left}) (evaluated as described in previous examples), and \text{poss}(\text{dir}(\text{right})) which relies on the epistemic model to evaluate its rule containing the ranged literal \text{loc}(1,1).

Notice that in the last query \text{poss}(\text{dir}(\text{right})), possibilities are inferred using existing belief rules – this is why we do not permit \text{poss} literals in rule bodies. We now provide a proof to show that using an epistemic reasoner to infer both certain and possible beliefs via belief rules is logically valid.

**Definition 3.2.3** (Proof: Using Rules to Infer Possibilities). Under standard Agent-Speak semantics, literals and/or rules (\( \varphi \)) contained by a belief base are inherently believed by the agent and thus wrapped with a belief modality: \( \Box_B \varphi \) — this is indicated by the original AgentSpeak belief function \( Bel \), presented earlier. Given a rule \( q \vdash p \) (i.e., \( p \rightarrow q \)) contained within a belief base, it follows that the agent believes the rule’s inference: \( \Box_B (p \rightarrow q) \). Given this belief rule, we provide proofs
to show whether the following are logically valid when using the epistemic model to evaluate formulae:

1. As per standard AgentSpeak, belief of the antecedent implies belief of the consequent: \[(\Box_B (p \rightarrow q) \land \Box_B p) \rightarrow \Box_B q\]

2. PAL-AgentSpeak uses existing belief rules to infer possibilities, thus, possibility of the antecedent implies possibility of the consequent: \[(\Box_B (p \rightarrow q) \land \Diamond_B p) \rightarrow \Diamond_B q\].

The two proofs in Figure 3.2 demonstrate that both are valid inferences via proof by contradiction.

Figure 3.2: Proof (by contradiction): given a belief rule \[\Box(p \rightarrow q)\], certain belief (left) or possibility (right) of \(q\) infer \(p\) is certain (left) or possible (right). Labels \((w)\) and \((v)\) denote the formulas modelled by worlds in an arbitrary epistemic model.

This subsection concludes with brief remarks regarding AgentSpeak’s three-valued querying logic and the use of higher-order belief queries.

**Preserving AgentSpeak’s Three-Valued Logic for Querying**  AgentSpeak’s three-valued logic for querying allows the agent to query 1) positive literals \(\ell\), 2)
strongly negated literals \( \sim \ell \), and 3) weakly negated literals \( not(\ell) \). The epistemic reasoner relies on a propositional closed-world assumption, i.e., by modelling either \( \ell \) or \( \neg \ell \). Fortunately, the three-valued logic can continue to be provided in PAL-AgentSpeak via modalities; the equivalent modal formulae are shown as follows:

- **(Positive):** \( \ell \equiv \Box_B \ell \)
- **(Strong Negation):** \( \sim \ell \equiv \Box_B \neg \ell \)
- **(Weak Negation):** \( not(\ell) \equiv \neg \Box_B \ell \)

**Higher-Order Beliefs** In epistemic logic one may want to express higher-order belief formulae, such as \( \Box_B \Box_B \varphi \). Expressing a higher-order belief formula would allow an AgentSpeak agent to be introspective about their own beliefs. Recall that there are different semantic systems of epistemic logic and that S5 semantics are used for the presented AgentSpeak extensions. When using S5 semantics, higher-order modalities are implicit and do not need to be explicitly expressed. Thus, there is no need to provide syntactic support for higher-order beliefs in AgentSpeak. The proof shown in Figure 3.2 demonstrates the equivalence between a belief formula \( \Box_B p \) and its higher-order form \( \Box_B \Box_B p \). As part of the S5 system, axiom (4): \( \Box_B \varphi \rightarrow \Box_B \Box_B \varphi \) can be recursively applied to this proof to show that the same is true for other higher-order belief formulae e.g., \( \Box_B \Box_B \Box_B \varphi \), etc. [52].

In the next subsection, we marry the concepts and functions from the model creation, model update, and model querying to provide the operational semantics for PAL-AgentSpeak.
Figure 3.3: Proof (by contradiction): $\Box_B p$ is equivalent to $\Box_B \Box_B p$ under an S5 system.

3.2.5 Operational Semantics

The operational semantics provides a marriage of all of the concepts we introduce with our extension and provide insight into how the extension operates and integrates into the AgentSpeak reasoning cycle.

**Definition 3.2.4 (PAL-AgentSpeak Configuration).** Operational semantics are presented as semantic rules defining transition relations between PAL-AgentSpeak configurations: $\langle ag, C, T, S \rangle_{PAL}$. The model creation phase discusses the initialization of this configuration such that it now includes an epistemic model capturing the uncertainty of the agent. For brevity, PAL-AgentSpeak’s configuration extends the standard AgentSpeak configuration (Definition 2.3.1) with the following sub-components of the agent $ag$ component:

- $ag_R$ holds the set of all ranged literals,
- $ag_M$ stores the current epistemic model.

Since PAL-AgentSpeak appends to the standard semantic rules of AgentSpeak, any additions to the rules will be boxed. We start with the operational semantics of the model creation phase.
3.2.5.1 Model Creation

The model creation phase initializes the agent’s configuration \( \langle ag, C, T, S \rangle_{PAL} \). In the standard AgentSpeak configuration, \( ag_B \) is the agent’s belief base. During model creation, \( ag_B \) is populated with the agent’s initial beliefs and rules. In PAL-AgentSpeak, model creation initializes the extended configuration where: \( ag_R = ranges(ag_B) \) contains the set of ranged literals and the initial epistemic models is created: \( ag_M = gen\_model(S) \) given \( S = all\_cons(ag_B, ag_R) \).

Due to the expensive operations performed during model creation, i.e., SAT solving and obtaining logical consequences, the worst-case time and space complexity will be exponential in nature. The formal complexity of this operation and how it impacts the feasibility of the extension in practice is further explored in Chapters 4 and 6.

3.2.5.2 Model Updating

The operational semantics for model updating are presented separately for belief addition and deletion. The overall objective of the model updating phase is to simultaneously update the belief base \( ag_B \) and the current epistemic model \( ag_M \) so that the proper entailment is provided during the model querying stage.

**Belief Additions** In the case of belief additions \( +b \), we add the belief to the belief base \( (ag_B + b) \) and apply a public announcement modelling the new belief (i.e., \([b!]\)) to the current epistemic model. This is shown by the semantic rule \( AddBel^* \) (a modified version of the standard AgentSpeak \( AddBel \) semantic rule) in Figure 3.4. Recall that additions or modifications to the standard semantic rules are boxed.

In PAL, applying a public announcement to an epistemic model is a linear operation with respect to the number of worlds in the model. In essence, the model update
Belief Deletions Due to the non-monotonic nature of belief deletions and the lack of a sound and efficient workaround, deletion operations will not be supported by PAL-AgentSpeak. This was extensively discussed in Section 3.2.3. The semantic rule shown in Figure 3.5 ensures that the deletion $-b$ of a ranged belief $b$ does not impact the belief base or epistemic model; additionally, the deletion does not place a belief deletion event in $C_E$ as no deletion operation occurs. This does not impact non-ranged literals.

In the next subsection, we introduce the operational semantics for transforming and delegating the evaluation of belief queries to the epistemic model.
3.2.5.3 Model Querying

This subsection presents the updated operational semantics for PAL-AgentSpeak which use the updated belief function $Bel_{(B,M,R)}$ to evaluate belief and possibility queries expressed in plan contexts and test goals. In AgentSpeak, plan contexts are evaluated when finding applicable plans (i.e., plans that are applicable for execution), and test goals are evaluated during the execution of a plan body.

**Finding Applicable Plans** Plan contexts are belief preconditions that must be entailed in order for a plan to execute. Relevant plans (i.e., plans that are relevant to handling the agent’s current goal) are referred to applicable plans when their contexts are believed by the agent. We define the following function, which, given a set $RP$ of relevant plan tuples $(p, \theta)$ containing a plan $p$ (let $Ctx(p)$ be the context of $p$) and its corresponding trigger unifier $\theta$, will find all applicable plans using the agent’s belief base $B$, epistemic model $M$, and set of ranged literals $R$:

$$AppPlans(RP, B, M, R) = \{(p, \theta \circ \theta') \mid (p, \theta) \in RP \text{ and } Bel_{(B,M,R)}(Ctx(p)\theta\theta')\}$$

The resulting set contains tuples $(p, \theta'')$ where $p$ is an applicable plan, and $\theta''$ is a unifier composed of $\theta$ and $\theta'$ which make $p$ both relevant and applicable. The semantic rules shown in Figure 3.6 override the standard AgentSpeak rule for finding applicable plans (step `ApplPl`). The presented semantics rules evaluate plan contexts via the updated PAL-AgentSpeak belief function.

**Evaluating Test Goals** Test goals $?\ell$ are used within the body of a plan to test for a belief $\ell$. We update the evaluation of test goals such that the updated belief function is used, and such that $\ell$ may be a certain or possible belief evaluated by the
Figure 3.6: The semantic rules for computing applicable plans in PAL-AgentSpeak, achieved via the AppPlans function.

epistemic reasoner. The function Test(M, B, R, ℓ) determines if ℓ is entailed by belief base B and epistemic model M, given a set of ranged literals R.

\[
\text{Test}(B, M, R, ℓ) = \{ \theta \mid \text{Bel}_{B,M,R}(ℓ\theta) \}
\]

The semantic rules in Figure 3.7 provide the operational semantics for the evaluation of test goals.

\[
\begin{align*}
T_i &= i[\text{head} ← ℓ; h] \quad \text{Test}(agB, agM, agR, ℓ) \neq \{\} \\
\langle ag, C, T, \text{ExecInt}\rangle_{PAL} &→ \langle ag, C', T, \text{Clrlnt}\rangle_{PAL} \\
\text{where} \quad T'_i &= i[(\text{head} ← h)\theta] \\
C'_i &= (C_i \setminus \{T_i\}) \cup T'_i
\end{align*}
\] (TestGl\textsubscript{1})

\[
\begin{align*}
T_i &= i[\text{head} ← ℓ; h] \quad \text{Test}(agB, agM, agR, ℓ) = \{\} \\
\langle ag, C, T, \text{ExecInt}\rangle_{PAL} &→ \langle ag, C', T, \text{Clrlnt}\rangle_{PAL} \\
\text{where} \quad C'_E &= C_E \cup \{⟨+?ℓ, T_i⟩\} \\
C'_I &= (C_I \setminus \{T_i\}) \\
C'_I &= (C_I \setminus \{T_i\})
\end{align*}
\] (TestGl\textsubscript{2})

Figure 3.7: The semantic rules for test goals in PAL-AgentSpeak.
Given an AgentSpeak belief formula model queries scale with respect to the number of worlds in the epistemic model and the number of modal formula that need to be evaluated. The complexity of this operation will be analyzed in Chapter 4 and evaluated in Chapter 6.

This concludes the operational semantics for PAL-AgentSpeak. In the next subsection, we summarize the contributions and limitations of PAL-AgentSpeak and provide concluding remarks.

### 3.2.6 Concluding Remarks

This section presented a novel extension “PAL-AgentSpeak”. Through the use of an epistemic reasoner, PAL-AgentSpeak agents are able to model and reason about uncertainty for their ranged literals. During model creation, the initial epistemic model is generated based on the ranges and constraints described by the agent.

Throughout the agent’s reasoning cycle, PAL-AgentSpeak updates the epistemic model by modelling belief additions as public announcement events. PAL-AgentSpeak is limited to the constructs provided by PAL and is inherently restricted to only capturing monotonic belief changes as invoked through the belief addition operator. Non-monotonic changes require a more expressive logic and AgentSpeak extension.

The model querying phase of PAL-AgentSpeak allows agents to reason about both certain and possible ranged literal beliefs through the added use of the possibility operator $\text{poss}$.

The operational semantics of PAL-AgentSpeak provided the formal integration of the epistemic reasoner with the model creation, updating, and querying phases of the AgentSpeak reasoning cycle.
For completeness, Listing 3.5 provides the PAL-AgentSpeak program for the navigation agent. Recall that since PAL-AgentSpeak does not model non-monotonic change, the listing is partial and does not sufficiently capture the change in uncertainty encountered by the domain. In the next section, we show how a more expressive extension, DEL-AgentSpeak, appropriately captures the change faced by the navigation agent. Chapter 5 provides a more appropriate use of PAL-AgentSpeak where a domain, Minesweeper, is sufficiently expressed.

```
range(loc(X, Y)) :- .ran(X, 0, 4) & .ran(Y, 0, 4).
range(loc none).

∼loc(X, Y) :- loc(X2, Y2) & [X, Y] \== (X2, Y2).
loc none :- ∼loc(0,0) & ... & ∼loc(4,4).

∼loc(1, 2).
∼loc(2, 2).
∼loc none.

// Range and constraints: obs(D)
range(obs(D)) :- .member(D, [up, down, left, right]).
obs(down) :- loc(1,1) | loc(2,1).
... // etc. for all obs(...) :- loc(..)
∼obs(D) :- not(obs(D)).

// Rule for nav direction (dir is not ranged)
dir(right, goal) :- loc(1,1) | loc(2,1).
!

// Plans for navigation
!*nav : dir(D, goal) | poss(dir(D, goal))
  <- move(D); !nav.
!*nav : not dir(_, goal)
  <- move(rand); !nav.
```

Listing 3.5: The PAL-AgentSpeak navigation program, excluding movement.

We briefly reiterate some of the limitations that come with PAL-AgentSpeak. First and foremost, it is critical that the agent maintains consistency among their ranged literal beliefs. This applies to the ranges and constraints described during model
creation and the changes that occur during the model update stage. This is a strict but reasonable requirement; the epistemic model is not valid and will not provide correct entailment of formulae when contradictions are present. Secondly, PAL-AgentSpeak is restricted to monotonic belief change captured by the belief addition operator; in the case where domains require non-monotonic change, we suggest the use of DEL-AgentSpeak.

In the next section we present DEL-AgentSpeak, a DEL-based extension that allows for the description of event uncertainty, and both monotonic and non-monotonic change, via DEL event models in the AgentSpeak program.

3.3 DEL-AgentSpeak: An Extension for Event Uncertainty and Non-Monotonic Change

In the previous section, PAL-AgentSpeak extended the standard AgentSpeak model creation, update, and querying phases through various semantic inferences and transformations that allow for the modelling and reasoning of uncertainty via PAL. PAL-AgentSpeak is a powerful and suitable extension for reasoning about uncertainty, but is limited to expressing monotonic belief change. When a domain requires the modelling of non-monotonic changes, or when there is uncertainty about the change event itself, we require DEL, a more expressive logic that handles these complex changes through the description of DEL event models.

This section presents a second AgentSpeak extension, DEL-AgentSpeak, which provides AgentSpeak agents with a standard convention, “on” plans, for expressing complex changes to uncertainty from within the AgentSpeak program; the extension uses the proposed syntactic convention to infer the creation and application of DEL
event models. There is considerable overlap between the methodologies used by PAL-AgentSpeak and DEL-AgentSpeak. As such, this section will present the more complex technical aspects that are involved in the integration with DEL but that were not necessary with PAL-AgentSpeak, including the need to provide semantics that maintain backwards compatibility with standard AgentSpeak.

3.3.1 DEL Event Models in DEL-AgentSpeak

As presented in Section 2.4, a DEL event model \( \langle E, \text{pre}, \text{post} \rangle \) captures the impact that an event or action has on the epistemic model and is considerably more expressive than a public announcement. Similar to the epistemic model’s set of possible worlds \( W \), the set of possible events \( E \) captures uncertainty about the event’s occurrence. Each possible event has their own pre- and post-conditions. In contrast, a public announcement corresponds to a single-event event model with a pre-condition. Public announcements thus fail to capture uncertainty about the event, and fail to capture ontic changes due to the lack of a post-condition. In order to take advantage of the full expressiveness of DEL event models in DEL-AgentSpeak, we propose “on” plans.

**Definition 3.3.1 (“On” Plans).** An “on” plan uses standard AgentSpeak plan syntax to describe the meta-impact of a belief event on the agent’s belief uncertainty and thus requires a different syntactic form and semantic assignment compared to standard AgentSpeak plans. For a given belief event \(+e\) (resp. \(-e\)), “on” plans are plans with the form \( te_{on} : ctx \leftarrow b_{+}^{-} \), where: \( te_{on} \) is a matching “on” event trigger \(+on(e)\) (resp. \(-on(e)\)), \( ctx \) is a standard plan context (without possibilities), and \( b_{+}^{-} \) is a plan body sequence strictly containing belief addition or deletion operations. The syntactic components of an “on” plan allow the agent to idiomatically describe a DEL event model via AgentSpeak syntax — i.e., when a belief event \(+e\) (resp. \(-e\)) occurs, the
corresponding “on” plans for \(+on(e)\) (resp. \(-on(e)\)) can be used to describe a DEL event model.

**Example 3.3.1.** In Figure 3.6, we provide an “on” plan that describes the effects of moving right (event \(+moved(right)\)) in the navigation example. Informally, we build a DEL event model \(\epsilon \rightarrow = \langle E, pre, post \rangle \) for \(+moved(right)\). The context can be used to describe \(pre\) and the plan body can be used to describe \(post\) (where \(+\ell\) is a \(\top\) post-condition and \(-\ell\) is a \(\bot\) post-condition for \(\ell\)). These plans are grounded by obtaining the logical consequences of the plan context with respect to all literals, and each ground instance of an “on” plan corresponds to a possible event instance in \(E\).

```
// "on" meta-plan to model event uncertainty
+on(moved(right)) : loc(X, Y) & X < 4
    <- -loc(X, Y);
    +loc(X + 1, Y).
```

Listing 3.6: The “on” plans for the navigation agent.

Using the ranged literals from our navigation example, we obtain the DEL event model \(\epsilon \rightarrow\) with \(\epsilon_{x,y} \in E\) where \((x,y)\) are the unifications obtained for each grounding of the “on” plan:

- \(pre(\epsilon_{0,0}) = loc(0,0), post(\epsilon_{0,0}, loc(0,0)) = \bot, post(\epsilon_{0,0}, loc(1,0)) = \top\)
- \(pre(\epsilon_{1,0}) = loc(1,0), post(\epsilon_{1,0}, loc(1,0)) = \bot, post(\epsilon_{1,0}, loc(2,0)) = \top\)
- \(\ldots\)
- \(pre(\epsilon_{3,4}) = loc(3,4), post(\epsilon_{3,4}, loc(3,4)) = \bot, post(\epsilon_{3,4}, loc(4,4)) = \top\)

By applying the created event model \(\epsilon \rightarrow\) to the agent’s current epistemic model, we correctly obtain a resultant model where the agent considers all locations right-adjacent to its previous possible locations.
“On” plans provide a standardized, concise, and powerful mechanism for expressing changes to uncertainty, enabled by an elegant combination of DEL’s expressive event models and various transformational semantics assigned by DEL-AgentSpeak described later in this section. We will now discuss a few critical challenges that arise due to the integration with DEL event models.

**Consistency with Ontic Changes** Ontic effects described by “on” plans may introduce new propositions to the epistemic model which require corresponding ranged literals. If we take an approach similar to PAL-AgentSpeak, which splits belief semantics of ranged and non-ranged literals between the epistemic model and the original belief base, the ontic effects of an “on” plan may inherently cause the epistemic model and belief base to become inconsistent. A naive fix is to limit ontic effects to ranged literals; however, this unnecessarily restricts the ability of “on” plans to express DEL event models. Instead, DEL-AgentSpeak chooses to model all literals using the epistemic model, removing the need to provide belief semantics via the belief base (note: the belief base component is still used for the purpose of grounding literals). As a result, we must show that DEL-AgentSpeak is backwards compatible with standard AgentSpeak programs.

**Backwards Compatibility With AgentSpeak** Since DEL-AgentSpeak no longer relies on the belief base for belief semantics, the model creation, update, and querying semantics provided by DEL must ensure backwards compatibility with standard AgentSpeak belief semantics, with respect to functionality and performance. Previously, Listing 2.1 provided a standard AgentSpeak listing without uncertainty. For the reader’s convenience, Listing 3.7 is an exact copy of the previous standard AgentSpeak listing — this program will be used throughout the section to consider the backwards
compatibility of DEL-AgentSpeak. We will now discuss some of the changes from PAL-AgentSpeak that need to occur in order to support DEL-AgentSpeak.

```prolog
loc(1,1).

// Rules for dir/obs that relate to location
obs(down) :- loc(1,1) | loc(2,1).
dir(right, goal) :- loc(1,1) | loc(2,1).
... // etc. for all obs/dir :- loc(..)

! nav.

// Plans for navigation
+! nav : dir(D, goal)
  <- move(D); ...
+! nav : not dir(_, goal)
  <- move(rand); ...

// Belief plans for changing locations (?)
+ moved(right) : loc(X, Y)
  <- -loc(X, Y);
  +loc(X + 1, Y).

// ... Etc.
```

Listing 3.7: From Listing 2.1, the standard AgentSpeak program for certain navigation.

### 3.3.2 Transition from PAL-AgentSpeak

The main differences between PAL-AgentSpeak and DEL-AgentSpeak lie in the model update process. The model creation and querying processes of the two extensions will be similar in nature, but require minor tweaks. For concision, DEL-AgentSpeak’s model creation and querying processes will reuse functions and definitions previously presented by PAL-AgentSpeak. We start with an updated propositionalization definition used by DEL-AgentSpeak.

**Definition 3.3.2** (DEL-AgentSpeak Propositionalization). In the section on PAL-AgentSpeak, Definition 3.2.1 introduced two propositionalization functions $pr(\varphi)$ for
the general propositionalization of all literals and \( pr_{(R)}(\varphi) \) which restricted propositionalization to those in the ranged literal set \( R \). Since DEL-AgentSpeak no longer distinguishes between ranged and non-ranged literals, the semantics of DEL-AgentSpeak will rely on the general function \( pr \) which propositionalizes any ground literal. In order to accommodate the change in propositionalization functions, we must redefine various dependant functions previously defined by PAL-AgentSpeak.

**Trivial Definition Updates**  Despite the removal of the belief base from belief semantics, DEL-AgentSpeak still requires the belief base \( B \) and a set of ranged literals \( R \) for grounding and unification purposes – hence, the updated definitions given below may still require \( B \) and \( R \). Since the redefinitions of the following functions are trivial, we do not provide their full definitions. The following functions are trivially redefined to replace usage of \( pr_{(R)} \) with \( pr \):

- The DEL-AgentSpeak functions \( pr\_range \) and \( pr\_con \) are based on \( pr_{(R)}\_range \) and \( pr_{(R)}\_con \) from PAL-AgentSpeak’s model creation (Section 3.2.2),

- DEL-AgentSpeak’s belief function \( Bel^{*}_{(B,M,R)}(\varphi_Q) \), test function \( Test^{*}(M,B,R,\ell) \), and applicable plans function \( AppPlans^{*}(R_P,B,M,R) \) are the redefined forms of \( Bel_{(B,M,R)}(\varphi_Q) \), \( Test(M,B,R,\ell) \), and \( AppPlans(R_P,B,M,R) \) from Section 3.2.4. Additionally, any inter-dependencies between the functions will be updated, e.g., if \( Test \) relies on \( Bel \) in PAL-AgentSpeak, then \( Test^{*} \) will now rely on \( Bel^{*} \) in DEL-AgentSpeak.

Before introducing the model creation, update, and querying processes for DEL-AgentSpeak, we first define the configuration of a DEL-AgentSpeak agent.

**Definition 3.3.3** (DEL-AgentSpeak Configuration). A DEL-AgentSpeak configuration \( \langle ag,C,T,S \rangle_{DEL} \) is the same as PAL-AgentSpeak (Definition 3.2.4), but is
denoted with the subscript $\text{DEL}$. To reiterate, the following sub-components in $ag$ are provided:

- Belief base $ag_B$ holds the agent’s beliefs and rules and is initialized with initial beliefs/rules,
- $ag_R$ holds the set of all ranged literals,
- $ag_M$ stores the epistemic model.

Since the semantic rules presented throughout this section will append to the semantic rules of standard AgentSpeak, any additions made to the semantic rules will be boxed.

### 3.3.3 Model Creation

DEL-AgentSpeak uses the agent’s belief ranges and constraints to describe the initial epistemic model. The semantics assigned to the ranges and constraints are exactly the same as PAL-AgentSpeak. Since all non-ranged beliefs will now be modelled using the epistemic model, we must ensure these beliefs are included in the constraints used for initial model generation. Similar to PAL-AgentSpeak, we do not propositionalize standard rules that describe non-ranged literals (i.e., non-constraints) — there is no benefit to propositionalizing non-ranged rules during model creation as they are evaluated appropriately when querying.

**Definition 3.3.4** (Obtaining Initial Model Constraints). Given belief base $B$ with the set of explicit beliefs $\text{bels}(B)$ and a set of ranged literals $R$, we include the propositionalization of all non-ranged initial belief literals in the set of propositional
constraints used to generate the initial epistemic model:

\[
all_{cons}^*(B, R) = \{ pr(\ell) \mid \ell \in bels(B) \} \cup \\
\{ pr_{range}(\ell) \mid \ell \in R \} \cup \\
\{ pr_{con}(\ell, \varphi) \mid (\ell, \varphi) \in cons(R, B) \}
\]

This constraint function is modified from PAL-AgentSpeak to include non-ranged initial beliefs. The next section discusses the operational semantics of DEL-AgentSpeak’s model creation process and provides remarks about its backwards compatibility.

### 3.3.3.1 Operational Semantics

The model creation phase initializes the DEL-AgentSpeak configuration \( \langle ag, C, T, S \rangle_{DEL} \), such that the set of ranged literals is initialized: \( ag_R = ranges(ag_B) \), and the epistemic model is initialized accordingly: \( ag_M = gen\_model(S) \) given \( S = all_{cons}^*(ag_B, ag_R) \). Similar to PAL-AgentSpeak, the worst-case time and space complexity will be exponential in nature due to the need for obtaining logical consequences of constraints and SAT solving.

#### Backwards Compatibility

The standard AgentSpeak program presented in Listing 3.7 has no ranges and thus no defined constraints. As such, when executing this listing with DEL-AgentSpeak, only the initial beliefs are propositionalized and used for generating the epistemic model. For example, the program in Listing 3.7 gives the set of propositional sentences \( S = \{ \text{lit}(1,1) \} \). Since no uncertainty exists, a single-world epistemic model is created where \( gen\_model(S) \) gives us:

- \( W = \{ \{ \text{lit}(1,1) \} \} \)
In the case of complete certainty, the epistemic model always holds a single world with a propositional state that matches an initial belief base in standard AgentSpeak. As a result, the epistemic model entails the same belief formulae as a belief base; i.e., belief (□_B) of all initial belief literals is entailed. From a performance perspective, generation of a single state given only true (or false) propositional symbols is trivial and does not deviate from the performance of initializing a standard belief base (i.e., adding initial belief literals to the set) — both are linear operations with respect to the number of initial beliefs. Thus, backwards compatibility is maintained for DEL-AgentSpeak’s model creation process. In the next subsection, we present the model update process for DEL-AgentSpeak.

3.3.4 Model Updating

Due to the differences in update semantics between DEL and PAL, the model update process for DEL-AgentSpeak will differ from PAL-AgentSpeak. In the introduction to DEL-AgentSpeak, we introduced “on” plans. Although “on” plans provide a concise and powerful mechanism for describing changes to uncertainty, we assign a default DEL event model to standard belief additions or deletions so that “on” plans are only required when the default event model is insufficient.

3.3.4.1 Default DEL Event Models

Given a belief addition +\ell or belief deletion −\ell, we assign default semantics to these operators to ensure that the epistemic model gets updated accordingly, without the need to specify “on” plans. There are two semantic options for modelling this update: 1) apply \ell as an epistemic update, or 2) apply \ell as an ontic update. Additionally, it is
critical that the default semantics we assign to belief additions and deletions maintain backwards compatibility with AgentSpeak.

As we saw with PAL-AgentSpeak, the epistemic (public announcement) semantics assigned to the belief addition operator allowed for monotonic changes to its knowledge but also required various workarounds to model non-monotonic changes (e.g., belief deletion). In contrast, using ontic change to model belief additions and deletions do not require workarounds, and as we will see, also trivially provides backwards compatibility.

**Definition 3.3.5** (Default DEL Event Models). Given a ground belief event $e$ we use the following function to create a default DEL event model $cr_{def}(e) = (E, pre, post)$ with:

- $E = \{e\}$
- $pre(e) = \top$
- $post(e, l) = \begin{cases} 
\top & \text{if } e = +l \\
\bot & \text{if } e = -l
\end{cases}$

After the event is applied to the epistemic model, all worlds in the model will contain $l$ if $+l$ or $-l$ if $-l$, leading to entailment of $\square_B l$ or $\square_B \neg l$, respectively.

**Example 3.3.2.** For example, a belief addition $+\ell$ and deletion $-\ell$ would result in the following:

- $+\ell \Rightarrow (E = \{+\ell\}, pre(+\ell) = \top, post(+\ell, \ell) = \top)$
- $-\ell \Rightarrow (E = \{-\ell\}, pre(-\ell) = \top, post(-\ell, \ell) = \bot)$
3.3.4.2 Custom Event Models: “On” Plans

DEL-AgentSpeak allows intricate epistemic event descriptions using “on” plans. These plans override the default event model assigned to a belief event. Definition 3.3.1 introduced “on” plans as AgentSpeak plans that have the form: \( te_{on} : ctx \leftarrow b_{+-} \), and Example 3.3.1 demonstrated the use of “on” plans with the navigation example. Before we are able to introduce the operational semantics of DEL-AgentSpeak’s model updates, we introduce preliminary definitions for finding “on” plans and grounding them before they are transformed into DEL event models. In the coming definitions, we use \( Tr(p) \), \( Ctxt(p) \), and \( Body(p) \) to represent the trigger, context, and body of a plan \( p \).

**Finding Relevant “On” Plans** Given a set of agent plan definitions \( P \) and a belief event \( te = +e \) or \( -e \), the function \( rel_{on}(P, te) \) finds relevant “on” plans whose triggers handle \( te \).

\[
rel_{on}(P, te) = \{(p, \theta) \mid p \in P \text{ and } \begin{cases}
{+on}(e) \models Tr(p)\theta & \text{if } te = +e, \\
{-on}(e) \models Tr(p)\theta & \text{else, } te = -e
\end{cases}
\}
\]

The resulting set contains pairs \((p, \theta)\) where \( p \) is an “on” plan whose trigger handles the corresponding belief event, given MGU \( \theta \).

**Grounding Relevant “On” Plans** Given a set \( R_{On} \) of pairs containing relevant “on” plans and their trigger unifiers, a belief base \( B \), a set of ranged literals \( R \), and their combined grounding set \( B_R = B \cup R \), we ground all “on” plans by obtaining the
logical consequences of the relevant plan contexts with $B_R$.

$$gnd_{on}(B_R, R_{On}) = \{(p', \theta' \circ \theta) \mid (p, \theta) \in R_{On} \text{ with } p' = (Tr(p) : \varphi \leftarrow Body(p)) \text{ s. th. } \varphi \in rw_{(B_R)}(Ctxt(p)\theta')\}$$

The resulting set contains pairs $(p, \theta'')$ where $p$ is a modified “on” plan whose context is rewritten into $\varphi$ for future propositionalization, and $\theta'' = \theta' \circ \theta$ is a composite unifier which combines unifications in the trigger unifier $\theta$ and context unifier $\theta'$. For convenience, we provide a utility function for the semantic rules that invoke both the $rel_{on}$ and $gnd_{on}$ functions, given a set of all plans $P$, $B_R$, and a belief event $te$:

$$find_{on}(P, B_R, te) = gnd_{on}(B \cup R, rel_{on}(P, te))$$

**Transforming “on” Plans into DEL Event Models**  
DEL-AgentSpeak transforms a ground “on” plan $(a, \theta)$ with plan $a$ and MGU $\theta$, into its corresponding DEL event as follows: $(a, \theta)$ is the event’s designated identifier and $pre((a, \theta)) = pr(Ctxt(a)\theta)$ is the plan context serving as the event precondition. Within the plan body $Body(a)\theta$, belief additions and deletions are interpreted as ontic effects, providing us with event post-conditions:

$$post((a, \theta)) = \begin{cases} (pr(\ell), \top) & \text{if } +\ell \in Body(a)\theta \\ (pr(\ell), \bot) & \text{if } -\ell \in Body(a)\theta \end{cases}$$

When processing “on” plans we must also capture and store any literals used to describe ontic effects as the agent may want to use these literals to ground future updates or queries. This only applies to ontic additions $+\ell$ defined by the “on” plan.
Conversely, ontic deletions $-\ell$ do not warrant the removal of $\ell$ as the ontic effects may not apply to all worlds (and thus $\ell$ may still need to be grounded). To obtain all new ground literals defined via ontic additions we present the following function given a set of ground “on” plan pairs $A$:

$$ontic\_lits(A) = \{\ell, -\ell \mid (a, \theta) \in A \text{ where } +\ell \in Body(a,\theta)\}$$

### 3.3.4.3 Operational Semantics

In DEL-AgentSpeak, we apply the DEL event models created by “on” plans when available, and the default event model in all other cases. Given a belief event $te$ and a set of ground “on” plan pairs $A$, we create a corresponding DEL event model for $te$ (or use the default event model when $A = \emptyset$). We define a function that provides the appropriate DEL event model:

$$del\_em(A, te) = \begin{cases} \langle E = \{(a, \theta) \in A\}, pre((a, \theta)), post((a, \theta))\rangle & \text{if } A \neq \emptyset \\ cr\_def(te) & \text{else, } A = \emptyset \end{cases}$$

Similar to PAL-AgentSpeak, the DEL event models are created and applied to the epistemic model as soon as the belief addition or deletion occurs. The semantic rules in Figures 3.8 and 3.9 find any applicable “on” plans, execute the appropriate DEL event model, and update ranged literals according to ontic changes, when a belief addition or deletion operation occurs.

The complexity of the model update operation scales linearly with respect to the number of current worlds in the epistemic model, the number of events in the resulting
\[ T_i = i[\text{head} \leftarrow +b; h] \]
\[ \langle ag, C, T, \text{ExecInt} \rangle_{\text{DEL}} \rightarrow \langle ag', C', T', \text{ClrInt} \rangle_{\text{DEL}} \quad (\text{AddBel}') \]

\[ \begin{aligned}
    ag'_B &= ag_B + b \\
    C'_E &= C_E \cup \{(+b, \top)\} \\
    T'_i &= i[\text{head} \leftarrow h]
\end{aligned} \]

where
\[ \begin{aligned}
    C'_i &= (C_i \setminus \{T_i\} \cup \{T'_i\}) \\
    A &= \text{find}_\text{on}(ag_{ps}, ag_B \cup ag_R, +b) \\
    ag'_R &= ag_R \cup \text{ontic}_\text{lits}(A) \\
    ag'_M &= ag_M \otimes \text{del}_\text{em}(A, +b)
\end{aligned} \]

Figure 3.8: The semantic rule for belief addition in DEL-AgentSpeak.

\[ T_i = i[\text{head} \leftarrow -b; h] \]
\[ \langle ag, C, T, \text{ExecInt} \rangle_{\text{DEL}} \rightarrow \langle ag', C', T', \text{ClrInt} \rangle_{\text{DEL}} \quad (\text{DelBel}') \]

\[ \begin{aligned}
    ag'_B &= ag_B - b \\
    C'_E &= C_E \cup \{(-b, \top)\} \\
    T'_i &= i[\text{head} \leftarrow h]
\end{aligned} \]

where
\[ \begin{aligned}
    C'_i &= (C_i \setminus \{T_i\} \cup \{T'_i\}) \\
    A &= \text{find}_\text{on}(ag_{ps}, ag_B \cup ag_R, -b) \\
    ag'_R &= ag_R \cup \text{ontic}_\text{lits}(A) \\
    ag'_M &= ag_M \otimes \text{del}_\text{em}(A, -b)
\end{aligned} \]

Figure 3.9: The semantic rule for belief deletion in DEL-AgentSpeak.

Backwards Compatibility In order to consider the backwards compatibility of DEL-AgentSpeak’s model updates, we refer to Listing 3.7. A standard listing will not contain “on” plans, thus all belief additions and deletions are assigned the default DEL event models. When the agent moves right, the standard plan \textit{+moved(right)} is executed with context unifier \( \theta = \{X = 1, Y = 1\} \) (due to belief \textit{loc}(1,1)). The event model, and the size of each event’s pre- and post-conditions. This is explored further in Section 4.2.3.
following sequence of DEL events are executed due to the belief deletion and addition that occur in the plan:

1. \( \langle E = \{ -loc(1, 1) \}, \text{pre}( -loc(1, 1) ) = \top, \text{post}( -loc(1, 1), loc(1, 1) ) = \bot \rangle \)

2. \( \langle E = \{ +loc(2, 1) \}, \text{pre}( +loc(2, 1) ) = \top, \text{post}( +loc(2, 1), loc(2, 1) ) = \top \rangle \)

During model creation, the initial epistemic model is created with a single world that reflects the certain beliefs of the agent. When these two event models are executed in sequence, the proposition \( loc(1, 1) \) is removed and \( loc(2, 1) \) is added to the world’s valuation. The effective result is the following model:

- \( W = \{ \{ \text{lit}(2, 1) \} \} \)

After each addition and deletion operation, the resulting epistemic model appropriately reflects the beliefs held by a belief base under standard belief semantics. Additionally, applying the default DEL event to the epistemic model can be done in linear time with respect to the number of worlds. Since the model only holds a single world, the default update semantics have identical performance to a belief addition/deletion using standard belief base semantics. Thus, model updates provided by DEL-AgentSpeak are backwards compatible with standard AgentSpeak semantics. In the next section, we present the model querying phase of DEL-AgentSpeak.

### 3.3.5 Model Querying

In DEL-AgentSpeak, the model querying process will be similar to the one presented for PAL-AgentSpeak, but will integrate the updated belief function \( Bel^*_\{B,M,R\}(\varphi_Q) \), updated test function \( Test^*(M,B,R,\ell) \), and updated applicable plans function \( AppPlans^*(R_P,B,M,R) \), which were previously introduced in Section 3.3.2. The operational semantics for model querying in DEL-AgentSpeak will now be presented.
3.3.5.1 Operational Semantics

The following semantic rules are based on the existing PAL-AgentSpeak rules presented in Section 3.2.5 with minimal changes to integrate the updated querying functions. The semantic rules \( Appl_1^* \) and \( Appl_2^* \) in Figure 3.10 determine the set of applicable plans based on DEL-AgentSpeak’s updated \( AppPlans^* \) function. Figure 3.11 provides the updated semantic rules \( TestGl_1^* \) and \( TestGl_2^* \) for executing test goals, which now rely on the updated \( Test^* \) function.

\[
\frac{\text{AppPlans}^*(T_R, ag_B, ag_M, ag_R) \neq \{\}}{\langle ag, C, T, Appl\rangle_{DEL} \rightarrow \langle ag, C, T', SelAppl\rangle_{DEL}} \quad (Appl_1^*)
\]

where \( T'_\text{Ap} = \text{AppPlans}^*(T_R, ag_B, ag_M, ag_R) \)

\[
\frac{\text{AppPlans}^*(T_R, ag_B, ag_M, ag_R) = \{\}}{\langle ag, C, T, Appl\rangle_{DEL} \rightarrow \langle ag, C, T, SelInt\rangle_{DEL}} \quad (Appl_2^*)
\]

Figure 3.10: The semantic rules for computing applicable plans in DEL-AgentSpeak.

\[
T_i = i[\text{head} \leftarrow ?\ell; h] \quad \frac{\text{Test}^*(ag_B, ag_M, ag_R, \ell) \neq \{\}}{\langle ag, C, T, ExecInt\rangle_{DEL} \rightarrow \langle ag, C', T, ClrInt\rangle_{DEL}} \quad (TestGl_1^*)
\]

where \( T'_i = i[(\text{head} \leftarrow h)\theta] \)

\[
\theta \in \text{Test}^*(ag_B, ag_M, ag_R, \ell)
\]

\[
C'_i = (C_i \setminus \{T_i\}) \cup T'_i
\]

\[
T_i = i[\text{head} \leftarrow ?\ell; h] \quad \frac{\text{Test}^*(ag_B, ag_M, ag_R, \ell) = \{\}}{\langle ag, C, T, ExecInt\rangle_{DEL} \rightarrow \langle ag, C', T, ClrInt\rangle_{DEL}} \quad (TestGl_2^*)
\]

where \( C'_E = C_E \cup \{+?\ell, T_i\} \)

\[
C'_I = (C_I \setminus \{T_i\})
\]

Figure 3.11: The semantic rules for test goals in DEL-AgentSpeak.
The complexity of the model querying operation scales linearly with respect to the number of current worlds in the epistemic model and the size of the modal formula being evaluated. This is explored further in Section 4.2.4.

**Backwards Compatibility** As shown in the above model creation and update subsections of DEL-AgentSpeak, both processes maintain backwards compatibility of the belief semantics provided by the epistemic model after it is created and updated. This transitively shows the backwards compatibility of model querying, which uses the created/updated epistemic model to provide belief semantics to belief queries. From the lens of performance, the evaluation of the belief modality $\square_B$ can be done in linear time with respect to the size of the model. Since a standard AgentSpeak program will only maintain a single world in the epistemic model, the performance of querying beliefs in DEL-AgentSpeak is identical to the performance demonstrated by a standard belief base.

With respect to functionality and performance, we shown that DEL-AgentSpeak maintains backwards compatibility with a standard AgentSpeak program across model creation, model updates, and model queries. We will now conclude the chapter with some final remarks and provide recommendations for choosing between DEL-AgentSpeak and PAL-AgentSpeak, as they both associate with different applications.

### 3.3.6 Concluding Remarks

This section presented DEL-AgentSpeak, an AgentSpeak extension for reasoning about uncertainty. DEL-AgentSpeak relies on a DEL reasoner, which allows for the use of DEL event models as a rich way to express changes to uncertainty. The model creation and querying stages of DEL-AgentSpeak followed a similar process
to PAL-AgentSpeak, where ranges and constraints are used to generate an initial epistemic model, and the semantics for evaluating belief formulae were delegated to the reasoner. DEL-AgentSpeak also introduced “on” plans, which use plan syntax to describe the change in uncertainty caused by a belief event. When a belief addition or deletion occurs, DEL-AgentSpeak transforms the applicable “on” plans into a DEL event model. In the case where “on” plans are not given for the corresponding belief event, a default ontic event model is used.

In PAL-AgentSpeak, there was a clear divide between belief semantics for ranged literals and non-ranged literals (i.e., epistemic reasoner vs. standard belief base). Since ontic effects may possibly change the valuation of any ranged or non-ranged literal, there is potential for inconsistency between the belief semantics provided by the epistemic reasoner and belief base. To alleviate this, DEL-AgentSpeak solely relied on the epistemic reasoner for belief semantics, and reserved the belief base for the grounding of literals. Due to this, DEL-AgentSpeak also demonstrated that its use of the epistemic reasoner maintains backwards compatibility with belief semantics provided by the belief base, across the model creation, update, and querying phases.

The final DEL-AgentSpeak listing is provided in Figure 3.8. In comparison to PAL-AgentSpeak’s Listing 3.5, DEL-AgentSpeak allows us to provide a complete program for the navigation agent due to its ability to appropriately express the change encountered in the domain. The DEL-AgentSpeak listing also requires “on” plans for $+\text{obs}(D)$ and $+\sim\text{obs}(D)$ to appropriately model them as epistemic change. Note that unlike PAL-AgentSpeak, $\text{obs}(\_)$ does not need to be ranged due to the difference in semantics of PAL-AgentSpeak’s belief addition public announcements and DEL-AgentSpeak’s “on” plans.
range(loc(X, Y)) :- ran(X, 0, 4) & ran(Y, 0, 4).

~loc(X, Y) :- loc(X2, Y2) & [X, Y] \== (X2, Y2).

range(locnone)).

loc(none) :- ~loc(0,0) & ... & ~loc(4,4).

~loc(1, 2).

~loc(2, 2).

~loc(none).

// Standard rules for non-ranged obs and dir
obs(down) :- loc(1,1) | loc(2,1).

... // etc. for all obs(...) :- loc(...)  
~obs(D) :- not(obs(D)).

dir(right, goal) :- loc(1,1) | loc(2,1).

... // etc. for all dir(...) :- loc(...)  
!nav.

// Epistemic change for perception of obstacles
+on(obs(D)) : obs(D).

+on(~obs(D)) : ~obs(D).

// "on" plans for ontic effects of movement
+on(moved(right)) : loc(X, Y) & X < 4

<- ~loc(X, Y);

  +loc(X + 1, Y).

... // Etc. for +moved(left)+moved(up)+moved(down)

// Plans for navigation
+!nav : dir(D, goal) | poss(dir(D, goal))

<- move(D); !nav.

Listing 3.8: The DEL-AgentSpeak navigation program.

PAL-AgentSpeak and DEL-AgentSpeak are novel extensions to AgentSpeak that present the creation, updating, and querying of an epistemic model via a PAL/DEL reasoner in a way that allows the AgentSpeak agent to model and reason about the changes to its uncertainty.

Since the logic of PAL is subsumed by DEL, DEL-AgentSpeak programs can implicitly express those sufficiently expressed by PAL-AgentSpeak, but require the specification of “on” plans. As such, PAL-AgentSpeak would be the recommended choice as it requires less syntactic input; this becomes increasingly advantageous when there are a large number of monotonic belief events that occur. In contrast,
DEL-AgentSpeak provides a much higher level of expressiveness, and is thus required in cases where PAL is not sufficiently expressive, such as non-monotonic changes and event uncertainty. It is thus up to the agent’s developer to choose the extension that is most appropriate for their domain. We further explore the specific applications for PAL-AgentSpeak and DEL-AgentSpeak in Chapter 5.

The next chapter presents the critical details regarding the implementation of PAL-AgentSpeak and DEL-AgentSpeak. This includes their integration with Hintikka’s World, the PAL/DEL reasoner, and their integration with the AgentSpeak reasoning cycle as implemented by Jason. Throughout the chapter, we also provide time and space complexity analyses for all operations performed by PAL-/DEL-AgentSpeak, so that we get a better understanding of how these extensions impact the performance of the agent.
Chapter 4

Implementation

In the previous chapter, we introduced the formal methodologies associated with PAL-AgentSpeak and DEL-AgentSpeak, and demonstrated via the running navigation example, how these extensions augment AgentSpeak with qualitative uncertainty reasoning. This chapter focuses on the implementation of these methodologies, their integration with the PAL/DEL reasoner Hintikka’s World, and the additional contributions that resulted from this integration.

To facilitate the integration between PAL-/DEL-AgentSpeak and Hintikka’s World, we require a medium for communication. This is achieved through the extension of Hintikka’s World with a Representational State Transfer (REST) API that exposes endpoints for the creation, updating, and querying of epistemic models. Furthermore, we offer a compact implementation for \textit{S5 models} in Hintikka’s World, which introduces various optimizations that directly influence the computation time and space required by PAL-/DEL-AgentSpeak to perform various operations; this evaluation will be performed in Chapter 6.
We begin by presenting our extensions and contributions to Hintikka’s World. Later, Section 4.2 presents the implementation of PAL- and DEL-AgentSpeak. Throughout the chapter, we provide the implementation of various operations and state their relevant time and space complexities. For brevity, implementations related to Hintikka’s World will be provided using pseudo-JavaScript, and those related to PAL-/DEL-AgentSpeak will be written in pseudo-Java, respective to their actual implementation languages. To save space, blocks of code are denoted via indentation.

4.1 Extending Hintikka’s World

Hintikka’s World (HW) offers an epistemic reasoner capable of PAL and DEL, as well as SAT solving capabilities through a MiniSAT-based engine known as TouIST [53]. In this section, we discuss the main implementation contributions made to HW, which extend the reasoner with:

- A compact and optimized representation for S5 epistemic models and events
- A REST API for external invocation of the HW PAL/DEL reasoner

These contributions allow us to efficiently integrate the PAL/DEL reasoning abilities of HW with PAL-/DEL-AgentSpeak, which will be discussed in Section 4.2. The extended implementation for Hintikka’s World is available at: https://github.com/MikeVezina/epistemic-reasoner.

We begin with an overview of our implementation for compact S5 epistemic and event model representations. Compact models were briefly introduced in Section 2.4 as an efficient method for describing S5 epistemic and event models, in which the indistinguishability relation, \( R \), is implicit and therefore excluded from the epistemic/event
model descriptions. As previously discussed in Section 2.4, we selected S5 models for this thesis due to their strong assertions about uncertainty [14] and, more importantly, because they enable us to present a compact model implementation that offers time and space optimizations, as we demonstrate in Section 4.1.3, which in turn impact the performance of PAL-/DEL-AgentSpeak.

To emphasize the advantages of these compact representations, we will investigate both standard and compact representations, their initialization and usage, and the complexities associated with them. We commence with the standard representation to provide the reader with an understanding of what HW already offers and how it can be enhanced for our specific use case.

### 4.1.1 Standard Model Implementation

HW is based on the standard epistemic and event model representation found in the literature [14]: \( M = \langle W, R_W, V \rangle \) and \( \varepsilon = \langle E, R_E, \text{pre}, \text{post} \rangle \), where \( R_W \subseteq W \times W \) and \( R_E \subseteq E \times E \) denote indistinguishability relations between worlds and events [14]. In HW, standard epistemic models and event models are represented through instances of `ExplicitEpistemicModel` and `ExplicitEventModel`. As this thesis employs the S5 system of epistemic logic (where \( R_W = W \times W \) and \( R_E = E \times E \)), we will now illustrate the initialization of S5 models using standard representations.

Listing 4.1 showcases the creation of an S5 epistemic model with a set of \( W \) worlds and a valuation \( V \) that maps worlds to sets of true propositions. On the other hand, Listing 4.2 demonstrates the creation of an S5 event model where \( E \) represents the set of events, \( \text{pre} \) maps events to their precondition formulas, and \( \text{post} \) maps events to a mapping of propositions and their corresponding postcondition formulas.
function createStandardModel(W: Set<World>,
V: Map<World,Set<Prop>>)
let M = new ExplicitEpistemicModel();
// Initialize Worlds and Valuations
for(let w of W)
  M.addWorld(w, V[w]);
// Initialize indistinguishability
for(let w1 of W)
  for(let w2 of W)
    M.addIndistinguishable(w1, w2);
return M;

Listing 4.1: Creating a standard S5 epistemic model.

function createStandardEvent(E: Set<Event>, pre: Map<Event,Formula>,
post: Map<Event,Map<Prop,Formula>>)
let EM = new ExplicitEventModel();
// Initialize Events
for(let e of E)
  EM.addEvent(e, pre[e], post[e]);
// Initialize indistinguishability
for(let e1 of E)
  for(let e2 of E)
    EM.addIndistinguishable(e1, e2);
return EM;

Listing 4.2: Creating a standard S5 event model.

Given an ExplicitEpistemicModel $M$ and ExplicitEventModel $E$, the functions $M$.getSuccessors($w$: World) and $E$.getSuccessors($e$: Event) respectively obtain the indistinguishable worlds and events for a world $w$ in $M$ or event $e$ in $E$. These successor functions are used by the corresponding implementations for model checking and applying event models, namely:

- $M$.modelCheck(form: KFormula), where KFormula represents the modal formula $\Box_B \varphi$ with getInner() = $\varphi$.

- $E$.apply(model: ExplicitEpistemicModel)
The implementation of these functions based on the standard model representations are provided by HW in Listings 4.3 and 4.4.

```javascript
function M.modelCheck(w: World, form: KFormula)
for(let w2 of this.getSuccessors(w))
  if(!w2.modelCheck(form.getInner()))
    return false;
return true;
```

Listing 4.3: Model checking a standard epistemic model.

```javascript
function E.apply(M: ExplicitEpistemicModel)
// Resultant model
let ME = new ExplicitEventModel();
// Initialize Events
for(let e of this.getEvents())
  for(let w of M.getWorlds())
    // Check pre and apply post
    if(M.modelCheck(w, e.getPre()))
      ME.addWorld((w, e), e.applyPost(w.getValuation()));
// Resultant indistinguishability relation
for(let w2 of M.getSuccessors(w))
  for(let e2 of E.getSuccessors(e))
    if(ME.hasWorld((w2,e2))
      ME.addIndistinguishable((w,e), (w2,e2));
return ME;
```

Listing 4.4: Apply a standard event model to a standard epistemic model.

In the next subsection, we conclude with a brief complexity analysis of the presented operations.

### 4.1.1.1 Complexity of Standard Models

When utilizing the standard model representations, we require $O(W^2)$ time and space to create the epistemic model (Listing 4.1), and $O(E^2)$ time and space to create the event model (Listing 4.2), given $W$ worlds and $E$ events. When applying an event model to an epistemic model, we see that the product-update operation (Listing 4.4) requires $O((E \times W)^2)$ time and space to initialize and store the resultant model.
Given that S5 models rely on an equivalence indistinguishability relation, there is an opportunity to optimize the initialization of these models such that the relation can be implicit and we don’t have to iterate or store $W^2$, $E^2$, or $(W \ast E)^2$. This is what our compact models propose to optimize.

### 4.1.2 Compact Model Implementation

Compact epistemic and event models are those that have been employed throughout the thesis, i.e., $M = \langle W, V \rangle$ and $\varepsilon = \langle E, pre, post \rangle$, which are permitted due to our strict usage of S5 models. As part of the contributions made by this thesis, we provide a compact representation for S5 models within HW. We provide the compact representations with the new classes `CompactEpistemicModel` and `CompactEventModel`. In comparison to the instantiation of the standard model representations presented in Listings 4.5 and 4.6, we see that the compact instantiations shown in Listings 4.5 and 4.6 no longer require the explicit creation of the indistinguishability relations.

```javascript
function createCompactModel(W: Set<World>, V: Map<World, Set<Prop>>)
let M = new CompactEpistemicModel();
// Initialize Worlds and Valuations
for(let w of W)
    M.addWorld(w, V[w]);
return M;
```

Listing 4.5: Creating a compact S5 epistemic model.

Given a `CompactEpistemicModel CM` and `CompactEventModel CE`, we provide updated implementations for model checking (Listings 4.7) and applying compact event models (Listings 4.8). Notice that the updated model checking function utilizes `this.get Worlds()` instead of `getSuccessors(w)` since the two are equivalent in an S5 system.
function createCompactEvent(E: Set<Event>, pre: Map<Event, Formula>,
post: Map<Event, Map<Prop, Formula>>)

let EM = new CompactEventModel();

// Initialize Events
for(let e of E)
EM.addEvent(e, pre[e], post[e]);
return EM;

Listing 4.6: Creating a compact S5 event model.

function CM.modelCheck(_: World, form: KFormula)
for(let w of this.getWorlds())
if(!w.modelCheck(form.getInner()))
return false;
return true;

Listing 4.7: Model checking a compact epistemic model.

function CE.apply(CM: CompactEpistemicModel)

// Resultant model
let CME = new CompactEventModel();
// Initialize Events
for(let e of this.getEvents())
for(let w of CM.getWorlds())
// Check pre and apply post
if(CM.modelCheck(w, e.getPre()))
CME.addWorld((w, e), e.applyPost(w.getValuation()));
return CME;

Listing 4.8: Apply a compact event model to a compact epistemic model.

We will now discuss the complexity of instantiating compact epistemic and event models. This will be followed with a practical performance comparison of standard and compact model representations.

4.1.2.1 Complexity of Compact Models

When utilizing the compact model representations, we require $O(W)$ time and space to create the epistemic model (Listing 4.5), and $O(E)$ time and space to create the event model (Listing 4.6), given $W$ worlds and $E$ events. When applying an event model
to an epistemic model, we see that the product-update operation (Listing 4.8) only requires $O(E \times W)$ time and space to initialize and store the resultant compact model. The following subsection will provide a comparison of the two model representations by measuring and comparing the time and space each instantiation takes.

### 4.1.3 Performance Benefits of Compact Models

We briefly evaluate the performance differences between the standard and compact models. This experiment simply records the computation time and storage space required for each model as the number of worlds $|W|$ increases. Figure 4.1 shows that compact models bring significant time and space optimizations due to the lack of a $R = W \times W$ relation. The raw data are provided in Table B.1 of Appendix B.

![Graphs showing performance comparison](image)

(a) Initialization time.  
(b) Model space.

Figure 4.1: Standard vs. compact model comparison.

The standard and compact model representations are classified as *explicit* (or semantic) representations, which require explicit storage of worlds. Despite the compact representation being much more efficient in terms of time and space, it shares a major disadvantage with explicit representations. Specifically, $|W|$ may increase to exponential values and thus requires an exponential state space that limits its practical applicability in more complex scenarios of uncertainty [54].
**Implicit Models**  The alternative representation to explicit models, *implicit models*, is based on a syntactic approach in which uncertainty is succinctly described using formulae. Implicit models are space efficient but bring an increase in time complexity, since SAT solving (NP-complete) is used instead of the tableau method (polynomial) for model updates and queries, which are frequently invoked by AgentSpeak agents [47]. This thesis proceeds with a compact explicit representation to maintain efficient update and query operations, as additional computation time is not suitable for AgentSpeak’s time-sensitive reasoning cycle. Future work involves the integration of PAL-/DEL-AgentSpeak with implicit models such that they can be employed when space constraints become impractical in more complex domains of uncertainty.

This concludes our presentation of the compact S5 model extensions made to HW. In the following subsection, we extend HW with a REST API that allows for external invocation of reasoning capabilities. The endpoints exposed by the API will be used by PAL-/DEL-AgentSpeak, but also signify a much more general contribution.

### 4.1.4 Providing an Epistemic Reasoning API

In this section, we present the extension of HW with a REST API. This API allows for the external creation, updating, and querying of compact epistemic/event models, and is not exclusive to PAL-/DEL-AgentSpeak. This API is generic and can be invoked by any program seeking to take advantage of PAL/DEL reasoning services.

We have expanded HW to include an additional class responsible for creating our desired REST API server. Figure 4.2 illustrates the fields and operations provided by the API server. The server contains a single epistemic model, represented by the `curModel` field, which can be created, updated, and queried by calling the `generateModel`, `applyModel`, and `queryModel` methods. These functions are exposed through endpoints
via the `createApp` method (Listing 4.9), as summarized in Table 4.1. The following subsections will discuss each function and its corresponding complexity in detail.

<table>
<thead>
<tr>
<th>API Server</th>
<th>curModel: CompactEpistemicModel</th>
</tr>
</thead>
<tbody>
<tr>
<td>generateModel(constraints: Set&lt;Formula&gt;)</td>
<td></td>
</tr>
<tr>
<td>applyEvent(event: CompactEventModel)</td>
<td></td>
</tr>
<tr>
<td>queryModel(f: Formula): bool</td>
<td></td>
</tr>
<tr>
<td>createApp(api)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.2: APIServer class UML description.

```python
1 function createApp(api):
2     api.post('/api/model', generateModel);
3     api.post('/api/apply-event', applyEvent);
4     api.get('/api/query', queryModel);
```

Listing 4.9: Create a REST API for the reasoner.

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Path</th>
<th>Mapped Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/api/model</td>
<td>generateModel</td>
</tr>
<tr>
<td>POST</td>
<td>/api/apply-event</td>
<td>applyEvent</td>
</tr>
<tr>
<td>GET</td>
<td>/api/query</td>
<td>queryModel</td>
</tr>
</tbody>
</table>

Table 4.1: The API endpoints and their mapped functions.

4.1.4.1 Model Creation

The API’s model creation function `generateModel` (Listing 4.10) initializes `this.curModel` using a set of propositional formula provided via the `constraints` parameter. This endpoint will be used by PAL-/DEL-AgentSpeak to achieve its model creation phase; recall from Chapter 3 that a SAT solver is required by the PAL-/DEL-AgentSpeak model creation process to transform propositionalized ranges and constraints into an initial epistemic model. SAT solving functionality is built into HW, and is provided via the following function:
• \( \text{solve}(S) \) iterates over the SAT solutions to a set of propositional sentences \( S \).

Each call returns a set of true propositional symbols or \text{null} if no solutions exist.

```javascript
function generateModel(constraints: Set<Formula>) {
    this.curModel = new CompactEpistemicModel();
    let wNum = 0;
    while ((cur = solve(constraints)) !== null) {
        this.curModel.addWorld(wNum++, cur);
    }
}
```

Listing 4.10: Generating a compact model from constraints.

Due to the complexity of SAT, the worst-case time and space complexity of \text{generateModel} is \( O(2^{at(S)}) \), where \( at(S) \) is the number of atomic propositions used by the sentences in \( S \). This includes the computation and storage of \( 2^{at(S)} \) potential propositional states. In the next section, we explore the model update endpoint.

4.1.4.2 Model Updates

The model update function \text{applyEvent} (Listing 4.11) is provided with a \text{CompactEventModel} that includes a set of events, pre-conditions, and post-conditions, and updates \text{this.curModel} accordingly.

```javascript
function applyEvent(event: CompactEventModel) {
    this.curModel = event.apply(this.curModel);
}
```

Listing 4.11: Apply a compact event model to the current epistemic model.

The update operation obtains a product of worlds (\( W \)) and events (\( E \)), for which the resultant model only contains worlds that model event preconditions (\( \varphi \)) and have their valuations adjusted according to post-conditions (\( P \)). In the worst-case scenario, the time complexity is \( O(|W| \times |E| \times (|\varphi| + |P|)) \) and the space complexity is \( O(|W| \times |E|) \) due to our use of compact models. In the case of public announcements
(|E| = 1), this is simplified to a time complexity of $O(|W| \times |\varphi|)$ and a space complexity of $O(|W|)$.

4.1.4.3 Model Queries

The model querying function `queryModel` (Listing 4.12) is provided with a formula (modal or propositional) and returns the model checking result based on `this.curModel`.

```python
1 function queryModel(f: Formula): bool
2 return this.curModel.modelCheck(f);
```

Listing 4.12: Query a given formula.

The worst-case time complexity for model querying is $O(|W| \times |\varphi|)$, where $|W|$ is the number of worlds in $M$ and $|\varphi|$ is the size and thus the number of evaluations required to query $\varphi$.

In summary, this section covered our provided API and the extensions we made to HW. We have presented a compact implementation for S5 epistemic and event models, which offer optimized linear time and space complexities. Additionally, we have expanded HW by introducing various API endpoints, which allow external programs to leverage our compact representations and reasoning services. These contributions are not restricted to PAL-/DEL-AgentSpeak, and can be utilized by anyone seeking to use HW for S5 epistemic reasoning, or who require PAL/DEL reasoning as an external REST API service. In the next section, we will provide implementations for PAL-/DEL-AgentSpeak and illustrate how they utilize the API to provide agents with epistemic reasoning capabilities.
4.2 Implementing PAL-AgentSpeak and DEL-Agent-Speak

In this section, we present the implementation of the PAL- and DEL-AgentSpeak function definitions and operational semantics, as detailed in Chapter 3. This section covers crucial implementation details, including the assignment of propositional formulae to ranges, constraints, and other AgentSpeak syntax, how we rewrite logical consequences to obtain rule-free formulae that can be propositionalized and interpreted by the epistemic reasoner, and the processes for creating, updating, and querying models for PAL-/DEL-AgentSpeak. This section will also demonstrate how PAL- and DEL-AgentSpeak utilize the epistemic reasoning API presented in the previous section.

Recall that AgentSpeak is an abstract language, thus the implementation of PAL-/DEL-AgentSpeak is based on Jason, the Java-based AgentSpeak interpreter and platform [1]. The complete implementation for PAL- and DEL-AgentSpeak can be accessed at https://github.com/MikeVezina/epistemic-jason.

We begin this section with a prerequisite implementation for the function which rewrites logical consequences, such that they are expressed without the use of rule-based literals. This rewriting function was formally defined in Definition 3.2.2 of the methodology chapter, and is used extensively throughout the model creation, updating, and querying processes to obtain rule-free consequence formulae.

4.2.1 Obtaining Rule-Free Consequences

We begin by introducing the standard Jason implementation for the logical consequences function, which obtains the consequences of a given literal from the belief
base. This function belongs to the class Lit, which represents literals, and serves as the parent class for logical expressions (i.e., conjunctions, disjunctions), and various other syntactic representations. This standard definition gives context to the rewriting function.

### 4.2.1.1 Standard Logical Consequences

Given a literal instance lit, the standard lit.logCons(ag, un) function (Listing 4.13) obtains the list of unifiers (Unif) that make lit with unifier un a consequence of the agent’s belief base ag.getBB(). The following functions are used within this implementation:

- ag.getBB().getCandBeliefs(Lit lit, Unif un) provides the set of beliefs or rules within the belief base that are candidates for the logical consequences of lit and un.

- unif.unifies(Lit l1, Lit l2) stores the variable unifications in unif based on l1 and l2, and l.apply(Unif unif) applies unif to the free variables in literal l.

```java
public List<Unif> logCons(Agent ag, Unif un) {
    List<Unif> results = new ArrayList<>();
    for (Lit l : ag.getBB().getCandBeliefs(this, un)) {
        Unif u = un.clone();
        if (l.isRule()) // l is a belief rule
            if (u.unifies(this, l.getHead())) {
                for (Unif resUn : l.getBody().logCons(ag, u))
                    if (u.unifies(this, head.apply(resUn)))
                        results.add(u);
            } else // l is explicit Belief
                if (u.unifies(this, l))
                    results.add(u);
    }
    return results;
}
```

Listing 4.13: The original logical consequences function for literals.
The worst-case computational complexity for the standard logical consequences function must account for the potential recursive evaluation of belief rules \( R \) and rule bodies containing beliefs \( B \); thus the worst-case time and space complexity is \( O(|B|^{|R|}) \) [51]. In practice, this depends heavily on how many literals require nested evaluation of rules. In the following subsection, we modify this function such that we obtain all rule-free formula that provide the logical consequences of literals.

### 4.2.1.2 Rewriting Rule-based Logical Consequences

Given a literal instance \( \text{lit} \), \( \text{lit.rwCons}(\text{ag}, \text{un}) \) function (Listing 4.14) obtains the list of pairs \(<\text{Lit}, \text{Unif}>\) where \( \text{Lit} \) is the rule-free consequence formula for the corresponding unifier \( \text{Unif} \). This function executes the exact same as \( \text{lit.logCons} \), but includes the explicit belief literals used to obtain the consequence unifiers.

```java
public List<<Lit, Unif>> rwCons(Agent ag, Unif un)
    List<<Lit, Unif>> results = [];
    for(Lit l : ag.getBB().getCandBeliefs(this, un))
        Unif u = un.clone();
        if(l.isRule()) // l is a belief rule
            if(u.unifies(this, l.getHead()))
                for(<Lit,Unif> res : l.getBody().rwCons(ag, u))
                    if(u.unifies(this, head.apply(res.unif)))
                        results.add(<res.lit, u>);
            else // l is explicit Belief

            if(u.unifies(this, l))
                results.add(<this, u>);
    return results;
```

Listing 4.14: The function for rewriting the logical consequences of literals.

The worst-case time and space complexity for \( \text{rwCons} \) is the same as \( \text{logCons} \), \( O(|B|^{|R|}) \); the appended return value is insignificant to the worst-case complexity.

The following subsections will cover the integration of PAL-AgentSpeak and DEL-AgentSpeak’s operational semantics for model creation, updating, and querying.
with the standard reasoning cycle implementation provided by Jason. To highlight
the differences between PAL-AgentSpeak and DEL-AgentSpeak, we distinguish be-
tween the two extensions using conditional blocks labelled [PAL-AgentSpeak] and
[DEL-AgentSpeak].

4.2.2 Model Creation

In Jason, the agent is initialized using a function `Create(String src)` which creates an
instance of the `Agent` class based on the parameter `src` holding the AgentSpeak source
code.

The model creation process for both PAL- and DEL-AgentSpeak override `Create
(String src)` with the implementation shown in Listing 4.17, utilizing `loadRanges()`
(Listing 4.15) and `getCons()` (Listing 4.16) to load the corresponding set of ranged
literals and to obtain the propositional formula describing the ranges and constraints
defined by the agent. Lastly, the overridden `Create` function invokes the model creation endpoint for the epistemic reasoning API presented in Section 4.1. For reference,
this implementation corresponds to the methodology for model creation presented
in Sections 3.2.5.1 (PAL-AgentSpeak) and 3.3.3.1 (DEL-AgentSpeak). Recall that
DEL-AgentSpeak includes all beliefs in the initial model and must create propositional
sentences for all initial explicit beliefs.

```java
1 public void loadRanges()
2     for (var cons : new Lit("range(Var)").rwCons(this, new Unif()))
3         var range = cons.unif.get("Var"); // Var is unified to range lit
4             this.ranges.add(range);
5         this.ranges.add(~range);
```

Listing 4.15: Load all ranged literals.
public List<Formula> getCons()
    var forms = [];
    for (var range : this.getRanges())
        // 1. Assign range semantics
        forms.add(~range ∨ range);
        // 2. Find range constraints + assign semantics
        for (var rw : range.rwCons(this, new Unif()))
            forms.add(rw.lit → range);
    if (DEL-AgentSpeak) // Add explicit initial beliefs
        for (var bel : this.getBB().getInitial())
            forms.add(bel);
    return prop(forms);

Listing 4.16: Get all model constraints.

public static Agent create(String src, ...)
    // Standard: Initialize agent from source (beliefs, plans, etc.)
    Agent ag = new Agent();
    ag.load(src);
    // Load PAL/DEL Components
    ag.loadRanges(); // Load \( a_{\text{R}} \) from belief base
    List<Formula> cons = ag.getCons(); // Call all_cons/all_cons*
    API.post("/api/model", cons); // Request model creation
    return ag;

Listing 4.17: Initialize the PAL-/DEL-AgentSpeak agent.

4.2.2.1 Complexity

The computational complexity for model creation is identical for both DEL-AgentSpeak and PAL-AgentSpeak. The complexity is primarily determined by the number of ranged literals \(|R|\) and constraints \(|C|\). Model creation involves two complex operations:
1) obtaining the rewritten logical consequence formulae for constraints: \text{getCons}(),
   and 2) generating the model: \text{API.post("/api/model", cons)}. The resulting time and
   space complexity is thus a summation of these operation complexities: \(O(|R|^{|C|} + 2^{|R|})\).
   The additional belief constraints included by DEL-AgentSpeak are linear and do not
   impact the overall complexity. This operation will vary heavily depending on the
   ranges, constraints, and rule definitions described by the agent.
The next subsection presents the implementation for the model update process, including its integration with the reasoning cycle and its interaction with the epistemic reasoning API.

4.2.3 Model Updating

This section provides the implementation for integrating the PAL-/DEL-AgentSpeak model update process with the reasoning cycle, as introduced in Sections 3.2.5.2 (PAL-AgentSpeak) and 3.3.4.3 (DEL-AgentSpeak).

Let us provide a brief introduction to Jason’s belief revision function, denoted by 
\[ \text{[adds,rem]} = \text{brf(Lit add, Lit rem)} \]. This function is overridden by agents to ensure belief consistency, and takes either a belief to be added \((\text{add} \neq \text{null})\) or a belief to be removed \((\text{rem} \neq \text{null})\) as inputs. Upon revision, the function returns the set of literal additions \text{adds} and removals \text{rems} that result from the revision. We will use these additions and removals as the basis for our epistemic events.

Listing 4.18 provides the update function for DEL-AgentSpeak, which iterates all effective belief additions and removals, finds all “on” plans, creates their relevant CompactEventModel (Section 4.1) representation, and applies the DEL event to the current epistemic model via the epistemic reasoning API. We do not provide implementations for the functions \text{find_on}, \text{ontic_lits}, and \text{del_em}, previously defined in Section 3.3.4.3, as their implementations are trivial.

A similar process is followed in Listing 4.19, where a public announcement event model is created based on ranged belief additions. The function creates an instance of a CompactEventModel based on createCompactEvent from Listing 4.5. Recall for PAL-AgentSpeak that only ranged literal additions will be applied to the epistemic model;
additions for non-ranged beliefs are ignored, and PAL-AgentSpeak does not support belief deletions.

```java
public void delUpdate(List<Lit> adds, List<Lit> rems)
for (var add : adds)
    var onPlans = find_on(this.getPlans(),
        (this.getBB() ∪ this.getRanges()), new Ev(+add));
this.ranges.addAll(ontic_lits(onPlans));
CompactEventModel delModel = del_em(onPlans);
API.post("/api/apply-event", delModel);
for (var rem : rems)
    var onPlans = find_on(this, new Event(-rem));
    CompactEventModel delModel = del_em(onPlans);
    API.post("/api/apply-event", delModel);
```

Listing 4.18: DEL model updates.

```java
public void palUpdate(List<Lit> adds, List<Lit> rems)
for (var add : adds)
    if (this.isRanged(add))
        CompactEventModel palModel = createAnn(add);
    API.post("/api/apply-event", palModel);
public CompactEventModel createAnn(Lit add)
    return createCompactEvent({e}, {e: prop(add)}, {});
```

Listing 4.19: PAL model updates.

```java
public void addBel(Lit l)
List<List<Lit>> [adds, rems] = brf(l, null); // Call agent's brf
if (DEL-AgentSpeak) // Call DEL-AgentSpeak update
    this.delUpdate(adds, rems);
else if (PAL-AgentSpeak) // Call PAL-AgentSpeak update
    this.palUpdate(adds, rems);
```

Listing 4.20: Add a belief to the belief base.
4.2.3.1 Complexity

In the case of PAL-AgentSpeak, effective ranged literal belief additions execute with a public announcement update, thus leading to a linear time and space complexity based on the number of worlds $W$ in the epistemic model: $O(|W|)$.

In contrast, DEL-AgentSpeak updates scale depending on the event model being created. Event models are created based on applicable “on” plans ($A$) defined by the agent. In the case where default event models are used (i.e., $A = \emptyset$) the update operation executes in linear time with respect to the number of worlds: $O(|W|)$. Otherwise, the complexity is based on the number of “on” plans $|A|$, the size of their preconditions $|\varphi|$, and their post-conditions $|P|$. The worst-case time complexity of an update is thus: $O(|W| \times |A| \times (|\varphi| + |P|))$ and the space complexity is $O(|W| \times |A|)$.

In the next section, we present the implementation and complexity for querying certain and possible beliefs in PAL- and DEL-AgentSpeak.

4.2.4 Model Querying

In Jason, beliefs are typically queried via the standard `logCons` function introduced earlier in Listing 4.13. As such, this function is used to evaluate the contexts of relevant plans, evaluate test goals, and perform other forms of belief queries. In Sections 3.2.5.3 (PAL-AgentSpeak) and 3.3.5.1 (DEL-AgentSpeak), we adapted these
functions such that the epistemic model is used for determining entailment of beliefs.

To achieve this in our implementation, we override the `logCons` function to use the epistemic reasoner for entailment, as shown in Listing 4.22.

```java
public List<Unif> logCons(Agent ag, Unif un)
    List<Unif> results = []
    for(<Lit, Unif> rw : this.rwCons(ag, un))
        if (DEL-AgentSpeak || (PAL-AgentSpeak && isRanged(this)))
            Modality modal = this.getFunctor() == "poss" ? □B : ◊B;
            if(entailed(modal, rw.lit))
                results.add(rw.unif);
        else
            results.add(rw.unif);
    return results;

public Boolean entailed(Modality modal, Lit lit)
    ModalFormula formula = new ModalFormula(modal, prop(lit));
    return API.get("/api/query", formula);
```

Listing 4.22: The updated logical consequences function for literals.

### 4.2.4.1 Complexity

Both the PAL- and DEL-AgentSpeak extensions utilize `rwCons` to obtain formulae that can be propositionalized and evaluated by the reasoner. This process has a complexity of $O(|B|R)$ with $B$ literals and $R$ rule invocations; this brings the same complexity as querying beliefs in standard Jason. We exceed the standard computational complexity when we evaluate formulae using the reasoner; given $|\varphi|$ rewritten formulae, the complexity for evaluation using the epistemic model is based on the number of worlds $|W|$: $O(|W| \times |\varphi|)$.

This concludes the implementation section for PAL-AgentSpeak and DEL-AgentSpeak. This section provided various insights into the critical implementation details of PAL- and DEL-AgentSpeak, and the corresponding time and space complexities. We conclude the chapter with an overview of the contributions and concluding remarks.
4.2.5 Concluding Remarks

To summarize, this chapter has covered the implementation of PAL- and DEL-AgentSpeak, and their integration with Jason’s implementation of the AgentSpeak reasoning cycle. We have provided a detailed description of the implementation and complexity analysis for the operations related to model creation, updating, and querying. Our implementation utilized the epistemic reasoning API to facilitate communication with the epistemic reasoner, Hintikka’s World.

In addition to extending Hintikka’s World with an API, we also introduced compact S5 epistemic/event models. Since PAL-/DEL-AgentSpeak strictly relies on S5 models, we implemented these models in a compact form, resulting in significant improvements to model initialization and updating time and space complexities; we ultimately improved these operations from quadratic to linear with respect to the number of worlds. This optimization had a direct impact on the complexities associated with PAL-/DEL-AgentSpeak’s usage of the API endpoints.

In the next chapter, we apply our implementations for PAL-AgentSpeak and DEL-AgentSpeak to several domains of uncertainty to examine their utility across various agent problems and domains. The following chapter, Chapter 6 will scale the complexity of uncertainty and examine its impact on the performance of PAL-/DEL-AgentSpeak.
Chapter 5

Application

In this chapter we look at a few applications of the proposed extensions PAL-AgentSpeak and DEL-AgentSpeak. In Section 5.1, we present a complete DEL-AgentSpeak listing that leverages possibilistic reasoning for map navigation to accomplish tasks in the 2019 MAPC; this extends the running navigation example presented throughout Chapter 3. In Section 5.2, we extend the navigation program with the ability to reason about agent identities in order to enable collaboration between teammates. Lastly, Section 5.3 presents a PAL-AgentSpeak program that reasons about uncertainty given knowledge revelations in the classic game of Minesweeper.

Throughout this chapter, we provide the complete PAL-AgentSpeak or DEL-AgentSpeak listings. Additionally, we provide a brief walkthrough of the execution of these programs so that the reader has a better understanding of how PAL-AgentSpeak and DEL-AgentSpeak operate within these domains.
5.1 MAPC: Navigating Under Uncertainty

This section presents the DEL-AgentSpeak program for achieving tasks in the 2019 MAPC via the navigation approach we used as a running example in Chapter 3. In contrast to the running example, additional entities are added to the map definition, and the agent is provided with additional actions and perceptions that allow it to complete tasks. The following subsection highlights these changes in more detail. Subsequently, we present the modified DEL-AgentSpeak program and offer the reader a walkthrough of the agent’s program execution.

5.1.1 Environment

A custom simulator for the 2019 MAPC was developed to simplify and visualize the uncertainty experienced by the agent. Figure 5.1 presents a 5x5 map populated with obstacles (dark grey), goals (yellow), and dispensers (red). In comparison to the running example, the agent must first dispense a block from a dispenser before visiting the goal cell. The perceptions and actions available to the agent will follow.

![Figure 5.1: A 5x5 map with dispensers (red), obstacles (dark grey), and goals (yellow).](image-url)
5.1.1.1 Perceptions and Actions

Recall that all directions are relative: \( \text{Dir} \) is one of \( \text{up} (\uparrow), \text{down} (\downarrow), \text{left} (\leftarrow) \) or \( \text{right} (\rightarrow) \). The following perceptions are given by the environment:\(^1\):

- \( \text{at}(\text{Obj}) \): the agent is at a location with \( \text{Obj} \in \{\text{goal, disp}\} \). This predicate is used for termination of the \( !\text{nav} \) plan and for the \( \text{dispense} \) action.

- \( \text{obs}(\text{Dir}) \): the agent perceives an obstacle (\( \text{obs}(\text{Dir}) \)) or no obstacle (\( \sim \text{obs}(\text{Dir}) \)) at its current location.

- \( \text{moved}(\text{Dir}) \): this perception is received after a successful \( \text{move}(\text{Dir}) \) action.

The agent has access to various actions that it can use in its plan definitions to interact with the environment. The actions are defined as follows:

- \( \text{move}(\text{Dir}) \): the agent moves to the cell in the given direction (\( \text{Dir} \)).

- \( \text{dispense} \): dispense a block when \( \text{at}(\text{disp}) \).

- \( \text{submit} \): submit the task when the agent has a block and \( \text{at}(\text{goal}) \).

Using the navigation approach presented in Chapter 3, the agent must perform the following to complete its task.

1. Navigate and dispense a block from a dispenser, then

2. Navigate and submit the block on a goal cell.

These objectives are represented by the goal \( !\text{completeTask} \), which will be defined by the DEL-AgentSpeak program presented in the next subsection.

\(^1\)Algorithm 1 (from Appendix A) presents the pseudocode for generating perceptions given the current environmental state.
5.1.2 The DEL-AgentSpeak Program

We present the DEL-AgentSpeak program (Listing 5.1) for our navigation agent with
minor adaptations that accommodate the changes stated above. The dir(Dir, Obj) rules now accommodate the shortest path direction Dir to the closest object Obj ∈ {goal, disp}; plans for !nav(Obj) also take this change into account. An exhaustive listing of all obs and dir rules is given in Appendix A.1.

The plan on Line 20 implements the !completeTask goal, relying on !nav(Obj) subgoals for navigation. Two !nav(Obj) plans are defined. The plan on Line 27 handles the case where the agent has reached its destination, i.e., a goal or disp. Otherwise, the plan on Line 30 moves the agent in a certain or possible direction, recursively introducing !nav(Obj) until the agent reaches its destination.

5.1.2.1 Program Execution

Initially, all navigation directions are possible since the agent considers all locations possible. As the agent eliminates impossible locations through incoming perceptions and their respective “on” plans, it implicitly eliminates the navigation directions that do not lead to its desired destination.

We list the general execution steps for the program:

1. **(Initialization)** DEL-AgentSpeak performs the model creation process, creating the initial epistemic model corresponding to the ranges and constraints.
   - Except for obstacle locations (1,2) and (2,2), the initial model contains a world with valuation \{loc(x, y)\} corresponding to each location (x, y).

2. **(Perceive)** Unbeknownst to the agent, its location is (1,1). Initial perception events for +obs(↓) and +~obs( _) otherwise are transformed into the following
Listing 5.1: The DEL-AgentSpeak program for single-agent task building and navigation in the MAPC.
DEL event models through the “on” plans and applied to the model in sequence. These event models remove worlds that do not match the corresponding obstacle perceptions.

(a)  (Line 35) \[ E = \{\downarrow\}, \text{pre}(\downarrow) = \text{obs}(\downarrow), \text{post}(\downarrow) = \} \]

(b) For all \( D \in \{\uparrow, \leftarrow, \rightarrow\} \):

\[ \langle E = \{D\}, \text{pre}(D) = \neg \text{obs}(D), \text{post}(D) = \rangle \]

3. (Line 20) The \( !\text{completeTask} \) plan executes, introducing the sub-goal \( !\text{nav} (\text{disp}) \).

4. (Line 30) The appropriate \( +!\text{nav} \) plan is executed as follows:

(a) The next best movement direction \( \text{Dir} \) is obtained via the plan context’s effective evaluation of \( \Box_B \text{dir} (\text{Dir}, \text{goal}) \lor \Diamond_B \text{dir} (\text{Dir}, \text{goal}) \). In this case, \( \text{Dir} = \rightarrow \) is selected, though another option is \( \text{Dir} = \leftarrow \). Unifications for \( \text{Dir} \) prioritize certainty \( \text{dir}(\text{Dir}, \text{goal}) \) over uncertainty \( \text{poss} (\text{dir}(\text{Dir}, \text{goal})) \) due to AgentSpeak’s short-circuiting of the \( \lor \) operator.

(b) \( \text{move}(\rightarrow) \) is performed. The agent moves to \((2,1)\) and the agent receives perception events \( +\text{moved}(\rightarrow), +\text{obs}(\downarrow) \), and \( +\neg\text{obs}(\_\_) \) otherwise. The following sequence of DEL event models is applied:

\[ \langle E = \{\rightarrow\}, \text{pre}(\rightarrow) = [\text{loc}(X, Y) \land X \leq 3], \text{post}(\rightarrow) = \{\text{loc}(x, y) = \bot, \text{loc}(x + 1, y) = \top}\} \]

- The same event models are used for \( +\text{obs}(\downarrow) \) (Line 35) and \( +\neg\text{obs}(\_\_) \) (Line 36) as previously mentioned.

\(^2\text{Note that since } \text{dir} \text{ is resolved using rules, these formulae are rewritten by DEL-AgentSpeak in terms of loc.}\)
5. (Line 27) Once the agent reaches its destination, the base case of the $!\text{nav}$ goal executes.

6. (Line 22) Program execution returns to $!\text{completeTask}$, which then dispenses a block using the action $\text{dispense}$. The above navigation process is repeated for $!\text{nav}(\text{goal})$, which then allows the agent to $\text{submit}$ its block.

To help solidify the reader’s understanding of how the agent’s uncertainty changes over time, Figure 5.2 presents the change in possible locations as the agent perceives obstacles and navigates the map. Locations with a red circle are those that the agent considers possible. Once the agent obtains full certainty of its location, navigation becomes trivial.

Figure 5.2: The evolution of uncertainty as the agent navigates to the dispenser.

This concludes the navigation application presented using DEL-AgentSpeak. In this section, we introduced a single-agent program where the agent uses DEL to localize itself on the map and provide best-effort navigation. Since the 2019 MAPC is a multi-agent contest where agents must collaborate to build multi-block tasks and
receive larger rewards, another source of uncertainty is the identity of perceived agents. In the next section, we build an identification approach on top of the navigation program.

5.2 MAPC: Identifying Agents for Collaboration

As mentioned in the previous section, the 2019 MAPC organizes agents into teams which must collaborate with each other to assemble multi-block task requirements. Each agent must localize, navigate, and obtain blocks from dispensers individually, but must synchronize when collaborating on tasks. To synchronize, agents must be able to identify perceptions of friendly agents. Unfortunately, the perceptions provided by the MAPC environment do not include the identities of friendly agents, and obtaining identities is non-trivial without additional information. In this section, we focus on addressing the management of uncertainty encountered during the agent identification process, and compare a standard AgentSpeak solution that relies on an ad hoc approach to a solution that relies on DEL-AgentSpeak. We introduce an identification approach that expands upon the possibilistic localization approach presented in Section 5.1.

5.2.1 Environment

In this section, we evaluate a static situation in which a friendly agent is perceived and we must distinguish and identify them. We extend the navigation simulation in Section 5.1 to include two additional agents; the situation under consideration is shown in Figure 5.3. Similar to the perception of obstacles, agents are perceived when they are in horizontally- or vertically-adjacent cells. In the figure, we take on Alice’s
perspective (the blue agent, location (1,1)), who unknowingly perceives Bob (the red agent, location (2, 1)). Charlie (the green agent, location (1, 3)), is not within the range of perception of Alice or Bob. Alice, Bob, and Charlie all run their own agent programs, which model their respective possible locations.

Since PAL- and DEL-AgentSpeak extend Jason’s implementation of AgentSpeak, they also benefit from its other amenities, such as agent messaging. We do not dive into the details of messaging, but simply assume that beliefs can be transferred between agents. In this case, the agents communicate their current possible locations to one another. We thus use the following beliefs throughout this section:

- **agent**(R\_X, R\_Y): an agent is perceived in an adjacent cell with relative location (R\_X, R\_Y).
- **locs**(Ag, loc(X, Y)): agent Ag ∈ {bob, charlie} has informed us that they currently consider **poss**(loc(X, Y)).

We propose the following identification strategy. When an agent is perceived, we cross-reference the known relative location with all of our teammates’ possible locations in order to eliminate any impossible identities. This challenge requires
complete certainty of identities before proceeding with collaboration; thus we can not
collaborate merely on possibilities. As such, this challenge highlights techniques for
management of uncertainty, i.e., managing which agent identities are possible, and
being able to distinguish certainty and collaborate when it arises.

The elegant solution to the identification problem is provided using DEL-AgentSpeak
in Section 5.2.3. However, we first provide readers with an understanding of the degree
of expressiveness made possible by DEL-AgentSpeak and its “on” plans, and the impli-
cations of managing uncertainty without the aid of an extension like DEL-AgentSpeak.
To this end, we start by demonstrating how identification would be carried out in
standard AgentSpeak, without incorporating any language extensions.

5.2.2 The Standard AgentSpeak Program

In this subsection, we entertain a standard approach to managing possibilities without
an extension such as PAL-/DEL-AgentSpeak. In this subsection only, poss(ℓ) is no
longer a special operator but rather a belief that the agent explicitly models in its belief
base. The common theme throughout this section is that the manual management of
uncertainty is ad hoc, burdensome, and takes away from the true intent of the agent’s
program.

Listing 5.2 outlines the plan implementation that is executed when an agent
perceives +agent(R_X, R_Y). In the plan body, we cross-reference all potential locations,
poss(loc(X, Y)), with those deemed possible by other agents. This is accomplished
through several for loops which iterate all friendly agents Ag that have possible
locations with delta (R_X, R_Y). All agents Ag that have at least one location which
match this delta are considered possible with: +poss(id(Ag)).
After cross-referencing all locations, we employ the built-in internal action `.count(poss(id(_)), 1)` to confirm that there is exactly one possibility (i.e., certainty) before proceeding with collaboration. We then unify the agent’s certain identity via the test goal `?poss(id(Ag))` and continue with collaboration, denoted arbitrarily by the action `collabWith(Ag)`. Finally, when agents decide to move and no longer perceive each other, the plan for `−agent(R_X, R_Y)` removes all added possible identities.

```prolog
+agent(R_X, R_Y)
   <- for(poss(loc(X, Y))) { // Iterate our possible locs
      for(locs(Ag, loc(X + R_X, Y + R_Y))) {
         +poss(id(Ag)); // Add possible identity using "on" plan
      }
   }.
   .count(poss(id(_)), 1); // Only permit certainty of the agent
?poss(id(Ag)); // Obtain single agent identity
   collabWith(Ag).

-agent(R_X, R_Y)
   <- for(poss(id(Ag))) {
      -poss(id(Ag)); // Remove all possible agents
   }.
```

Listing 5.2: The plan implementation for agent identification using the possible locations of each agent.

When an agent does not utilize extensions (such as PAL-/DEL-AgentSpeak) for possibilistic uncertainty reasoning, they must store, manage, and query their possibilities in an ad hoc manner. Even in simple examples like agent identification, ad hoc management of possibilities becomes a burdensome task that detracts from the core objective of the program. This issue is further exacerbated as uncertainty scales. In the case of the above listing, most of the `+agent(R_X, R_Y)` plan is dedicated to iterating and managing possibilities rather than initiating collaboration with the other agent.
The PAL-/DEL-AgentSpeak extensions provide the ability to facilitate declarations of uncertainty that minimize the management needed by the agent. The subsequent subsection presents the DEL-AgentSpeak equivalent for the above listing and demonstrates how using DEL-AgentSpeak can alleviate the burden of uncertainty reasoning for both the agent program and the developer.

### 5.2.3 The DEL-AgentSpeak Program

This section presents the DEL-AgentSpeak agent identification program. Listing 5.3 provides the relevant identification program, which is functionally equivalent to the listing provided for standard AgentSpeak (Listing 5.2) in the previous section.

```plaintext
1 // Create uncertainty at runtime
2 +on(agent(R_X, R_Y)) : loc(X,Y) & locs(Ag, loc(X+R_X,Y+R_Y))
3   <- +id(Ag).
4 +agent(R_X, R_Y) : id(Ag)
5   <- collabWith(Ag).
6 -on(agent(R_X, R_Y)) : id(Ag)
7   <- -id(Ag).
```

Listing 5.3: The DEL-AgentSpeak program for agent identification.

The “on” plan defined on Line 2 is transformed and executed as a DEL event immediately upon receiving the perception `agent(R_X, R_Y)`. When the DEL event is applied to the epistemic model, the proposition `id(Ag)` is added to each possible world where our location, `loc(X,Y)`, has an agent `Ag` located at `loc(X + R_X, Y + R_Y)`. If in all possible worlds only one agent identity is possible, then the agent becomes certain of this identity. This process will be further illustrated when we trace the execution of the program.

Once the “on” plan has been processed, the standard plan for `+agent(R_X, R_Y)` is selected and executed. The context queries the certainty of the agent’s identity, i.e.,
□ $B$ id($Ag$), and subsequently collaborates with the agent using the arbitrary action $\text{collabWith}(Ag)$. In comparison to the standard AgentSpeak program (Listing 5.2), there is no need for explicit management of uncertainties. Through the declaration of “on” plans, DEL-AgentSpeak efficiently updates possibilities as needed. As a result, the DEL-AgentSpeak program effectively captures uncertainty while maintaining the primary focus of the agent program.

### 5.2.3.1 Program Execution

We use the concrete example presented in Figure 5.3 to demonstrate the execution of the DEL-AgentSpeak program presented in Listing 5.3. Our agent, Alice, currently considers the possible locations $\text{loc}(1,1)$ and $\text{loc}(2,1)$ represented by the following set of worlds:

- $\{\text{loc}(1,1)\}$
- $\{\text{loc}(2,1)\}$

Additionally, all worlds contain the following beliefs which have been provided by Bob and Charlie:

- Bob’s possible locations: $\text{locs}(bob, \text{loc}(1,1))$ and $\text{locs}(bob, \text{loc}(2,1))$.
- Charlie’s possible locations: $\text{locs}(charlie, \text{loc}(1,3))$ and $\text{locs}(charlie, \text{loc}(2,3))$.

In the current example, the agent perceives the agent to its right, which raises the perception event:

- $+_\text{agent}(1,0)$: an agent is perceived to the right.

The “on” plan defined on Line 2 is transformed into the following relevant DEL event model with $E = \{e_{1,1}, e_{2,1}\}$:
\[ \text{pre}(e_{1,1}) = [\text{loc}(1,1) \land \text{locs}(\text{bob}, \text{loc}(2,1))], \text{post}(e_{1,1}, \text{id}(\text{bob})) = \top, \text{ which identifies Bob to the right of our location (1,1).} \]

\[ \text{pre}(e_{2,1}) = [\text{loc}(2,1) \land \text{locs}(\text{bob}, \text{loc}(3,1))], \text{post}(e_{2,1}, \text{id}(\text{bob})) = \top, \text{ which identifies Bob to the right of our location (2,1).} \]

Note that there are no unifying values for Charlie, and as such, no events exist in the event model. After applying this event model, the resultant epistemic model is as follows:

\[ \{\text{loc}(1,1), \text{id}(\text{bob}), \text{locs}(\text{bob}, \text{loc}(2,1)), \ldots\} \]

This remaining model provides certainty that the identified agent is Bob. However, notice that the perception of Bob also eliminates uncertainty about our location; given that no friendly agents consider \text{loc}(3,1) possible, we confidently eliminate the possible world modelling the location \text{loc}(2,1).

The standard plan for \(+\text{agent}(1,0)\) executes, which, given certainty that this perceived agent is Bob, allows us to fulfill the plan by collaborating with him. For completeness, we also define the “on” plan describing the removal of \(-\text{agent}(1,0)\).

When this DEL event model executes, it removes all identity propositions from the current set of possible worlds, thus allowing the identification process to repeat upon the next agent perception.

In conclusion, when agents do not employ extensions like PAL-/DEL-AgentSpeak for possibilistic uncertainty reasoning, they are left to manage and query possibilities in a manner that is ad hoc. This ad hoc approach burdens the developer, even in simple scenarios like agent identification, and ultimately detracts from the program’s main objectives, due to the need to manage manually uncertainty. When expressing the identification challenge using DEL-AgentSpeak, the agent describes the changes
to uncertainty in a declarative manner, and can thus rely on the transformational semantics provided by the extension to update its possibilities accordingly. This largely reduces the management effort required by the agent, and maintains the focus of the program to be its intended objectives. Additionally, as we saw in the program trace, uncertainty about the agent’s identity also reduced the certainty we had about our location without the need to be explicitly modelled by the agent or developer.

As demonstrated, DEL-AgentSpeak equips agents with an intuitive, yet powerful means for handling uncertainty through the use of DEL. By employing possibilistic reasoning, agents can be localized, resulting in improved navigation (Section 5.1) and can be leveraged to enable agent identification, two critical challenges that involve uncertainty in the 2019 MAPC.

In the following section, we present a different domain altogether. To show that our contributions are general and not limited to the 2019 MAPC, we apply PAL-AgentSpeak to the uncertain domain of Minesweeper.

5.3 Minesweeper

Minesweeper is a classic game of knowledge where the objective is to clear a board containing hidden mines. Each cell on the game board may contain either a mine or a numerical hint indicating the number of adjacent cells that contain mines. The agent knows about the total number of mines and may also be provided with some initially-revealed hints. The board itself does not change; the agent iteratively reveals cells until a mine is hit (loss) or when all non-mine cells are clicked (win). Since the Minesweeper domain consists solely of revelations of a static state, it is a perfect candidate for the simpler PAL-AgentSpeak extension.
5.3.1 Environment

Let’s consider a game of 3x3 Minesweeper with 2 mines; the agent’s visible initial state is shown in Figure 5.4.

![Figure 5.4: The initial Minesweeper situation.]

Given that there are two mines total, we can enumerate the four possible states $s_1 \ldots s_4$ as shown from left to right in Figure 5.5 (note that ‘M’ is a mine and the coloured cells are those that have been revealed to the agent).

![Figure 5.5: The enumeration of initial possible Minesweeper states.]

Now we will introduce the perceptions and actions available to the agent.

5.3.1.1 Perceptions and Actions

Cells (X, Y) are referenced from (1,1) to (3,3). The following perceptions are available:

- $\text{hint}(X, Y, N)$ indicates that the cell (X, Y) contains $N$ horizontally, vertically, and diagonally adjacent mines.
- $\text{status}(\text{win}) / \text{status}(\text{lose})$ indicates the agent has won (revealed all mine-free cells) or lost (clicked on a mine).

The agent also uses the literal $\text{mine}(X, Y)$ to keep track of potential mines and has access to an action $\text{click}(X, Y)$ that allows the agent to reveal the cell (X, Y).
Before presenting the PAL-AgentSpeak program for this agent, we briefly discuss how we plan to model the initial epistemic state.

5.3.1.2 Initial State

As shown previously in Figure 5.5, there are 4 possible states to consider, each with their own respective hints and mines. We plan to model these states with an initial epistemic model, described as $M_{MS} = (W, V)$ where:

- $W = \{s_1, s_2, s_3, s_4\}$, the set of possible state enumerations

- For all $s_i \in W$:
  - $state(i) \in V(s_i)$ is a prop representing the possible state enumeration,
  - $mine(X, Y) \in V(s_i)$ if cell $(X, Y)$ is a mine in state $s_i$,
  - $hint(X, Y, N) \in V(s_i)$ if cell $(X, Y)$ has $N$ adjacent mines (horizontal, vertical, and diagonal) in state $s_i$

For example, world $s_1$ has state $V(s_1) = \{state(1), mine(2, 1), \ldots, hint(1, 1, 1)\}$. The next section introduces the PAL-AgentSpeak program for our minesweeper agent.

5.3.2 The PAL-AgentSpeak Program

Figure 5.4 provides the listing for the minesweeper PAL-AgentSpeak program. The ranges and constraints create an initial epistemic model similar to the ideal model described in the previous subsection, but the model’s valuation only captures state propositions (e.g., $state(1), \ldots$). Instead of explicitly modeling hints and mines in the model, they are implicitly handled by rewriting the logical consequences of the rules.
// Range and constraints: state(S)
range(state(S)) :- member(S, [none, 1, 2, 3, 4]).
~state(S) :- state(S2), S != S2. // States are mutually exclusive

/* Remove case of all states being false */
state(none) :- ~state(1) & ... & ~state(4).
~state(none).

// Rules for hints
hint(1,1,0) :- state(4).
hint(2,1,1) :- state(4).
... // etc. for all hint(X, Y, N) :- state(1/2/3/4).

// Rules for mines
mine(3,1) :- state(4).
mine(1,3) :- state(4).
... // etc. for all mine(..) :- state(1/2/3/4).
~mine(X, Y) :- not(mine(X, Y)).

!play.
+
play : status(Status)
<- print("The game has ended with: ", Status).
+
play : ~mine(X, Y) | poss(~mine(X, Y))
<- +click(X, Y);
!play.

Listing 5.4: The PAL-AgentSpeak program for Minesweeper.

starting on lines 10 and 15. The listing also ensures that exactly one state is always possible through the constraints described for state(none).

The initial goal of the agent is !play. The plan on Line 22 handles the case when the end-game perception status(_) is given, and prints out a relevant message. Alternatively, when the game has not ended, the plan on Line 25 will execute by obtaining a cell (X, Y) that the agent certainly or possibly does not contain a mine; as expected, the agent prioritizes certain non-mine cells (~mine(X,Y)) over those that may be possible (~poss(mine(X,Y))).
When the agent clicks on a cell \((X, Y)\), its perceptions are updated with new hints and applied to the epistemic model as public announcements. This in turn provides new certain or possible locations that will not contain a mine. The \(\textit{!play}\) plan is recursively introduced until the agent wins or loses the game.

\subsection*{5.3.2.1 Program Execution}

Initially, one can see that location \((1, 1)\) will certainly not contain a mine. As such, the initial epistemic model leads to entailment of \(\sim \text{mine}(1, 1)\) and thus the agent performs \(\text{click}(1, 1)\). The environment responds with a perception event \(+\text{hint}(1, 1, 1)\) that reveals a hint of 1 in the clicked cell. This addition is applied to the initial model \(M_{MS}\) as \(M_{MS} \otimes [\text{hint}(1, 1, 1)]\) which eliminates possible states 1 and 4. The resulting model is shown in Figure 5.6

\begin{figure}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
1 & M & 1 \\
\hline
1 & 2 & 1 \\
\hline
M & 1 & 0 \\
\hline
\end{tabular}
\end{figure}

Figure 5.6: The enumeration of possible Minesweeper states after \(+\text{hint}(1, 1, 1)\).

From these possible states, the agent is no longer certain of which cell to click next, as it is possible for any non-clicked cell to contain a mine. The agent must pick a cell to click at random — in this case, the \(\textit{!play}\) plan chooses \texttt{poss}(\(\sim\text{mine}(2, 1)\)) and executes \(\text{click}(2, 1)\). By sheer luck, the agent receives a hint perception: \(+\text{hint}(2, 1, 1)\). The model resulting from this update is shown in Figure 5.7. This perception leads to a single possible state, where the agent has full certainty over where the mines are located. The agent is has eliminated all uncertainty of its current state and can deliberate perfectly as it is certain of all mine locations.
In this chapter, we have presented three significant applications of PAL-AgentSpeak and DEL-AgentSpeak. We equipped our 2019 MAPC agent with navigation and identification capabilities when faced with locational uncertainty using DEL-AgentSpeak, while we used PAL-AgentSpeak to develop an agent that reasons about uncertainty in a game of Minesweeper. The MAPC identification example also highlighted the benefits of PAL-/DEL-AgentSpeak over standard AgentSpeak for expressing changes to uncertainty from the perspective of agent development, and for maintaining the focus of the program’s intent on describing the operation of the agent. Furthermore, this example made it clear that various sources of uncertainty within the same domain may positively interact with each other without requiring explicit modelling by the developer. For instance, identification of a friendly agent implicitly helped us to reduce the uncertainty we had about our location.

On the general applicability of the works presented in this thesis, the 2019 MAPC is an appropriate representation of the class of uncertainty faced by typical AgentSpeak agents, as the MAPC exists as a simulation of realistic uncertainty typically encountered by these agents [55]. Additionally, due to their integration with PAL and DEL, the proposed AgentSpeak extensions are also amenable to domains that exist in the PAL and DEL planning literature (see [54]), for example, security [56] and robotics [57]. Thus, the applications of PAL- and DEL-AgentSpeak are therefore not limited to those presented in this thesis, but rather are open to multiple fields of applications.
Throughout this chapter, the following patterns emerged in the PAL-/DEL-AgentSpeak programs which warrant future work:

- **Mutually Exclusive Ranges**: In both the navigation and Minesweeper domains, we modelled the mutual exclusivity of the loc and state ranges. This presented with a pattern using multiple similar constraint rules. Given the prevalence of this requirement, there may be a need for syntactic shortcuts to succinctly capture this domain; for example, expressing a belief such as $\text{excl}(\text{loc}(X, Y))$ infers the mutual exclusivity of the loc range. As part of future work, we plan to examine other application domains and extract various syntactic shortcuts that can simplify the expression of uncertainty in PAL-/DEL-AgentSpeak.

- **Prioritization of Certainty**: A pattern emerges when expressing the prioritization of certainty over uncertainty in plan contexts, for example: $\text{dir}(\text{Dir}, \text{goal}) \lor \text{poss}(\text{dir}(\text{Dir}, \text{goal}))$ and $\neg \text{mine}(X, Y) \lor \text{poss}(\neg \text{mine}(X, Y))$. It should be expected that the agent wants to prioritize certainty over possibilities, as it provides a more robust foundation for reasoning and actions. The concept of prioritization also spans the selection of plans; our integration with PAL and DEL for uncertain reasoning in AgentSpeak warrants a mechanism for automatic plan preferences for those that express contexts with certainty. Plan selection based on preferences is a well-studied area [2], but it lies beyond the scope of this thesis. As part of future work, the integration of preferences with PAL-/DEL-AgentSpeak’s uncertainty reasoning would enable more robust and automatic decision making in uncertain situations.

In summary, this chapter demonstrated how PAL- and DEL-AgentSpeak are applied to a variety of uncertain domains and challenges, however, given the importance of
performance in the context of agent operation, we must extend this work to show how
the extensions perform and scale in practice. In the upcoming chapter, we will assess
the performance and scalability of PAL-AgentSpeak and DEL-AgentSpeak. We utilize
the 2019 MAPC to provide context to the feasibility of the extensions.
Chapter 6

Evaluation

6.1 Introduction and Structure

In this chapter, we evaluate the performance and scalability of PAL-AgentSpeak and DEL-AgentSpeak. Sections 6.2, 6.3, and 6.4 correspond to the evaluation of the model creation, updating, and querying processes, respectively.

The evaluation methodologies in this chapter utilize variations of the map from the running navigation example from Chapter 3; this allows for the scaling of uncertainty. The purpose of our evaluation is to confirm the worst-case computational complexities stated in Chapter 4, and to examine the overall impact on the agent’s reasoning cycle. Although we specifically use the navigation example in our evaluation, the presented evaluation methodology produces results such that the trends can be interpreted with respect to any domain, including the identification and Minesweeper applications seen in Chapter 5.
6.1.1 Introductory Remarks

PAL- and DEL-AgentSpeak are novel contributions to the field of autonomous agents, and as such, are the first extensions to integrate qualitative uncertainty reasoning with an agent-oriented language such as AgentSpeak. As of the current state of the literature, there are no agent-oriented languages nor language extensions that allow for symbolic uncertainty reasoning available as a benchmark for comparison. Instead, we provide a brief qualitative comparison with the standard language of AgentSpeak.

As we introduced in Chapter 3, PAL- and DEL-AgentSpeak provide declarative syntax that enable uncertainty reasoning. Standard AgentSpeak does not provide any mechanisms for uncertainty reasoning, and thus fails to allow the developer to sufficiently capture any amount of uncertainty.

Without a mechanism for uncertainty, it becomes difficult for the agent to make an informed decision. This is especially true for an agent that operates within the MAPC domain. In this case, an agent that does not have the ability to reason about uncertainty to localize itself must move randomly unless it becomes certain of its location. This trivial method of navigation is both inefficient and ineffective as the agent has no predictable way to get to its goal. Instead, developers that require uncertainty reasoning end up developing ad hoc solutions, as witnessed by the many ad hoc approaches taken by other developers in the 2019 MAPC [55].

Ad hoc approaches to uncertainty are imperative in nature as the developers must explicitly manage their uncertainty, rather than integrating with a reasoner which provides uncertainty reasoning. In general, this can be detrimental to the readability and maintainability of the code. With integrations such as PAL- and DEL-AgentSpeak, ad hoc approaches are no longer needed, and the declarative nature of these programs avoid issues that reduce the overall maintainability of agent code.
In the following sections of this chapter, we detail the evaluation methodology, the evaluation results, and provide a brief discussion and interpretation of the results in general. All experiments provided in this chapter are run with the following specifications.

- CPU: Intel i7-8700K
- RAM: 48GB
- OS: Windows 11
- JRE/JDK: 19

The final section of the chapter will use the parameters and time deadlines of the 2019 MAPC to provide feasibility remarks about the use of each operation in practice. Before we dive into the evaluation of performance and scalability, we provide a few prerequisite remarks about our evaluation.

### 6.2 Model Creation

Model creation involves creating an initial epistemic model from the initial beliefs, ranges, and constraints of the agent. This section evaluates the performance and scalability of the model creation implementation by examining how the model creation time varies with respect to the intended size of the initial model described via ranges and constraints.

We will now present the evaluation methodology and results.
6.2.1 Evaluation Methodology

We evaluate the model creation implementation provided in Section 4.2.3 based on the following parameters and metrics.

6.2.1.1 Parameters

We use the size of the initial epistemic model, created by the model creation process, to quantify the uncertainty described by the agent’s ranges and constraints. We evaluate the performance of PAL- and DEL-AgentSpeak by scaling the number of uncertain locations in the running navigation example from Chapter 3. The parameter used to scale and evaluate model creation is provided as follows:

- **Map Size ($|W|$)**: this parameter represents the size of the navigation map and corresponds to the number of worlds in the initial epistemic model $|W|$.

The map size serves as an effective evaluation parameter, since it directly affects the number of literals in the range $\text{loc}$ and constraints that require rewriting consequences, propositionalization, and SAT solving. In the evaluation of model creation, we vary the map size from 5x5 to 50x50.

6.2.1.2 Metrics

The following metrics are used to evaluate the performance of PAL- and DEL-AgentSpeak.

- **Model Creation Time (ms)**: the amount of time it takes for the initial model to be created from the initial beliefs/ranges/constraints.

- **Model Creation Space (MB)**: the maximum amount of space to create the initial epistemic model.
6.2.1.3 Methodology

Using the parameters and metrics provided above, we evaluate the scalability of the model creation process. The evaluation methodology is provided below.

1. Setup (given $|W|$)
   
   (a) Create a map of size $|W|$.
   
   (b) Update the navigation program where $\text{loc}(X, Y)$ ranges and constraints reflect the scaled map.

2. Start the DEL-AgentSpeak (or PAL-AgentSpeak) program and the timer for Model Creation Time. Monitor the maximum memory usage for Model Creation Space
   
   • Run the model creation process.

3. Record results for Model Creation Time and Model Creation Space.

The results for the evaluation of model creation will now be presented and discussed.

6.2.2 Model Creation Results

Recall from Section 4.2 that the time and space complexity of model creation is exponential with respect to the number of rule-based ranges and constraints due to the worst-case complexities for obtaining logical consequences and SAT solving. In practice, this depends on the nature of the ranges and constraints described by the program; in the case of the MAPC domain, the relationship between $|W|$ and Model Creation Time (ms) and Space (MB) for both PAL- and DEL-AgentSpeak are shown in Figure 6.1. Experiment data are provided in Table B.2 of Appendix B.
First, we point out that PAL-AgentSpeak and DEL-AgentSpeak have the same performance and scalability during the model creation phase. Given that the model creation process is highly dependent on the description of the agent program, it is difficult to use these results to confirm the exponential time and space complexity associated with the operation. However, it is evident that the model creation process grows quickly with respect to the number of worlds that need to be created.

In our evaluation, we see that model creation takes up to 300 seconds and 10 GB of memory to generate the initial epistemic model for the 50x50 navigation map. Note that 10GB is the space requirement for the model creation operation (SAT solving...
the logical consequences of the agent’s ranges and constraints); once generated, the compact initial epistemic model itself only requires 4 MB of memory (Section 4.1.3).

Given that the initial beliefs, ranges, and constraints are available at compile time, the model creation operation can be completed offline and cached. From an architectural perspective, SAT-solving during the model creation phase can also be delegated to an external high-performance machine and does not have to be performed on the same machine as the agent. These approaches ensure that the model creation process does not directly impact the agent’s time-sensitive reasoning cycle.

In the next section, we evaluate the performance and scalability of the model update operation.

6.3 Model Updating

In this section, we evaluate the performance and scalability of the model update operation. First, we introduce the evaluation methodology, including the parameters and metrics used in the evaluation, followed by a brief presentation and discussion of the results obtained.

6.3.1 Evaluation Methodology

In this experiment, we examine the impact that the epistemic model and the size of the event model have on the performance of the model update. This corresponds to the implementation provided in Section 4.2.3.
6.3.1.1 Parameters

There are two parameters of interest listed as follows; we utilize the initial epistemic models created during the experiment presented in Section 6.2.

• **Model Size** ($|W|$): the number of worlds $|W|$ in the pre-update epistemic model.

• **Event Size** ($|E|$): the number of events $|E|$ in the applied event model.

As in the evaluation of the model creation operation, $|W|$ will be scaled based on various sizes of the navigation map. We arbitrarily select large values for $|E| = 1, 50, 100$. During evaluation we fabricate $|E|$ applicable “on” plans with a pre-condition of $\top$ to ensure evaluation of the worst-case scenario (i.e., a non-eliminative event). In Section 6.5 we comment on how these arbitrarily-created events apply in practice, given the context of the MAPC.

6.3.1.2 Metrics

The metrics used to evaluate the performance of the model update operation are listed below.

• **Update Time (ms)**: the time to execute the model update process.

• **Update Space (MB)**: the maximum space required to perform the model update operation and to persist the resulting model.

6.3.1.3 Methodology

We measure the impact of the model size ($|W|$) and the size of the events ($|E|$) on the time and space required for model updates. We perform the following steps for each pair of values $|W|$ and $|E|$.
1. Setup (given $|W|$): use the existing initial epistemic models generated for $|W|$ in Section 6.2.

2. We start the timer for the Update Time metric and monitor the memory usage for Update Space.

3. We update the epistemic model via “on” plans that describe a DEL event model of size $|E|$.

4. Record the results for Update Time and Update Space, given $|W|$ and $|E|$.

We will now present and discuss the results obtained from the experiments.

### 6.3.2 Model Updating Results

Recall from Section 4.2 that the time and space complexity for the model update operations will scale linearly with respect to the current number of worlds and the size of the event model: $|W| \times |E|$. In the case of PAL-AgentSpeak, public announcements are equivalent to events with $|E| = 1$. Figure 6.2 presents the performance of the model update operation with respect to parameters $|W|$ and $|E|$. The experiment data are provided in Table B.3 of Appendix B. The results obtained by the experiment confirm the suspected update complexity of $|W| \times |E|$ for DEL-AgentSpeak, and $|W|$ for PAL-AgentSpeak.

This concludes the evaluation of model updates in PAL- and DEL-AgentSpeak. In the next section, we evaluate the performance and scalability of the model querying operation.
6.4 Model Querying

In this section, we evaluate the performance and scalability of the model querying implementation. The results presented in this section are applicable to both PAL-AgentSpeak and DEL-AgentSpeak. The evaluation methodology will now be presented.
6.4.1 Evaluation Methodology

The parameters, metrics, and methodology presented in this section measure the implementation of model querying for both PAL- and DEL-AgentSpeak from Section 4.2.4.

6.4.1.1 Parameters

The following parameters directly impact the model querying process:

- **Model Size** ($|W|$): the number of worlds in the epistemic model
- **# of Queries** ($|Q|$): the number of ground queries being evaluated by the extension.

As in the previous sections, $|W|$ will be scaled based on various sizes of the navigation map (5x5 to 50x50). The number of queries ($|Q|$) will be scaled based on the large arbitrarily chosen values of 50, 100, and 500. During the experiment, we fabricate $|Q|$ belief queries such that they will be entailed by the corresponding epistemic model; this ensures evaluation of the worst-case scenario (i.e., all worlds are evaluated). This will be interpreted within the context of the MAPC in Section 6.5.

6.4.1.2 Metrics

The following metric is used to evaluate the implementation of the model query operation.

- **Query Time** (ms): the time it takes to perform model querying.

Model querying is a strictly computational task; thus, metrics that measure space requirements are not required.
6.4.1.3 Methodology

In order to evaluate the model querying process, we look at the impact of the number of worlds in the current model and the number of queries on the total model querying time. We perform the following steps for each value of $|W|$ and $|Q|$ provided.

1. Setup: Initialize the DEL-AgentSpeak navigation program with $|W|$ locations/worlds.

2. Start the timer for Query Time.
   
   (a) Query $|Q|$ true belief formulae. The chosen belief formulae must be entailed, as this evaluates all worlds in the model (the worst-case time).

3. Stop evaluation, record values for Query Time.

The scalability results obtained for model querying will now be presented.

6.4.2 Model Querying Results

Figure 6.3 shows the impact that the number of queries $|Q|$ and the size of the epistemic model $|W|$ have on Query Time. $|Q|$ takes on values of 50, 100, 500, reflected by separate lines on the graph. The experiment data are provided in Table B.4 of Appendix B.

These results confirm the suspected model querying time complexity presented in Section 4.2, where time is linear with respect to the number of worlds and queries: $|W| \times |Q|$. Model querying operations utilize constant space and are thus not evaluated with respect to memory.

In the next section, we conclude the chapter with a brief overview of the performance and scalability of model creation, update, and querying. Using the parameters from
the 2019 MAPC as context, we discuss the feasibility of using PAL-AgentSpeak and DEL-AgentSpeak in practice.

### 6.5 Concluding Remarks: Feasibility

In this section, we provide some general conclusions on the performance and scalability of PAL- and DEL-AgentSpeak for model creation, updates, and queries. To provide context for the feasibility of the proposed extensions, we refer to the actual map size and perception parameters used by the 2019 MAPC.

- Map size: 50x50
- Perception range: 5 cell radius
  - 61 obstacle perception updates
- Deliberation time: \(\leq 4\) seconds

Using the results presented throughout this section, we comment on the feasibility of DEL-AgentSpeak in practice. Recall that the model creation and querying operations
perform the same across PAL- and DEL-AgentSpeak due to similar operational semantics, thus the following creation and querying feasibility interpretations apply to both extensions despite the MAPC only being modelled with DEL-AgentSpeak.

**Model Creation** Model creation is the most time- and space-consuming operation. To create an epistemic model that captures the uncertainty of a 50x50 map, the model creation operation takes just under 300 seconds and around 10GB of memory. Recall that this time and space are required by the functions used to create the resulting model (i.e., obtaining range/constraint consequences and SAT solving). Once generated, the resulting compact model only requires 4MB to be stored in memory. This operation can be done offline, and the compact model can be cached so that the operation has no direct impact on the agent’s reasoning cycle.

**Model Updating** Model updates are performed any time the agent receives new perceptions or modifies its beliefs. In the case of the 2019 MAPC, the agent receives 61 obstacle perceptions in each reasoning cycle. We estimate that 61 model updates will take around 1.9 seconds and around 200MB of memory. The agent may also perceive moved(\textit{Dir}) which takes 43 milliseconds and 50MB. The total time and space required for model updates in the 2019 MAPC are 1.943 seconds and a maximum of 200MB, respectively.

Due to its restrictive update semantics, PAL-AgentSpeak is unable to model the level of expressiveness required by the MAPC. However, in general, PAL-AgentSpeak performs the exact same as DEL-AgentSpeak when $|E| = 1$. Thus, monotonic belief updates with PAL-AgentSpeak perform fairly efficiently, where each execute in 43ms with 50MB of memory given a large model of uncertainty ($|W| = 2500$, $|E| = 1$).
Model Queries  The time it takes to perform a single model query is linearly correlated with the number of worlds and formulas evaluated. In the case of the 2019 MAPC, we evaluate the certainty or possibility of four navigation directions (←, →, ↑, ↓). During the evaluation, these literals are rewritten as the corresponding loc literals. Given that there are at most 2500 (50x50) locations (|W| = 2500 and |Q| = 2500), it is estimated that the dir queries will take at most 223 * 5 = 1115 milliseconds (based on a model querying time of 223 for |W| = 2500 and |Q| = 500).

In the worst-case scenario, the agent requires 1.115 + 1.943 = 3.058 seconds per reasoning cycle to perform model updates and queries. As the agent reduces its uncertainty, the computation time and memory requirements are reduced. However, even in the worst-case scenario, the agent is able to deliberate within the MAPC’s imposed time constraint of 4 seconds. We conclude the chapter with some final remarks.

6.5.1 Final Remarks

In this chapter, we evaluated the performance and scalability of the implementations for PAL- and DEL-AgentSpeak. The reader must keep in mind that these evaluation results are provided for the implementation presented in Chapter 4, where compact epistemic and event models are used. These results will vary if the underlying reasoner or model representation is changed. For example, if one uses implicit models instead of compact models for the sake of providing a space-efficient model representation (see Chapter 4), model updates and queries may become exponential in time in the worst-case scenario [47].

We conclude with the following final remarks.
• **Model Creation:** In Chapter 4, we analyzed model creation to be exponential in the worst-case scenario. Although our evaluation did not test for the worst-case scenario, the creation process required a large amount of computation time and memory to generate the model. Across all operations, model creation was the least performant; however, its impact on the time-sensitive reasoning cycle is negligible, since the operation can be performed offline and cached.

• **Model Updates and Queries:** In our evaluation, we confirm that model updates and queries are tractable operations that bring linear time and space (updates only) with respect to the magnitude of uncertainty. The model updating and querying operations occur frequently during agent execution, and their efficient operation provides an optimal integration with the reasoning cycle. These results are made possible by the compact model representations, whose implementations were presented in Chapter 4.

• **Generalizability:** The evaluation methodologies were designed to measure the worst-case scenarios for update and query operations. This allows our impact on the reasoning cycle to be interpreted within the context of agents in other uncertainty domains and is not limited to the 2019 MAPC.

• **Feasibility:** Overall, the cost of being able to express and reason about uncertainty within the reasoning cycle is available through computationally efficient and practical means. The results obtained throughout this chapter were interpreted within the context of the 2019 MAPC; we found that PAL- and DEL-AgentSpeak both constituted feasible approaches to reasoning about uncertainty in the 2019 MAPC. Uncertainty has the potential to grow much larger than the scale seen in the 2019 MAPC, although, we are confident that the
efficient model update and query operations provided by the extensions will
allow them to be used with more complex domains. Future work involves the
integration of PAL-/DEL-AgentSpeak with other applications (e.g., using DEL
to model knowledge in autonomous robotics [57]) and the evaluation of its ability
to reason promptly within these domains.

In the next chapter, we conclude the thesis with an overview of the presented
chapters, the contributions made, and to propose future work for the extensions.
Chapter 7

Conclusion

As we explored earlier in the thesis, AgentSpeak is a popular agent-oriented programming language, but one major limitation is the lack of native support for modelling and reasoning about belief uncertainty. In the literature, there is a lack of AgentSpeak extensions for qualitative uncertainty, despite the clear need for such an integration in a symbolic language such as AgentSpeak.

This thesis proposed two novel contributions, PAL-AgentSpeak and DEL-AgentSpeak, which extend AgentSpeak with epistemic logic and the respective dynamic forms PAL and DEL. The logics and their corresponding AgentSpeak extensions provide varying levels of expressiveness for modelling changes in uncertainty. The extensions are formalized, implemented, applied to various domains of uncertainty, and evaluated on their performance, scalability, and feasibility in practice.

The following sections will summarize the main contributions of this thesis, discuss the implications of these extensions on the broader field of AI, and recap the limitations and potential avenues for future research.
7.1 Summary and Contributions

In this section, we provide an itemized overview of the main contributions made by this thesis.

- **State of the Art:** This thesis motivated the need for uncertainty reasoning in AgentSpeak and extensively explored the relevant literature with respect to existing extensions. We found that quantitative approaches constitute the entirety of implemented extensions for uncertainty, but these approaches may not always be available or feasible and do not integrate well with symbolic languages such as AgentSpeak. In contrast, qualitative approaches to uncertainty exist strictly in theory and fail to be implemented due to the use of logical dialects that bring a high computational complexity. Thus, this thesis demonstrates the need for a computationally feasible implementation for qualitative uncertainty reasoning in AgentSpeak.

- **Methodological Contributions:** This thesis proposes the usage of epistemic logic, via its dynamic forms PAL and DEL, to extend AgentSpeak in an expressive and computationally tractable way. PAL-AgentSpeak and DEL-AgentSpeak were formally presented in Chapter 3. These extensions integrate a PAL/DEL reasoner with AgentSpeak and involve epistemic model creation, updating, and querying, respectively using defined ranges and constraints, belief addition and deletion events, and queries that enable possibilistic reasoning (via \texttt{poss}). The extensions differ mainly in their ability to apply updates.

- **Extensions for Uncertainty:** PAL-AgentSpeak is a simpler extension which only allows for simple monotonic belief change. As we saw with the Minesweeper
agent in Chapter 5, if an agent is restricted to simple belief changes, PAL-AgentSpeak is preferred because it doesn’t require additional input from the developer, and its restricted semantics ensure that the complexity of uncertainty remains limited throughout execution. DEL-AgentSpeak, on the other hand, provides more advanced and expressive constructs that allow for non-monotonic changes and event uncertainty through “on” plans, but comes at an additional cost due to its allowance for the growth of uncertainty over time. It is up to the agent’s developer to choose an extension based on these trade-offs.

- **Implementation Contributions:** We also discussed the implementation of PAL- and DEL-AgentSpeak in Chapter 4. This chapter detailed all of the crucial components required to integrate the PAL/DEL reasoning engine, Hintikka’s World, with Jason, a popular AgentSpeak development platform. As part of our contributions, we also extended Hintikka’s World with an optimized and compact implementation of S5 epistemic and event models, and extended the reasoner with a REST API. The impact of these extensions surpasses our presented AgentSpeak extensions, as they allow other users of Hintikka’s World to leverage our optimized model implementation and API endpoints.

- **Application Domains:** We applied DEL-AgentSpeak to two 2019 MAPC challenges: navigation and identification. The navigation application showed how DEL-AgentSpeak manages and reasons about uncertainty while simultaneously localizing and navigating the map. On the surface, these applications are about localizing and navigating the agent, but our presented solution represents a much more general problem: being able to make rational decisions despite being
faced with uncertainty. Through simply modelling our possibilities, we were able to make better decisions about how to navigate the map.

In the identification application, DEL-AgentSpeak provides a succinct way to manage uncertainty about a perceived agent’s identity; this application also illustrated how declarative constructs like “on” plans can be used to handle uncertainty without interfering with the behavioural intent of the AgentSpeak program. Finally, we applied the simpler extension PAL-AgentSpeak to Minesweeper, a domain restricted to monotonic belief changes, resulting in a more concise description of the domain in comparison to DEL-AgentSpeak. Lastly, we identified patterns which emerged in the development of these applications as a result of our new-found ability to express uncertainty.

We also briefly discussed the general applicability of PAL- and DEL-AgentSpeak. PAL- and DEL-AgentSpeak are not limited to the MAPC, however, the MAPC provides an appropriate representation of the class of uncertainty typically faced by AgentSpeak agents and therefore demonstrates that our extensions have use in a variety of other uncertainty domains as well. Additionally, given the integration with PAL and DEL, we inherently allow AgentSpeak programs that can now reason about domains where PAL and DEL are best suited (e.g., security [56] and robotics [57]).

- **Performance Evaluation:** This thesis evaluated PAL- and DEL-AgentSpeak through experiments which evaluate the worst-case performance, scalability, and feasibility, per the magnitude of uncertainty required by the domain. Model creation was undoubtedly the most time- and space-consuming operation due to its exponential nature. The silver lining is that it can be performed offline.
and there are techniques (e.g. caching) that eliminate the burden this process places on the agent’s reasoning cycle. Model updates and queries both occur frequently throughout the agent’s reasoning cycle; the experiments found that these operations are tractable and efficient in terms of computational and memory requirements, and pose a plausible integration with AgentSpeak as a result. Lastly, the 2019 MAPC was used to provide context for the feasibility of PAL-/DEL-AgentSpeak. Through the experimental results data, we find that the agent efficiently performs the required updates and queries within the reasoning cycle deadlines imposed by the MAPC. The trends obtained in our evaluation confirm the worst-case computational complexities for our presented implementations, and are thus applicable to other domains outside of the MAPC and Minesweeper. We commented on the limitations of our evaluation, and suggest lines of future work to extend the applicability of these results.

In the following section, we discuss the implications and limitations of our findings and provide various potential avenues for future research.

### 7.2 Final Remarks: Limitations and Future Work

The contributions presented in this thesis have significant implications for the broader field of AI, particularly in the context of agent-oriented programming languages and their ability to symbolically express and reason about uncertainty. The integration of PAL and DEL into AgentSpeak enables idiomatic reasoning and decision-making abilities when faced with uncertainty, improving the applicability of AgentSpeak to various diverse domains.
• **Integration of Fields**  This thesis bridges the gap between theory and practice through the integration of PAL / DEL and AgentSpeak, allowing theoretical techniques that only exist in the literature to be realized. For example, DEL-based planning [54] is a community dedicated to modelling and reasoning about uncertainty in planning. DEL-AgentSpeak provides the mechanisms for expressing and implementing planning policies that allow the future development of BDI-based agents that follow DEL-based goals and policies.

• **Large State Space**  Despite our use of compact epistemic models, the exponential space requirement presents a major challenge that hinders the scalability and generalizability of our approach for larger real-world scenarios. Compact models are *explicit* representations of uncertainty, meaning that each world is explicitly modelled. Future research involves the exploration of *implicit models* which represent uncertainty in a space-efficient manner using formulae rather than explicit worlds, at the cost of a worse time complexity, since they require SAT-solving for all model operations [47]. Although this trade-off might not be ideal for time-sensitive reasoning cycles, it could become necessary when space requirements become impractical. At the moment, Hintikka’s World only has partial support for DEL using implicit model representations [15].

• **Multi-Agent Reasoning**  This thesis focused on the uncertainty reasoning of a single agent to limit the scope and computational complexity [47]. However, both DEL and AgentSpeak possess constructs to model multi-agent systems [38]. Future work involves integrating multi-agent DEL for uncertainty reasoning
in multi-agent AgentSpeak scenarios. Furthermore, it is essential to examine performance and scalability to ensure efficient and feasible integration.

- **Plan Preferences** In AgentSpeak, the selection of plans for execution based on preferences is a well-known field of research [2]. Throughout the thesis, it is evident that agents prefer certainty over possibilities. This paves the way for further investigation into mechanisms for plan preferences that automatically prioritize plans based on their level of certainty; allowing for more robust decision-making in uncertain environments.
Bibliography


Representing, Processing, and Learning Preferences: Theoretical and Practical Challenges.


Appendix A

Application Listings

A.1 DEL-AgentSpeak Navigation Program

A.1.1 Navigation Environment

Algorithm 1 presents the pseudocode for obtaining perceptions from the map environment.

Algorithm 1 Obtaining Navigation perceptions.

```latex
\begin{algorithm}
\SetKwProg{Fn}{function}{}{end}
\Fn{perceive(map, loc)}{
    \textit{P} ← \{\}\;
    \For{dir ∈ \leftarrow, \rightarrow, \uparrow, \downarrow}{
        \If{map has \texttt{obs(dir)} at \textit{loc}}{
            \textit{P} ← \textit{P} \cup \{\texttt{obs(dir)}\}
        }\Else{
            \textit{P} ← \textit{P} \cup \{\sim\texttt{obs(dir)}\}
        }
    }
    \If{map has Obj ∈ \{goal, disp\} at \textit{loc}}{
        \textit{P} ← \textit{P} \cup \{\texttt{at(Obj)}\}
    }
    \Return \textit{P} \Comment{Provide perceptions to agent}
}\end{algorithm}
```
### A.1.2 Navigation Listing

Listing A.1 presents the complete DEL-AgentSpeak program for the navigation agent from Chapter 3. This listing is trivially adapted to the navigation and identification agents from Chapter 5.

```prolog
1  range(loc(X,Y)) :- (. range(X,0,4) & . range(Y,0,4)).
2  ~loc(X1, Y1) :- loc(X1, Y1) & loc(X2, Y2) & (X1 \== X2 | Y1 \== Y2).
3  ~loc(1,2).
4  ~loc(2,2).
5  /* Remove case of all locations being false */
6  range(loc(none)) :- true.
7  loc(none) :- ~loc(0,0) & ... & ~loc(4,4).
8  ~loc(none).
9  // Obstacle mappings
10  obs(down) :- loc(1,1) | loc(2,1).
11  obs(left) :- loc(0,2).
12  obs(up) :- loc(1,3) | loc(2,3).
13  obs(right) :- loc(3,2).
14  // No obstacles otherwise (closed-world)
15  ~obs(D) :- not(obs(D)).
16  // Direction mappings for goal
17  dir(down, goal) :- loc(0,0) | loc(0,1) | loc(0,2) | loc(1,0) | loc(2,0) | loc(3,0) | loc(3,1) | loc(3,2) | loc(4,0) | loc(4,1) | loc(4,2).
18  dir(right, goal) :- loc(0,3) | loc(0,4) | loc(1,0) | loc(1,1) | loc(1,3) | loc(1,4) | loc(2,0) | loc(2,1).
19  dir(left, goal) :- loc(1,0) | loc(1,1) | loc(3,3) | loc(3,4) | loc(4,0) | loc(4,1) | loc(4,2) | loc(4,3) | loc(4,4).
20  dir(up, goal) :- loc(0,4) | loc(1,4) | loc(2,4) | loc(3,4) | loc(4,4).
```
// Direction mappings for disp

```prolog
dir(down, disp) :- loc(0,0) | loc(0,1) | loc(0,2) | loc(1,0) | loc(4,0) | loc(4,1) | loc(4,2) | loc(4,3) | loc(3,0) | loc(3,1) | loc(3,2) | loc(2,0).
dir(right, disp) :- loc(0,3) | loc(0,4) | loc(1,0) | loc(1,1) | loc(1,3) | loc(1,4) | loc(2,0) | loc(2,1) | loc(3,4).
dir(up, disp) :- loc(1,4) | loc(2,4) | loc(0,4).
dir(left, disp) :- loc(1,0) | loc(3,3) | loc(1,1).

// No directions otherwise (closed-world)
~dir(D, Obj) :- not(dir(D, Obj)).

// "On" plans for obstacles
+on(obs(Dir)) : obs(Dir).
+on(~obs(Dir)) : ~obs(Dir).

// "On" plans for movement
+on(moved(right)) : loc(X, Y) & X < 4
  <= -loc(X, Y);
  +loc(X + 1, Y).
+on(moved(left)) : loc(X, Y) & X > 0
  <= -loc(X, Y);
  +loc(X - 1, Y).
+on(moved(up)) : loc(X, Y) & Y > 0
  <= -loc(X, Y);
  +loc(X, Y - 1).
+on(moved(down)) : loc(X, Y) & Y < 4
  <= -loc(X, Y);
  +loc(X, Y + 1).
```

// Navigation Plans
!nav(goal).
!nav(Obj) : at(Obj)
A.2 PAL-AgentSpeak Minesweeper Program

This section presents the code for the Minesweeper program.

A.2.1 Navigation Environment

Algorithm 2 presents the pseudocode for obtaining perceptions from the Minesweeper environment given the actual state \textit{state} and set of clicked locations \textit{clicked}.

\textbf{Algorithm 2} Obtaining Minesweeper perceptions.

\begin{verbatim}
function perceive(state, clicked)
    P ← {} \\
    for \((x, y) \in \text{clicked} \text{ where } \text{hint}(x, y, N) \text{ in state} \text{ do}
        P ← P ∪ \{\text{hint}(x, y, N)\}
    if \text{all free spaces clicked in state} \text{ then}
        P ← P ∪ \{\text{status(win)}\}
    else if \text{mine clicked in state} \text{ then}
        P ← P ∪ \{\text{status(lose)}\}
    return P ▷ Provide perceptions to agent
\end{verbatim}

A.2.2 Listing

Listing A.2 presents the full PAL-AgentSpeak program for the Minesweeper agent.

\begin{verbatim}
// Range and constraints: state(S)
range(state(S)) :- .member(S, [1, 2, 3, 4]).
\end{verbatim}
\[ \neg \text{state}(S) :\neg \text{state}(S2) \land S \equiv S2. \]

/* Remove case of all states being false */

\[ \text{range}(\text{state}(\text{none})) :\neg \text{true}. \]

\[ \text{state}(\text{none}) :\neg \text{state}(1) \land \ldots \land \neg \text{state}(4). \]

\[ \neg \text{state}(\text{none}). \]

// Rules for hints

\[ \text{hint}(X,Y,N) :\text{state}(1) \land \text{member}([X,Y,N], [[1,1,2], [3,1,1], [1,3,1], \ldots]). \]

\[ \text{hint}(X,Y,N) :\text{state}(2) \land \text{member}([X,Y,N], [[1,1,1], [3,1,1], [1,2,1], \ldots]). \]

\[ \text{hint}(X,Y,N) :\text{state}(3) \land \text{member}([X,Y,N], [[1,1,1], [2,1,1], [1,3,1], \ldots]). \]

\[ \text{hint}(X,Y,N) :\text{state}(4) \land \text{member}([X,Y,N], [[1,1,0], [2,1,1], [1,2,1], \ldots]). \]

\[ \neg \text{hint}(X,Y,N) :\neg \text{not(\text{hint}(X,Y,N))}. \]

// Rules for mines

\[ \text{mine}(2,1) :\text{state}(1) \lor \text{state}(2). \]

\[ \text{mine}(1,2) :\text{state}(1) \lor \text{state}(3). \]

\[ \text{mine}(1,3) :\text{state}(2) \lor \text{state}(4). \]

\[ \text{mine}(3,1) :\text{state}(3) \lor \text{state}(4). \]

\[ \neg \text{mine}(X,Y) :\neg \text{not(\text{mine}(X,Y))}. \]

// Plans for playing minesweeper

\[ !\text{play}. \]

\[ *!\text{play} : \text{status}(\text{Status}) \]

\[ \quad \leftarrow \text{.print("The game has ended with: ", \text{Status}).} \]

\[ *!\text{play} : \neg\text{mine}(X,Y) \lor \text{poss}(\neg\text{mine}(X,Y)) \]

\[ \quad \leftarrow \text{+click}(X,Y); \]

\[ !\text{play}. \]

Listing A.2: Full PAL-AgentSpeak Minesweeper Program
Appendix B

Evaluation Result Tables

Table B.1 presents the time and space requirements for creating $|W|$ worlds in an epistemic model using the original and compact representations in Hintikka’s World.

B.1 Compact Model Performance

Table B.1: Comparison between original and compact model representations, given $|W|$ worlds.

| $|W|$ | Total Time (Original, ms) | Total Time (Compact, ms) | MAX Memory (Original, MB) | MAX Memory (Compact, MB) |
|------|---------------------------|--------------------------|---------------------------|--------------------------|
| 100  | 2                         | 1                        | 1                         | 1                        |
| 200  | 6                         | 1                        | 2                         | 1                        |
| 500  | 18                        | 1                        | 11                        | 1                        |
| 1000 | 70                        | 1                        | 22                        | 1                        |
| 2000 | 243                       | 1                        | 97                        | 1                        |
| 5000 | 1698                      | 3                        | 954                       | 3                        |
| 10000| 10166                     | 22                       | 3007                      | 4                        |
B.2 Model Creation Results

Table B.2 presents the time (ms) and space (MB) required for the model creation process for both PAL-AgentSpeak and DEL-AgentSpeak.

Table B.2: Model creation results.

| Map Size | |W| | Total Time (DEL, ms) | Total Time (PAL, ms) | MAX Memory (DEL, MB) | MAX Memory (PAL, MB) |
|----------|----------|----------|-----------------|-----------------|-----------------|-----------------|
| 5x5      | 25       | 52       | 80              | 15              | 17              |
| 10x10    | 100      | 295      | 422             | 26              | 35              |
| 20x20    | 400      | 2,951    | 2,595           | 382             | 310             |
| 25x25    | 625      | 6,514    | 6,930           | 753             | 721             |
| 30x30    | 900      | 22,323   | 17,693          | 1,504           | 1,530           |
| 35x35    | 1,225    | 78,851   | 79,112          | 2,107           | 2,331           |
| 40x40    | 1,600    | 127,779  | 141,218         | 3,603           | 3,878           |
| 45x45    | 2,025    | 216,121  | 218,214         | 6,489           | 6,757           |
| 50x50    | 2,500    | 304,462  | 295,209         | 9,821           | 9,636           |

B.3 Model Update Results

Table B.3 shows the results for model updating.

B.4 Model Querying

Table B.4 presents the time required to evaluate 50, 100, and 500 queries on an epistemic model with |W| worlds.
Table B.3: Model update results.

| Map Size | $|W|$ | $|E|$ | Total Time (ms) | MAX Memory (MB) |
|----------|-----|-----|-----------------|-----------------|
| 5x5      | 25  | 1   | 1               | 13              |
| 10x10    | 100 | 1   | 5               | 12              |
| 20x20    | 400 | 1   | 6               | 14              |
| 30x30    | 900 | 1   | 13              | 14              |
| 40x40    | 1600| 1   | 18              | 20              |
| 50x50    | 2500| 1   | 43              | 50              |
| 5x5      | 25  | 50  | 23              | 15              |
| 10x10    | 100 | 50  | 74              | 25              |
| 20x20    | 400 | 50  | 140             | 40              |
| 30x30    | 900 | 50  | 353             | 52              |
| 40x40    | 1600| 50  | 643             | 100             |
| 50x50    | 2500| 50  | 1268            | 201             |
| 5x5      | 25  | 100 | 40              | 17              |
| 10x10    | 100 | 100 | 100             | 34              |
| 20x20    | 400 | 100 | 326             | 53              |
| 30x30    | 900 | 100 | 757             | 82              |
| 40x40    | 1600| 100 | 1310            | 168             |
| 50x50    | 2500| 100 | 2879            | 287             |
Table B.4: Model query results.

| Map Size | $|W|$ | Queries | Total Time (ms) |
|----------|-----|---------|-----------------|
| 5x5      | 25  | 50      | 9               |
| 10x10    | 100 | 50      | 12              |
| 20x20    | 400 | 50      | 19              |
| 30x30    | 900 | 50      | 22              |
| 40x40    | 1600| 50      | 35              |
| 50x50    | 2500| 50      | 39              |
| 5x5      | 25  | 100     | 20              |
| 10x10    | 100 | 100     | 25              |
| 20x20    | 400 | 100     | 32              |
| 30x30    | 900 | 100     | 39              |
| 40x40    | 1600| 100     | 51              |
| 50x50    | 2500| 100     | 64              |
| 5x5      | 25  | 500     | 118             |
| 10x10    | 100 | 500     | 121             |
| 20x20    | 400 | 500     | 145             |
| 30x30    | 900 | 500     | 169             |
| 40x40    | 1600| 500     | 201             |
| 50x50    | 2500| 500     | 223             |