VISUALISING DYNAMIC EPISTEMIC LOGIC

By

Maduka Kingsley Attamah

A DISSERTATION

Submitted to

The University of Liverpool

in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE
IN ADVANCED COMPUTER SCIENCE

September 13, 2012
ABSTRACT

VISUALISING DYNAMIC EPISTEMIC LOGIC

By

Maduka Kingsley Attamah

In dynamic epistemic logic scenarios, information often results in the change of epistemic state of agents, hence a change in the model representing the scenario. In such dynamic scenarios, it is desirable to visualise the changes in the knowledge of the agents by automatically generating graphical models for both the initial scenarios and updated epistemic scenarios due to new information. In this project we develop dynamic epistemic modelling tool, vdel, which takes an epistemic description, and builds a graphical Kripke model for it. Subsequently, the system accepts public announcements, and updates the model accordingly. For the original and the updated model, the user can then submit epistemic logical formulas, and the system checks whether they are true or not in the designated state. Well-known epistemic scenarios such as the Muddy Children puzzle, Sum and Product puzzle, Prisoners and Hats game, Game of Hexa and the Consecutive Numbers Puzzle were used to evaluate the tool. We also compare vdel with the state-of-the-art dynamic epistemic modelling tool DEMO. This project introduces the vdel description language, vdel interpreter and vdel model checker, as well as an Epistemic Keypad – a graphical user interface to the vdel model checker.
DECLARATION

I hereby certify that this dissertation constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I declare that the dissertation describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

Maduka Kingsley Attamah
ACKNOWLEDGMENTS

I would like to acknowledge and thank those who have provided advice, guidance and support throughout this project:

Prof. Wiebe van der Hoek (Supervisor) and Dr. Davide Grossi (Second Supervisor) for their help and guidance throughout this project. Their positive and constructive feedback, suggestions and constant availability shaped the outcome of this project.

I would also like to thank family and friends who have tolerated the many hours I have dedicated purely to this project over the previous months.
# Table of Content

Abstract .......................................................................................................................... ii  
Declaration .................................................................................................................. iii  
Acknowledgments ....................................................................................................... iv  
Table of Content ......................................................................................................... v  
List of Tables ............................................................................................................... viii  
List of Figures ............................................................................................................. ix  
List of Algorithms ..................................................................................................... 1  
List of Code Listings ................................................................................................. 2  
Introduction ............................................................................................................... 3  
  1.1 Scope .................................................................................................................... 3  
  1.2 Problem Statement ............................................................................................ 3  
  1.3 Approach ............................................................................................................. 4  
  1.4 Outcome ............................................................................................................. 4  
  1.5 Outline ................................................................................................................ 5  
Background ............................................................................................................... 6  
  2.1 Epistemic Logic .................................................................................................. 6  
    2.1.1 Language and Semantics of Epistemic Logic ............................................ 6  
    2.1.2 Possible world Semantics for Knowledge and belief ............................. 7  
  2.2 Dynamic Epistemic Logic .................................................................................. 8  
    2.2.1 Language of Dynamic Epistemic Logic ................................................. 8  
    2.2.2 Semantics of Dynamic Epistemic Logic ............................................... 9  
  2.3 The Logic of Public Announcement ................................................................. 10  
  2.4 Epistemic Model Checking ............................................................................. 11  
Design ......................................................................................................................... 12  
  3.1 Design of the Description Language ................................................................. 12  
  3.2 Evaluation of the Description Language ......................................................... 12  
  3.3 Design of the Description Language (VDEL) Interpreter ............................. 15  
    3.3.1 Epistemic Description Parser ................................................................. 15
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.2 State Variable Generator</td>
<td>15</td>
</tr>
<tr>
<td>3.3.3 The Constraint Evaluator (CE)</td>
<td>16</td>
</tr>
<tr>
<td>3.3.4 The Epistemic Property Evaluator (EPE)</td>
<td>16</td>
</tr>
<tr>
<td>3.3.5 Kripke Model Generator</td>
<td>16</td>
</tr>
<tr>
<td>3.4 Model Checker</td>
<td>17</td>
</tr>
<tr>
<td>3.4.1 Algorithms for the Epistemic Operators</td>
<td>17</td>
</tr>
<tr>
<td>3.5 DOT Code Generator</td>
<td>24</td>
</tr>
<tr>
<td>3.6 Graph Drawing</td>
<td>24</td>
</tr>
<tr>
<td>3.7 The Epistemic Keypad</td>
<td>25</td>
</tr>
<tr>
<td>3.8 Language for Epistemic Properties</td>
<td>26</td>
</tr>
<tr>
<td>3.9 User Interaction</td>
<td>27</td>
</tr>
<tr>
<td>Realisation</td>
<td>28</td>
</tr>
<tr>
<td>4.1 The vdel Language</td>
<td>28</td>
</tr>
<tr>
<td>4.1.1 Atoms declaration</td>
<td>28</td>
</tr>
<tr>
<td>4.1.2 Agents Declaration</td>
<td>29</td>
</tr>
<tr>
<td>4.1.3 Constraints Declarations</td>
<td>29</td>
</tr>
<tr>
<td>4.2 The vdel Interpreter: Test and Evaluation</td>
<td>32</td>
</tr>
<tr>
<td>4.3 The Epistemic Keypad</td>
<td>38</td>
</tr>
<tr>
<td>4.4 The vdel Model Checker: Test and Evaluation</td>
<td>39</td>
</tr>
<tr>
<td>4.4.1 Case Study: Muddy Children Puzzle</td>
<td>41</td>
</tr>
<tr>
<td>4.4.2 Case Study: Consecutive Numbers Puzzle</td>
<td>46</td>
</tr>
<tr>
<td>4.4.3 Solving the Sum and Product Puzzle</td>
<td>52</td>
</tr>
<tr>
<td>Evaluation</td>
<td>54</td>
</tr>
<tr>
<td>5.1 DEMO vs. VDEL</td>
<td>54</td>
</tr>
<tr>
<td>5.1.1 General Functionality</td>
<td>54</td>
</tr>
<tr>
<td>5.1.2 Epistemic Description</td>
<td>54</td>
</tr>
<tr>
<td>5.1.3 Model Visualisation</td>
<td>55</td>
</tr>
<tr>
<td>5.1.4 Model Checking and Time Complexity</td>
<td>55</td>
</tr>
<tr>
<td>5.1.5 General implementation differences</td>
<td>56</td>
</tr>
<tr>
<td>5.2 Summary of Strengths, Weaknesses and Recommendations</td>
<td>56</td>
</tr>
<tr>
<td>Professional Issues</td>
<td>58</td>
</tr>
<tr>
<td>6.1 Code of Conduct</td>
<td>58</td>
</tr>
<tr>
<td>6.2 Code of Good Practice</td>
<td>59</td>
</tr>
</tbody>
</table>
Conclusions ............................................................................................................................................. 60
  7.1 Summary of Main Outcomes........................................................................................................ 60
  7.2 Future Work ................................................................................................................................ 60
Appendix A: BNF Grammar for vdel Description Language ............................................................ 63
Appendix B: BNF Grammar for the Arithmetic and Boolean Logic Evaluator (ABLE) ............ 65
Appendix C: BNF Grammar for the Epistemic Property Evaluator .................................................. 66
Appendix D: Project Plan ..................................................................................................................... 68
LIST OF TABLES

Table 1: Meaning of Symbols on the Epistemic Keypad ................................................................. 39
Table 2 : Model Checking 2-Atom-2-Agent (GLO) - Inputs and Outputs ........................................ 40
Table 3 : Model Checking Muddy Children Puzzle (Initial Scenario) - Inputs and Outputs .......... 42
Table 4: Model Checking Muddy Children Puzzle – After the First Announcement - Inputs and Outputs ......................................................................................................................... 44
Table 5 : Model Checking Muddy Children Puzzle – After the Second Announcement - Inputs and Outputs ......................................................................................................................... 45
Table 6 : Model Checking the Initial Scenario of the Consecutive Numbers Puzzle - Inputs and Outputs ......................................................................................................................... 47
LIST OF FIGURES

Figure 1: Overview of the DEL Visualiser .......................................................... 4
Figure 2: A Simple Sample Graphical Kripke Model ........................................... 7
Figure 3: High-Level Object Components of the Epistemic Description Interpreter ... 15
Figure 4: Entity-Relation Diagram of the Graph Data Structure .......................... 17
Figure 5: Sample Output from the DOT Tool .................................................... 25
Figure 6: A Design Sketch of the Epistemic Keypad ......................................... 26
Figure 7: Activity Diagram of the Software Tool (DEL Visualiser) ...................... 27
Figure 8: Generated Model for the Hello World vdel ........................................ 33
Figure 9: Generated Model for the 2-Atoms-2-Agent (GLO) Scenario .................. 33
Figure 10: Generated Model for the 2-Atom-2-Agents (GLO) Scenario with one Constraint .... 34
Figure 11: Generated Model for the 4-Atom-2-Agent (GLO) scenario .................. 35
Figure 12: Generated Model for the Game of Hexa ......................................... 36
Figure 13: Generated Model for the Prisoners and Hats Puzzle (5-hat variant, 3 hats ON) .... 37
Figure 14: Screen-shot of the Epistemic Keypad ............................................. 38
Figure 15: cm1 .......................................................................................... 40
Figure 16: cm2 .......................................................................................... 41
Figure 17: Generated Model of the Initial Situation in the Muddy Children Puzzle .... 42
Figure 18: cm3 .......................................................................................... 43
Figure 19: Generated Model the Muddy Children Puzzle after the First Announcement .... 43
Figure 20: cm4 .......................................................................................... 44
Figure 21: Generated Model the Muddy Children Puzzle after the Second Announcement ... 45
Figure 22: Generated Model of the Consecutive Numbers Puzzle ......................... 47
Figure 23: cm5 .......................................................................................... 49
Figure 24: Consecutive Numbers Puzzle after the First Announcement ................ 50
Figure 25: Consecutive Numbers Puzzle after the Second Announcement ............. 51
Figure 26: Consecutive Numbers Puzzle after the Third Announcement .............. 51
Figure 27: Unique Solution to the Sum and Product Puzzle ............................... 53
# List of Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm 1</td>
<td>First order K-Operation</td>
<td>18</td>
</tr>
<tr>
<td>Algorithm 2</td>
<td>Iterated K-Operation</td>
<td>19</td>
</tr>
<tr>
<td>Algorithm 3</td>
<td>K-Operations involving Conjunctions, Disjunctions, Implications and Negations</td>
<td>20</td>
</tr>
<tr>
<td>Algorithm 4</td>
<td>First Order General Knowledge (&quot;Everybody Knows&quot;) Operator</td>
<td>21</td>
</tr>
<tr>
<td>Algorithm 5</td>
<td>Iterated General Knowledge (&quot;Everybody Knows&quot;) Operator</td>
<td>21</td>
</tr>
<tr>
<td>Algorithm 6</td>
<td>Common Knowledge</td>
<td>22</td>
</tr>
<tr>
<td>Algorithm 7</td>
<td>Distributed Knowledge</td>
<td>23</td>
</tr>
<tr>
<td>Algorithm 8</td>
<td>Graph Update after an announcement</td>
<td>23</td>
</tr>
<tr>
<td>Algorithm 9</td>
<td>DOT Code Generation</td>
<td>24</td>
</tr>
</tbody>
</table>
# List of Code Listings

<table>
<thead>
<tr>
<th>Code Listing</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2-Atom-2-Agent (GLO) Scenario</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>Muddy Children Puzzle (initial scenario)</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Muddy Children Puzzle (after first announcement)</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>Sum and Product Puzzle (after the first two announcements)</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>Basic Atoms and Agents Declaration</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>An Example of Static Constraint Declaration</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>An Example of Dynamic Constraint Declaration</td>
<td>31</td>
</tr>
<tr>
<td>8</td>
<td>Sample Use of Epistemic Constraints to Represent Public Announcements</td>
<td>31</td>
</tr>
<tr>
<td>9</td>
<td>Hello World vdel</td>
<td>32</td>
</tr>
<tr>
<td>10</td>
<td>Generated DOT Code for the Hello World vdel</td>
<td>32</td>
</tr>
<tr>
<td>11</td>
<td>vdel Description for the 2-Atoms-2-Agent (GLO) Scenario</td>
<td>33</td>
</tr>
<tr>
<td>12</td>
<td>2-Atom-2-Agent Scenario (GLO) with one Constraint</td>
<td>34</td>
</tr>
<tr>
<td>13</td>
<td>vdel Description for the 4-Atom-2-Agent (GLO) Scenario</td>
<td>34</td>
</tr>
<tr>
<td>14</td>
<td>vdel Description for the Game of Hexa</td>
<td>35</td>
</tr>
<tr>
<td>15</td>
<td>vdel Description for the Prisoners and Hats Puzzle</td>
<td>36</td>
</tr>
<tr>
<td>17</td>
<td>vdel Description of the Initial Scenario of the Muddy Children Puzzle</td>
<td>41</td>
</tr>
<tr>
<td>17</td>
<td>vdel Description of the Initial Scenario of the Consecutive Numbers Puzzle</td>
<td>46</td>
</tr>
<tr>
<td>18</td>
<td>vdel Description of the Sum and Product Puzzle with all the announcements</td>
<td>52</td>
</tr>
</tbody>
</table>
INTRODUCTION

1.1 Scope

Epistemic logic is a formal way of describing and reasoning about the knowledge of agents. It is often employed in the specification and verification of protocols in distributed and intelligent systems. In epistemic logic, epistemic descriptions are expressed as epistemic logical formulas whose interpretation yields a model of the system described. Kripke semantics (Meyer & Hoek 1995), also known as possible world semantics, provides a key for interpreting (or generating a model) which satisfies a given epistemic description.

According to Kripke semantics, a world is possible (or accessible) for a given agent if the information in that world is consistent with the information of that agent. For example, the epistemic formula $K_a \phi$ (i.e., agent $a$ Knows $\phi$) is true if and only if it is the case in all worlds that are accessible to the agent. As such, Kripke semantics for epistemic logic allows us to formalise reasoning about knowledge, and provides a concise way to represent the information of multiple agents such that it also represents information that agents have about each other, that is, high order knowledge (Ditmarsch, Hoek & Kooi 2007).

Furthermore, when information is communicated, we also expect some change in knowledge of agents involved in the communication. The formalisation of change in knowledge due to communication is the object of Dynamic Epistemic Logic (Ditmarsch, Hoek & Kooi 2007). As a result of information change, previous models describing epistemic states of agents prior to the communication may cease to represent current scenario, unless updated. In addition, communications or announcements may effect property changes in the system such that new properties are satisfied in certain states and some previously satisfied properties may not hold anymore.

This project seeks to create a tool which enables the visual observation of the dynamics of epistemic scenarios, from initial model generated from an input description, to models arising due to public announcements. This tool will enable the automatic generation and update of graphical models of epistemic scenarios, following epistemic descriptions including public announcements.

This project also builds on the theory of the module COMP521 (Knowledge Representation).

1.2 Problem Statement

Problem statement of the project is as follows: create a software tool that takes an epistemic description, and builds a Kripke model for it. The system should be able to accept public announcements, and update the model accordingly. For the original and the updated model, the user can submit epistemic logical formulas, and the system checks whether they are true or not in a designated state.
The system will be evaluated using some well-known epistemic scenarios like the 2-agent-2-atom scenario (the 2-agent version of the GLO-scenario in, Muddy Children Paradox, and the Sum and Product or Consecutive Numbers Scenario(Ditmarsch, Hoek & Kooi 2007).

1.3 Approach

In figure 1 we present an overview of the design of the vdel system. We provide a description language to the user, the user submits an epistemic description as a computer program written in the provided description language, the Epistemic Interpreter parses the description and generates a graphical Kripke model of the described scenario. The user can update the epistemic description by adding ‘public announcements’. The updated description is once again passed to the epistemic interpreter to regenerate the graphical model which incorporates the new ‘information’ provided.

Figure 1: Overview of the DEL Visualiser

The design of the Description Language Interpreter follows the Object Oriented Design approach. The following figure (Figure 3) presents a high level view of the object components which make up the Interpreter, as well as a high level sequence of their interaction.

The graphical rendering of the Kripke model is handled by the DOT Graph-drawing Tool (Gansner, Koutsofios & North 2009). This tool accepts a graph description in the dot language, and outputs a graphical image corresponding to the textual description of the graph. DOT imposes a hierarchical structure on graph drawings, enabling us to see the intuitive geometrical shapes which the Kripke models of certain scenarios possess. DOT also enables the drawing of undirected graphs (which captures the symmetry of S5 models) using fairly simple constructs, making it easier to automatically generate the corresponding statements.

1.4 Outcome
To describe given epistemic scenarios, there is need for a description language. This language will in turn require an interpreter which will translate the epistemic description into a graphical Kripke model. Furthermore, there is need to provide a language for expressing the epistemic properties to be checked on a given state in the generated model of the given scenario.

Thus in this project we have created the following:

a) a **description language** such that a user of the system can describe epistemic scenarios. Specifically, the user can use the language to describe well known epistemic puzzles such as the muddy children scenario and the Sum and Product scenario. This language also embodies constructs for expressing public announcements in the scenarios in question.

b) an **Interpreter for the language in (a)** whose purpose is to translate the given epistemic description into one instance of a graphical Kripke model. The user will feed the epistemic description into the Interpreter to obtain a graphical Kripke model.

c) a **model checker for epistemic properties** which will be employed in checking given epistemic properties on the generated model.

d) “**Epistemic Keypad**” a Graphical User Interface for entering epistemic properties into the model checker.

A major product of this project is a dynamic epistemic modelling tool called DEL Visualiser, vdel for short. This tool could be used to facilitate students’ learning and engagement in courses of Epistemic and Dynamic Epistemic Logic. Specifically, DEL Visualiser could be employed as a teaching tool in the module COMP521: Knowledge Representation.

### 1.5 Outline

This dissertation presents the dynamic epistemic modelling tool implemented in this project. We begin with a background in chapter two where we present the dynamic epistemic logic which this project implements. In chapter three we present the design of the vdel system, and give a tour of the implemented features in chapter four where we run tests (which also serve as our evaluation of the created tool) and present test results. In chapter five we present further evaluation of the vdel tool by briefly comparing it with the state-of-the-art dynamic epistemic modelling tool. We then present some professional and ethical issues relevant to the project in chapter six, and in chapter seven we conclude by stating the main findings in the work, and point the direction for future work.
In this chapter we present the dynamic epistemic logic which the vdel system implements. The presentation in this chapter is based on (Ditmarsch, Hoek & Kooi 2007) and (Ruan 2008). We begin with a short overview of epistemic logic and then present dynamic epistemic logic and logic of public announcements. We then conclude with some discussion on available dynamic epistemic model checkers.

2.1 Epistemic Logic

Epistemic logic has been influenced by the development of modal logic, especially by Kripke semantics. Hintikka built on Von Wright’s idea on modal logic (Wright 1951) in order to give one of the first clear semantics to the notions of knowledge and belief. However it was not until early 1960s that possible world semantics emerged. First found in Carnap’s work (Carnap 1946), it was enriched with the notion of accessibility between worlds by Hintikka (Hintikka 1957), which was further refined by Kripke (Kripke 1963). The ensuing semantics became very useful in interpreting various notions such as knowledge, belief, time, change and provability. Hence modal logic rapidly evolved into a main tool for reasoning in many different kinds of disciplines.

2.1.1 Language and Semantics of Epistemic Logic

Definition 2.1 (Possible World Model). Given a set of agents \( \mathcal{A} \), the language of Epistemic Logic (EL) is as follows:

\[
\varphi ::= p \mid \neg \varphi \mid \varphi \land \psi \mid K_i \varphi \mid C_B \varphi
\]

where \( i \in \mathcal{A} \), \( B \subseteq \mathcal{A} \).

Definition 2.2 (Epistemic Logic Semantics). Given a Kripke model \( \langle W, R_1, \ldots, R_n, \pi \rangle \) and a possible world \( w \), we have:

\[
M, w \models p ::= p \in \pi(s)
\]

\[
M, w \models \neg \varphi ::= \text{not } M, w \models \varphi
\]

\[
M, w \models \varphi \land \psi ::= M, w \models \varphi \land M, w \models \psi
\]

\[
M, w \models K_i \varphi ::= \text{for all } v \in W \text{ such that } w R_i v, M, v \models \varphi
\]

\[
M, w \models C_B \varphi ::= \text{for all } v \in W \text{ such that } w R_B^* v, M, v \models \varphi
\]
where $R_B^*$ is the transitive closure of $\bigcup_{i \in B} R_i$

Similar to the relations in temporal models, the relation $R$ in a Kripke model can also have more constraints. Here we mention three of such constraints:

Reflexivity: for all $x$: $(xRx)$;

Symmetry: for all $xy$: $(xRy \rightarrow yRx)$;

Transitivity: for all $xyz$: $(xRy \text{ and } yRz \rightarrow xRz)$.

For any relation $R$, if all of reflexivity, symmetry and transitivity hold, then we call $R$ an $S5$ relation, or an equivalence relation.

### 2.1.2 Possible world Semantics for Knowledge and belief

Possible world semantics for knowledge and belief asks the question: for a given agent, which are the possible worlds which are consistent with the information that the agent has. These possible and consistent worlds are said to be accessible for the agent. Thus according to Kripke semantics, an agent knows or believes that something is the case, if and only if it is the case in all the worlds that are accessible to the agent. This semantics, apart from allowing us to formalise our reasoning about knowledge and belief, provides a concise way to represent the information of an agent, add more agents to the scenario, and represent the information that the agents have about each other.

**Example 2.1.** Suppose that Peter is putting on a hat, $p$. The colour of the hat is either white or black. Peter cannot see $p$ so he does not know which colour it is; in other words, he considers both cases possible.

**Definition 2.3 (Possible World Model).** Given a set of atomic propositions $P$ and a set of $n$ agents, a possible world model is a tuple $(\mathcal{W}, R_1, \ldots, R_n, \pi)$, where $\mathcal{W}$ is a finite non-empty set of worlds, $R_i$ is a binary relation, and $\pi$ is a valuation function from $\mathcal{W}$ to $\wp(P)$.

Another name for possible world models is Kripke models. Going back to example 2.1, Figure 2 is a graphic representation of the Kripke model of the given example.

![Figure 2: A Simple Sample Graphical Kripke Model](image-url)
There are two possible worlds: one in which \( p \) is black, and the other in which \( p \) is white. Suppose \( p = \text{white} \) is the real world. Peter considers the \( p = \text{black} \) world is also possible. Suppose again that \( p = \text{black} \) is the real world, Peter would also consider the \( p = \text{white} \) world to be possible. As such we have an accessibility relation from \( p = \text{white} \) to \( p = \text{black} \), and vice versa. So we represent this accessibility with an undirected line, to show the symmetry. Moreover, whichever is the real world, Peter would consider that same world possible too, thus the reflective relation on each of the possible worlds.

### 2.2 Dynamic Epistemic Logic

Furthermore, many logicians have also considered cases in which agents in an epistemic scenario do communicate information among themselves. Hence knowledge and belief are by no means static in these scenarios. Therefore various approaches have been developed in order to formalise reasoning about information change, these various approaches are collected under the term “Dynamic Epistemic Logic”.

The development of dynamic epistemic logic was partly inspired by the view that the meaning of an utterance is best interpreted as a ‘cognitive program’ that changes the information states of participants in the discourse (Muskens, van Benthem & Visser 1997). Here, the semantics is not based on truth conditions but on update conditions which describe the information change which that utterance gives rise to (Veltman 1996).

Dynamic epistemic logic was also inspired by developments in dynamic modal logic which started as a specification language to reason about the correctness and behaviour of some aspects of numerical computations as well as computer programs (Harel 1984; Harel, Kozen & Tiuryn 2000; Pratt 1978). In dynamic modal logic, formulas of the form \( [\pi]\phi \) means ‘successfully executing program \( \pi \) yields a \( \phi \) state’. This led to dynamic epistemic logic by allowing the programs to describe information change, and then the language is equipped with epistemic operators so as to reason about information and its change.

In this project, as well, specifically in this chapter, we implement, and discuss, the Dynamic epistemic logic based on (Ditmarsch, Hoek & Kooi 2007). In addition to the language of epistemic logic, dynamic epistemic logic introduces action modalities which allow us to express actions explicitly. To the semantics of epistemic logic, dynamic epistemic logic introduces actions models which are a new semantic structure for actions.

#### 2.2.1 Language of Dynamic Epistemic Logic

**Definition 2.4 (DEL Language).** The logical language \( \mathcal{L}_{\text{DEL}} \) of Dynamic Epistemic Logic is inductively defined as

\[
\phi ::= q \mid \neg \phi \mid (\phi \land \psi) \mid K_i \phi \mid C_{B} \phi \mid [M, w] \phi
\]

Where \( q \) is a propositional atom, \( i \) is an agent, and \( B \) is a group of agents.
2.2.2 Semantics of Dynamic Epistemic Logic

State models capture the static view of multi-agent systems, essentially possible world models. However action models capture the uncertainty of agents concerning possible actions. But instead of having valuation for worlds, as in the case of state models, they have preconditions to actions, indicating under which conditions the actions could be executed.

**Definition 2.5 (State Model).** A state model $M$ for $n$ agents is a structure $\langle W, \sim_i, \pi \rangle$

Where $W$ is a finite not-empty set of states, $\sim_i$ is an equivalence relation on $W$, and $\pi$ is a valuation function from $W$ to $\wp(P)$.

**Definition 2.6 (Action Model).** An action model $M$ for $n$ agents is a structure $\langle W, \sim_i, pre \rangle$

where $W$ is a finite non-empty set of action points, $\sim_i$ is an equivalence relation on $W$, and $pre: W \rightarrow \mathcal{L}_{DEL}$ is a precondition function that assigns a precondition $pre(w)$ to each $w$ element of $W$.

Let $MOD$ be the class of state models and $ACT$ the class of $\mathcal{L}_{DEL}$ models. Then $\mathcal{L}_{DEL}$ update is an operation of the following type:

$\otimes: MOD \times ACT \rightarrow MOD$.

The operation $\otimes$ and the truth definition for $\mathcal{L}$ are defined by below:

**Definition 2.7 (Update, Truth).** Given a state model $M$ and an action model $M'$, we define $M \otimes M'$ as $(W', R', \pi')$,

where

$W' := \{ (w, w') | w \in W, w' \in W, M \models w \ \text{pre}(w) \}$,

$\pi'(w, w') := \pi_M(w)$,

$(w, w) \sim_i (w', w') \in R' := w \sim_i w' \in R, w \sim_i w' \in R$,

and where the truth definition is given by:

$M, w \models p := p \in V_M(w)$

$M, w \models \neg \phi := \neg M, w \models \phi$

$M, w \models \phi$ and $\psi := M, w \models \phi$ and $M, w \models \psi$

$M, w \models K_i \phi := \text{for all } w, w' \text{ with } w \sim_i w', M, w' \models \phi$
2.3 The Logic of Public Announcement

Public announcement is an action that informs a whole group of agents with a public message. The logic of public announcements is an extension of epistemic logic with dynamic modal operators to model the effects of public announcements. Originally proposed by Plaza (1989) who used a different notation, without dynamic modal operators, and did not incorporate common knowledge. Later on, Gerbrandy et al. (1997) and Baltag et al. (1998) would incorporate common knowledge together with further generalisations. Here, since public announcements can be modelled using action models, we present the logic public announcements as a special case of dynamic epistemic logic.

Definition 2.8 (Public Announcement Logic (PAL) Language). The language of Public Announcement Logic is inductively defined as

\[ \phi ::= q \mid \neg \phi \mid (\phi \land \psi) \mid K_i \phi \mid C_B \phi \mid [\phi] \psi \]

where \( q \) is a propositional atom, \( i \) is an agent, and \( B \) is a group of agents.

The language of PAL, like DEL, is also interpreted over state models. A singleton action model with universal access for all agents represents a public announcement. An action model,

\[ (M_0, w_0) = (\langle \{ w_0, \ldots, \{ w_0, \ldots, \rangle, pre), w_0 \rangle) \]

where \( \sim_i = \{ (w_0, w_0) \} \) for \( 1 \leq i \leq n \) and \( pre(w_0) = phi \), represents a public announcement of \( \phi \). In DEL, formula \( [M_0, w_0] \psi \) stands for ‘after executing action \( (M_0, w_0) \) it holds that \( \psi \)’, whereas in PAL, formula \( [M_0, w_0] \psi \) stands for ‘after public announcement of \( \phi \) it holds that \( \psi \)’. \( [M_0, w_0]phi \) is often simplified to \( [\phi] \psi \).

The semantics of PAL basically follows from that of the DEL. For example,

\[ M, w \models [\phi] \psi ::= M, w \models \phi \rightarrow M \otimes M_0, (w, w_0) \models phi \]

where \( (M_0, w_0) = (\langle \{ w_0, \ldots, \{ w_0, \ldots, \rangle, pre), w_0 \rangle \) with \( \sim_i = \{ (w_0, w_0) \} \) for \( 1 \leq i \leq n \) and \( pre(w_0) = \phi \).
We now turn attention to the verification of specified properties on dynamic epistemic scenarios.

2.4 Epistemic Model Checking

Model checking is the process of determining whether a given model satisfies a logical specification. Formally, we state the model-checking problem as follows: given a desired property which is expressed as a logical formula $\varphi$, and a model $M$ with a state $w$ in $M$, determine whether $M, w \vDash \varphi$.

We can identify three main steps in the model-checking process:

- **Modelling**: translate a system specification into a formalism that is accepted by a model-checking tool
- **Specification** – produce formal specifications, in the form of logic formulas which represent desired properties of the system
- **Verification** – test the specification against the model

Current epistemic model-checkers include MCK, by Gammie and van der Meyden (Raimondi & Lomuscio 2004), DEMO, by van Eijck (van Eijck 2004), and MCMAS, by Raimondi and Lomuscio (Raimondi & Lomuscio 2004). Both MCK and MCMAS require temporal epistemic specifications, as well as the interpreted systems architecture, wherein the results of actions are modelled by explicit representation of time. On the other hand DEMO is a truly dynamic epistemic model checker, wherein time is implicit.

DEMO allows for the specification and graphical display of epistemic models and action models in dynamic epistemic scenarios, and allows properties, in form of logical formulas, to be evaluated in specific states of the model. Hence it is possible to check given post condition, as effect of specific action executions in given epistemic models. DEMO is written in the functional programming language Haskell.

Whereas DEMO specifies state models and action models directly in the form of mathematical structures, MCK and MCMAS specify interpreted systems in the form of programs. In this project we introduce the VDEL model checker which combines some of the characteristics of being a true DEL model checker like DEMO, with the program formalism of MCK and MCMAS.
In this chapter we present the design of the vdel system. vdel provides a description language to the user, the user submits an epistemic description written in the provided description language, and the Epistemic Interpreter parses the description and generates a graphical Kripke model of the described scenario. The user can update the epistemic description by adding ‘public announcements’. The updated description is once again passed to the epistemic interpreter to regenerate the graphical model which incorporates the new ‘information’ provided. The overview of the design of the vdel system is as shown in Figure 1.

3.1 Design of the Description Language

The description language was designed to allow the description of initial scenarios and public announcements for epistemic and dynamic epistemic scenarios. Some well-known scenarios were used as benchmarks for the adequacy of the designed language to describe epistemic scenarios. Some of the benchmarks/test scenarios are the 2-atoms 2-agents (GLO) scenario, the muddy children puzzle, the Sum and Product puzzle, the Consecutive Numbers puzzle and Game of Hexa, and Prisoners and Hats puzzle. We take these scenarios mainly from (Ditmarsch, Hoek & Kooi 2007). We will show (in coming sections) how succinctly vdel represents these scenarios, giving a good confidence that the designed language would actually be able to model a wide range of epistemic and dynamic epistemic scenarios.

Please refer to appendix A for the design of the Description Language using BNF notation.

3.2 Evaluation of the Description Language

We proposed to evaluate the description language on its ability to describe well known epistemic scenarios. Particularly, the description language should be capable of being used to describe the Muddy Children Scenario and the Sum and Product Puzzle. Below, we use examples to show that in fact the design of the description language is sufficient to capture these epistemic scenarios.

2-Atom-2-Agent (GLO) Scenario

A rain sensor agent in Manchester is called man. A rain sensor agent in Liverpool is called liv. man observes the variable q. liv observes the variable p. q is true if it is raining in Manchester, or false otherwise. p is true if it is raining in Liverpool, or false otherwise.

Description:

```
begin
  boolean p;
  boolean q;
  agent liv;
  agent gron;
```
liv observes p;
Man observes q;
end

Code Listing 1: 2-Atom-2-Agent (GLO) Scenario

The Muddy Children Puzzle (for three children)

A group of children has been playing outside and are called back into the house by their father. The Children gather round him. Some of them have become dirty from the play and in particular: they may have mud on their forehead. Children can only see whether other children are muddy, and not if there is any mud on their own forehead...

Description:

begin
    boolean childOneMuddy;
    boolean childTwoMuddy;
    boolean childThreeMuddy;

    agent childOne;
    agent childTwo;
    agent childThree;

    childOne observes childTwoMuddy;
    childOne observes childThreeMuddy;
    childTwo observes childOneMuddy;
    childTwo observes childThreeMuddy;
    childThree observes childOneMuddy;
    childThree observes childTwoMuddy;
end

Code Listing 2: Muddy Children Puzzle (initial scenario)

...Father now says: “At least one of you has mud on his or her forehead. Will those who know whether they are muddy please step forward.” Nobody steps forward...

Updated Description:

begin
    boolean childOneMuddy;
    boolean childTwoMuddy;
    boolean childThreeMuddy;

    agent childOne;
    agent childTwo;
    agent childThree;

    childOne observes childTwoMuddy;
    childOne observes childThreeMuddy;
    childTwo observes childOneMuddy;
    childTwo observes childThreeMuddy;
    childThree observes childOneMuddy;
    childThree observes childTwoMuddy;

    /* the following updates to the description are made */
constraint (childOneMuddy==TRUE) OR (childTwoMuddy==TRUE) OR (childThreeMuddy==TRUE);
epistemic constraint childOne ¬knows childOneMuddy;
epistemic constraint childTwo ¬knows childTwoMuddy;
epistemic constraint childThree ¬knows childThreeMuddy;
end

Code Listing 3: Muddy Children Puzzle (after first announcement)

The Sum and Product Puzzle

A says to Sum and Prod: “I have chosen two natural numbers x and y such that 1<x<y and x+ y ≤ 100. I am now going to announce their sum s= x+y to Sum only, and their product p = x•y to Prod only. The content of these announcements remains a secret.” He acts accordingly. The following conversation between Sum and Prod then takes place.

1. Prod says: “I don’t know the numbers.”
2. Sum: “I knew that.”

Description:

begin
    int x(2,99);
    int y(3,99);
    agent sum;
    agent prod;
    constraint x<y;
    constraint x+y<=100;
    dynamic constraint s = x+y;
    dynamic constraint p = x*y;
    sum observes s;
    prod observes p;
    /* In the following we represent sum’s higher order knowledge first, */
    /* since it has temporal priority over prod’s own announcement */
    epistemic constraint sum knows (prod ¬knows(x AND y));
    epistemic constraint prod ¬knows(x AND y);
end

Code Listing 4: Sum and Product Puzzle (after the first two announcements)
3.3 Design of the Description Language (VDEL) Interpreter

The design of the Description Language Interpreter follows the Object Oriented Design approach. Figure 3 presents a high level view of the object components which make up the Interpreter, as well as a high level sequence of their interaction.

![High-Level Object Components of the Epistemic Description Interpreter](image)

**Figure 3:** High-Level Object Components of the Epistemic Description Interpreter

What follows is a description of the components in Figure 3.

### 3.3.1 Epistemic Description Parser

The Epistemic Description Parser is one of the major components of the VDEL interpreter. It parses the VDEL program describing the epistemic scenarios, checking for syntactic correctness of the code, as well as executing appropriate semantic actions to realise the goal of the interpreter.

The VDEL parser implements the LALR parsing technique (Aho et al. 2007). LALR parsing is attractive since it can be constructed to recognise most programming language constructs for which context-free grammars can be written, and it can be efficiently implemented. The main drawback of LALR technique is that it is too much work to construct an LALR parser by hand for a typical programming language grammar. Usually, a specialised tool, an LALR Parser Generator is needed. In this project we employed the CUP Parser Generator. CUP is written in Java, uses specifications that include embedded Java code and produces parsers implemented in Java (Hudson).

### 3.3.2 State Variable Generator

The goal of this component is to generate all the states possible from a given epistemic description. Given the atoms and all their possible values, this component generates all possible combinations of values of all the atoms given. For each combination (or set of state
values) generated, the Constraint Evaluator (Arithmetic and Boolean Logic Evaluator) is called to test whether that state satisfies all the given constraints. If the state values satisfy all the given constraints, then that set of state values is included in the set of valid states.

### 3.3.3 The Constraint Evaluator (CE)

The Constraint Evaluator embodies two sub-components: The Arithmetic and Boolean Logic Evaluator (ABLE) and the Epistemic Property Evaluator (EPE).

ABLE is responsible for evaluating given arithmetic expressions or Boolean expressions. Before an expression is given to ABLE to evaluate, the values of state variables must be substituted for those variables first. ABLE returns “true” or “false” (in the case of boolean expressions), or the real number being the value of the arithmetic operation performed.

ABLE implements the BNF grammar shown in appendix B.

### 3.3.4 The Epistemic Property Evaluator (EPE)

The Epistemic Property Evaluator (EPE) implements algorithms for the knowledge operator (first order and iterated knowledge operators). EPE operates on the entire generated graph. It evaluates those states of the Kripke model that satisfy the given Epistemic Constraints (truth set), and prunes those other states and accessibility relations which do not satisfy all the constraints (false set). Thus EPE is usually be called to carry out updates on the original graph based on the information contained in the Epistemic Constraints. Notice also (especially from the following grammar) that ABLE is a sub-routine of EPE.

EPE implements the BNF grammar shown in appendix C.

### 3.3.5 Kripke Model Generator

The diagram below illustrates the graph data structure which stores the Kripke model of the epistemic scenario. We give a state ID to each valid state generated, and map it to the agents in the scenario. Each agent is in turn mapped to a list of all its successor states from the current state. And with an identifier for a given successor state we can locate the state object for that state.

This data structure for the generated Kripke model was basically designed with the K-operator (“knows”) in mind since the labelling of accessibility relations are according to the agents that cannot distinguish between two the given states. As such this design makes the implementation of the K-operator more intuitive. Searches involving the D-operator (distributed knowledge) and E-operator (general knowledge) are also made more intuitive.
In the following subsections we describe algorithms used by the model checker, as well as the other property evaluators, to search the graph data structure. In addition we present a general description of algorithms for the C-Operators, E-Operators and D-Operators. We also give an overview of how the graph data structure is updated following a public announcement, and how the DOT statements are generated from this data structure.

### 3.4 Model Checker

The vdel model checker accepts epistemic logical properties, and checks whether they are true, or false, on a designated state in the generated model. The following gives an overview of some of the algorithms to be implemented for some of the model checker’s tasks.

#### 3.4.1 Algorithms for the Epistemic Operators

In the following algorithms, when we refer to “first order” we mean those epistemic formulars which have a single epistemic operator over a propositional sub-formular, whereas iterated (or higher order) operators will refer to those epistemic operators which have at least one epistemic operator over a first-order epistemic formular. For example, a first-order k-operator would refer to the K-operator in expressions adhering to the following grammar,
whereas an iterated first-order k-operator would refer to the k-operator over the “statement” as seen in the following grammar.

\[
\text{statement ::= IDENTIFIER KNOWS boolean_expr} \\
\quad | \text{IDENTIFIER NOT KNOWS boolean_expr} \\
\quad | \text{IDENTIFIER KNOWS phi} \\
\quad | \text{IDENTIFIER NOT KNOWS phi};
\]

Where boolean_expr and phi are as already defined in grammar shown in appendix A, and IDENTIFIER semantically refers to an Agent in the scenario. Let us call this Agent Ag

**Algorithm 1: First order K-Operation**

**Algorithm: K-Operator (First Order)**

Given the graph data structure G

for each state \( u \) in \( G \)

\[
\text{Case boolean_expr:} \\
\quad \text{for each Ag-successor state } v, \\
\quad \quad \text{substitute state values of } v \text{ in the boolean_expr part of the statement and pass the resulting expression to ABLE}
\]

end for

\[
\text{Case phi:} \\
\quad \text{Substitute state values of } u \text{ in the phi part of the statement (that is, create a boolean expression which equates the variables in phi to their corresponding values in } u) \\
\quad \text{for each Ag-successor state } v, \\
\quad \quad \text{Substitute state values of } v \text{ in the resulting boolean expression} \\
\quad \quad \text{Pass the resulting expression to ABLE}
\]

end for

If ABLE returns true for all the successor states,

Then add \( u \) to the truth set, \( T \)

Otherwise add \( u \) to the false set \( F \)

end for

Output sets \( T \) and \( F \)
Algorithm 2: Iterated K-Operation

Algorithm: K-Operator (Iterated K-Operator)

We proceed in a bottom-up manner.

Evaluate T and F for the first order K-operator.

For each subsequent k-operator encountered ‘upwards’

Create temporary truth and false sets (TT and FF)

for each state u in the model

Create the set S of all successor states v following Ag’s accessibility relations

If all the elements of S are contained in T

Then add u to TT

Otherwise add u to FF

end for

end for

Replace T with TT, and F with FF

Output sets T and F
Algorithm 3: K-Operations involving Conjunctions, Disjunctions, Implications and Negations

**Algorithm: NOT K-Operator**

Obtain the truth sets T and F  
Swap T with F  
Output sets T and F

**Algorithm: “epistemic_fact1 AND epistemic_fact2”**

Obtain the truth sets T1 and F1 for epistemic_fact1  
Obtain the truth sets T2 and F2 for epistemic_fact2  
Let T be the set intersection of T1 and T2  
Let F be the set intersection of F1 and F2  
Output sets T and F

**Algorithm: “epistemic_fact1 OR epistemic_fact2”**

Obtain the truth sets T1 and F1 for epistemic_fact1  
Obtain the truth sets T2 and F2 for epistemic_fact2  
Let T be the set union of T1 and T2  
Let F be the set union of F1 and F2  
Output sets T and F

**Algorithm: “epistemic_fact1 IMPLIES epistemic_fact2”**

Obtain the truth sets T1 and F1 for NOT (epistemic_fact1)  
Obtain the truth sets T2 and F2 for epistemic_fact2  
Let T be the set union of T1 and T2  
Let F be the set union of F1 and F2  
Output sets T and F
Algorithm 4: First Order General Knowledge ("Everybody Knows") Operator

**Algorithm: E-Operator (First Order)**

Given the graph data structure G

for each state u in G
    for each agent, Ag
        Case boolean_expr:
            for each Ag-successor state v,
                substitute state values of v in the boolean_expr part of the statement and pass the resulting expression to ABLE
        end for
        Case phi:
            Substitute state values of u in the phi part of the statement (that is, create a boolean expression which equates the variables in phi to the their corresponding values in u)
            for each Ag-successor state v,
                Substitute state values of v in the resulting boolean expression
                Pass the resulting expression to ABLE
            end for
        end for
    end for
If ABLE returns true for all the successor states of u for all agents
    Then add u to the truth set, T
Otherwise add u to the false set F
end for
Output sets T and F

Algorithm 5: Iterated General Knowledge ("Everybody Knows") Operator

**Algorithm: E-Operator (Iterated)**

Given the graph data structure G

We proceed in a bottom-up manner

Evaluate T and F for the first order E-operator

For each subsequent E-operator encountered ‘upwards’
    Create temporary truth and false sets (TT and FF)
    for each state u in G
        for each agent, Ag
            Update the set S of all successor states v with all Ag-successor states
        end for
        If all the elements of S are contained in T
            Then add u to TT
        Otherwise add u to FF
    end for
End for
Replace T with TT, and F with FF
Output sets T and F
Algorithm 6: Common Knowledge

**Algorithm: C-Operator (Common Knowledge)**

Given the graph data structure $G$

for each state $u$ in $G$
  for each agent $Ag$
    
    **Case boolean_expr:**
    for each $Ag$-successor state $v$,
    substitute state values of $v$ in the boolean_expr part of the statement and pass the resulting expression to ABLE
  end for
  
  **Case phi:**
  Substitute state values of $u$ in the phi part of the statement (that is, create a boolean expression which equates the variables in phi to the their corresponding values in $u$)
  
  for each $Ag$-successor state $v$,
    Substitute state values of $v$ in the resulting boolean expression
    Pass the resulting expression to ABLE
  end for

end for

If ABLE returns true for all successor states, $v$, of all states in $G$
  Then output “TRUE”
Otherwise output “FALSE”
**Algorithm 7: Distributed Knowledge**

**Algorithm: D-Operator (Distributed Knowledge)**

Given the graph data structure $G$

for each state $u$ in $G$
    for each agent $Ag$
        Case boolean_expr:
            for each $Ag$-successor state $v$,
                substitute state values of $v$ in the boolean_expr part of the
                statement and pass the resulting expression to ABLE
        end for
        Case phi:
            Substitute state values of $u$ in the phi part of the statement (that is,
            create a boolean expression which equates the variables in phi to the
            their corresponding values in $u$)
            for each $Ag$-successor state $v$,
                Substitute state values of $v$ in the resulting boolean expression
                Pass the resulting expression to ABLE
        end for
    end for

If ABLE returns true for all successor states, $v$, of all agents $Ag$
    Then add $u$ to the truth set, $T$
Otherwise add $u$ to the false set $F$
End for

**Algorithm 8: Graph Update after an announcement**

**Algorithm: Updating the graph after an epistemic announcement**

Input: The truth and false sets ($T$ and $F$);
The Graph Data Structure, $G$;
The identifier, $agId$, of the agent that performed the uppermost (or outermost) $k$-operation

For each of the states, $s$, in $F$
    //we exploit the symmetry of the relations
    For each of $agId$’s successors, $t$
        Delete $s$ from the list of $t$-$agId$-successors
    End for
Delete $s$’s node in $G$
End for
3.5 DOT Code Generator

This component takes as input the populated graph data structure and generate DOT statements corresponding to the structure of the graph. The following is a description of the algorithm for this component.

Algorithm 9: DOT Code Generation

Algorithm: Translating the Graph Data Structure to DOT

Input: the graph data structure G, the StateMap M
Output: DOT Graph Statements

For each of the state, s, in G
    For each agent, agId
        Pick a colorIndex that corresponds to agId’s index
        For each successor, t, of agId
            Generate DOT statement connecting s and t
            Remove s from the list of t-agid-successor states
        End for
    End for
End for

For each state, s in M
    Generate a DOT state label corresponding to the label of state s
End for

3.6 Graph Drawing

The graphical rendering of the Kripke model was done by the DOT Graph-drawing Tool. This tool accepts a graph description in the dot language, and outputs a graphical image corresponding to the textual description of the graph. DOT imposes a hierarchical structure on graph drawings, and thus enables us to see the intuitive geometrical shapes which the Kripke models of certain scenarios possess. DOT also enables the drawing of undirected graphs (which captures the symmetry of S5 models) using fairly simple constructs, making it easier to automatically generate the corresponding statements. The following is an example of the graph output when experimented on the 2-atom-2-agent (GLO Scenario):
The preference for DOT over its counterpart, neato (North 2004), is based on our experiments which were carried out with graphs that have well-known geometrical structure, for example, the muddy children puzzle and consecutive numbers puzzle. DOT reproduced the expected shapes of these models, whereas neato did not.

### 3.7 The Epistemic Keypad

The Epistemic Keypad provides an interface for submitting epistemic logical properties to be checked by the model checker. The following presents the Graphical User Interface design of the Epistemic Keypad, using the example of the Muddy Children scenario.

The **State Selection dropdown** list enables the user to select a state of the model on which to check a property. The **Agent Selection dropdown** list contains the list of all the agents given in the epistemic description. For each agent selected, a fly-out menu emerges with the list of individual epistemic operators possible with that agent. In the above example we see that the user could select “ChildThree **Knows**…” or “ChildThree **¬Knows**…”

The **Atom Selection dropdown** provides a list of all the atoms declared in the epistemic description. Here, instead of using a numeric or separate keypad to select the values of a desired atom, we provide fly-out menu containing all the possible values of the selected atom. The view is shown above as the `childOneMuddy` atom is selected, we see that a list containing its possible values emerges: **TRUE** (childOneMuddy == true); **FALSE** (childOneMuddy == false); `childOneMuddy` (childOneMuddy == true OR childOneMuddy == false).
Finally we have other operators such as the conjunction operator (AND), disjunction operator (OR), implication (IMPLIES), and parenthesis (to improve readability of formulas). The Display shows the formula being constructed.

The “Check Model” submits the constructed formula to the Model Checker. When model checking finishes, the result is shown on the Display: The result is “TRUE”/”YES” if the submitted formula is true in the selected state of the model, “FALSE”/”NO” if the selected formula is false in the selected state of the model, or “Syntax Error” if a syntactically incorrect formula is submitted.

### 3.8 Language for Epistemic Properties

The epistemic properties follow the grammar shown in appendix C. Some examples (using the muddy children puzzle) of properties that can be constructed with the grammar are as follows:

- (childOneMuddy==TRUE) OR (childTwoMuddy==TRUE) OR (childThreeMuddy==TRUE)
- childOne $\neg$knows childOneMuddy
- childOne knows (childTwo $\neg$knows childTwoMuddy)
- childOne knows (childTwo knows (childOne knows childTwoMuddy==TRUE))
- (childTwoMuddy == TRUE ) IMPLIES ((childOne knows (childTwoMuddy == TRUE)) AND (childTwo knows (childTwoMuddy == TRUE)))
3.9 User Interaction

The activity diagram of the vdel is shown in Figure 7 below. The diagram shows the three major activities the user could carry out on the system. The user will be able to describe the epistemic scenarios using the description language we have designed. The user would then call the vdel interpreter to generate a Kripke model from the given description. This generated model could then be checked for epistemic logical properties by the user. Subsequently, the user could update the epistemic description with public announcements (depicted as “constraints” within the epistemic description), or modify other parts of the description, and regenerate the Kripke model based on the updated description.

Figure 7: Activity Diagram of the Software Tool (DEL Visualiser)

DEL Visualiser is launched as a command line tool. The default launch option is to load the epistemic keypad following the generation of the Kripke model. The main command line argument will be the .vdel file containing the epistemic description to be interpreted.

3.10 Summary of Design

In this chapter we presented the design of the vdel tool, beginning from the vdel description language to be used for describing epistemic and dynamic epistemic scenarios. Then we presented the design of the vdel interpreter used to interpret the described epistemic scenario into a graphical model. When the Kripke model is generated, we would need a model checker to check given epistemic properties on the generated model. The design of the vdel model checker, as well as the GUI interface to this model-checker was also presented.
In this chapter we present details of the implemented tool, vdel. We begin by describing the vdel description language from the point of the view of the user or programmer who would use the description language to describe epistemic and dynamic epistemic scenarios. Next we present some sample scenarios and show how they are completely described using vdel; we presented the graphical models of these test scenarios (including the updated models due to public announcements). We also show how the vdel model-checker is used to check epistemic properties in the initial model, as well as on the updated model due to public announcement. We able to verify the results of these puzzles are in accordance with literature on the puzzles and scenarios under consideration.

4.1 The vdel Language

The vdel language is a programming language which can be used to describe epistemic scenarios for visualisation purposes. By visualisation we mean the generation of graphical Kripke model for the described epistemic scenario. The Kripke model is updated as a result of ‘public announcements’ being introduced into the epistemic description.

vdel program structure consists of the following sections:

- a “begin” statement
- atoms declaration
- agents declaration
- optional constraints declarations
- an “end” statement

4.1.1 Atoms declaration

vdel provides three constructs for defining propositional atoms, namely: boolean, int, enum.

The boolean construct enables basic declaration of propositional atoms. For example, a propositional atom \( q \): “The patient is alive”, would be defined as follows in the vdel program:

```vdel
boolean q;
```

When we have a group of propositional atoms that differ only by the value of a natural number, we could declare such group of atoms succinctly using the int construct. For example, consider the following group of propositional atoms:

- “the sum is 10”
“the sum is 11”
“the sum is 12”

Instead of having three boolean declarations we would have a single int declaration as follows:

    int sum(10,12);

The comma-separated numbers in parenthesis represent the lower and upper bounds, respectively. vdel produces states in which sum is 10, 11, 12,…(denoted by “sum=10”, “sum=11” etc.). It is assumed that if a state contains, say, “sum=10”, then sum is 10 is true in that state, whereas sum is 11 (or any other number) is false in that state (i.e. sum is not 10 is false in that state)

Likewise, where we have a group of propositional atoms that differ only in a string variable, or a set of non-sequential natural numbers, we could define them succinctly using the enum construct. For example, consider the following group of atoms:

    “it is sunny in Liverpool”
    “it is rainy in Liverpool”
    “It is cloudy in Liverpool”
    “It is snowy in Liverpool”

We could represent the above as:

    enum liv(sunny, rainy, cloudy, snowy);

4.1.2 Agents Declaration

The construct for agent declaration is simple. Say we have two agents, namely, cam and gyro. Their declaration in vdel would look as follows:

    agent cam;
    agent gyro;

4.1.3 Constraints Declarations

This section is optional and may include declarations such as:

- static propositional constraints
- dynamic propositional constraints
- epistemic constraints
- scopes of observation

Scopes of observation are statements that declare which atoms are 'observed' by which agents. For example given the following atoms declaration:
begin
    boolean p;
    boolean q;
    boolean r;

    agent a;
    agent b;
end

Code Listing 5: Basic Atoms and Agents Declaration

We could append the following observation statements which say that *agent a knows the value of p and q*, and *agent b knows the value of q and r*:

```
    a observes p;
    a observes q;
    b observes q;
    b observes r;
```

The *observation statement* allows the vdel interpreter to compute which states of the model are distinguishable by each of the agents, in order to establish the accessibility relations between the states of the model, for the given agent.

*Static constraints* are the propositional expressions formed according to the BNF grammar shown in Appendix A. They are basically boolean expressions whose operands are one or more of the atoms declared in the atoms declaration section of the epistemic description. See an example in the following vdel code fragment:

```
begin
    int area (20, 50);
    int peri (5, 20);

    agent a;
    agent p;

    //the following is the static constraint
    constraint ((area + peri) > 30) && ((area + peri) < 65);
end
```

Code Listing 6: An Example of Static Constraint Declaration

The static constraint in the above code snippet says that the sum of *area* and *peri* must be between 30 and 65.

*Dynamic propositional constraints* are different because they are arithmetic expressions (“expr” fragment in the BNF grammar shown in Appendix B), and the value of this expression is used to create a new atom. Thus, for each evaluation of a dynamic constraint, we obtain a new propositional atom. Take for example the following code fragment:
begin
  boolean fire;
  boolean smoke;
  int humid (30, 120);

  agent monitor;

  //the following line is a dynamic constraint declaration
  dynamic constraint danger = (fire || smoke);

  monitor observes danger;
end

Code Listing 7: An Example of Dynamic Constraint Declaration

The dynamic constraint expression creates a new group of propositional atoms, danger, using the values of the expression on the right hand side of the '=' symbol. Thus, in the generated model, the state variables will include three atomic variables, namely: fire, smoke, danger.

**Epistemic constraints** are epistemic formulas. The grammar followed by epistemic constraints is shown in Appendix C. The following example shows the use of epistemic constraint declarations to represent a typical public announcement in a dynamic epistemic scenario:

begin
  enum c1 (red, white);
  enum c2 (red, white);
  enum c3 (red, white);

  agent p1;
  agent p2;
  agent p3;

  p1 observes c2;
  p1 observes c3;
  p2 observes c3;

  //the following lines is are epistemic constraint
  epistemic constraint p1 !@ [c1];
  epistemic constraint p2 @ p3 !@ [c3];
end

Code Listing 8: Sample Use of Epistemic Constraints to Represent Public Announcements
The first epistemic constraint in the code snippet above says that *agent p1 does not know the value of c1* (note the use of square brackets to denote a disjunction of all the values of the atomic variable *c1*). The second epistemic constraint says that *agent p2 knows that agent p3 does not know the value of c3*.

Generally, in vdel, when two or more epistemic constraints are declared, the succeeding constraint is treated as being in a different time frame from the preceding constraint. As such, the preceding epistemic constraint is computed, and the model updated, before the next constraint is treated. Therefore the order of declaration of the epistemic constraints is important in generating the right model for the scenario. In all, the declared constraints must be satisfied by every state of the generated model.

A list of operators which can be used in the epistemic constraints, following the grammar in Appendix B, can be seen in table 1.

### 4.2 The vdel Interpreter: Test and Evaluation

The vdel interpreter translates the epistemic scenario, represented by the vdel program, into a corresponding graphical Kripke model. In this section we use some popular epistemic scenarios to demonstrate the generation of Kripke models using the vdel Interpreter.

**Hello_world vdel**

```vdel
begin
  boolean p;
  agent liv;
  liv observes p;
end
```

**Code Listing 9: Hello World vdel**

Running the vdel interpreter on the above program yields a `.dot` file with the following output code:

```dot
graph G {
  1–1 [color=black];
  2–2 [color=black];
  1[label="p=false " ];
  2[label=" p=true " ];
}
```

**Code Listing 10: Generated DOT Code for the Hello World vdel**

Running the dot graph-drawing engine on the above dot code yields the graphical model as shown below.
2-Atom-2-Agents (GLO) Scenario

```
begin
  boolean p;
  boolean q;

  agent liv;
  agent gron;

  liv observes p;
  gron observes q;
end
```

Code Listing 11: vdel Description for the 2-Atoms-2-Agent (GLO) Scenario

The generated dot code is shown in appendix C. The graphical model is as follows.

Figure 9: Generated Model for the 2-Atoms-2-Agent (GLO) Scenario
2-Atom-2-Agent(GLO) Scenario with One Constraint

begin
  boolean p;
  boolean q;
  agent liv;
  agent gron;
  liv observes p;
  gron observes q;
  constraint p==true;
end

Code Listing 12: 2-Atom-2-Agent Scenario (GLO) with one Constraint

Figure 10: Generated Model for the 2-Atom-2-Agents (GLO) Scenario with one Constraint

4-Atom-2-Agents(GLO) Scenario

begin
  boolean p;
  boolean s;
  boolean m;
  boolean n;
  agent liv;
  agent gron;
  liv observes p;
  gron observes s;
  liv observes m;
  gron observes n;
end

Code Listing 13: vdel Description for the 4-Atom-2-Agent (GLO) Scenario
Figure 11: Generated Model for the 4-Atom-2-Agent (GLO) scenario

Game of Hexa

begin
    int card1(0, 2);
    int card2(0, 2);
    int card3(0, 2);

    agent anne;
    agent bill;
    agent cath;

    anne observes card1;
    bill observes card2;
    cath observes card3;

    constraint card1 != card2 && card2 != card3 && card1 != card3;
end

Code Listing 14: vdel Description for the Game of Hexa
The constraint statement in the above listing states that no two cards are equal. This eliminates repetition of the same card in the model.

![Generated Model for the Game of Hexa](image)

**Figure 12: Generated Model for the Game of Hexa**

Prisoners and Hats Puzzle (5-hat variant, 3 hats ON) from (Prisoners and Hats Puzzle)(Wikipedia)

begin

    enum A(red, blue);
    enum B(red, blue);
    enum C(red, blue);

    agent P1;
    agent P2;
    agent P3;

    P1 observes B;
    P1 observes C;
    P2 observes C;

    constraint !(A==red && B==red && C==red);

end

**Code Listing 15: vdel Description for the Prisoners and Hats Puzzle**

In above listing the constraint is used to eliminate the situation where three red caps are used. This is due to the fact that only two red caps are specified in the 5-hat variant of the game.
Figure 13: Generated Model for the Prisoners and Hats Puzzle (5-hat variant, 3 hats ON)
4.3 The Epistemic Keypad

The Epistemic Keypad provides a Graphical User Interface for checking epistemic properties on the generated model. A screen-shot of the Epistemic Keypad is shown below:

![Figure 14: Screen-shot of the Epistemic Keypad](image)

The Epistemic Keypad is launched for the Interpreted vdel program. Thus in the keypad we provide a list of all states, a list of all the agents and a list of all atoms in the current model. The keypad features numeric keys, boolean and logical operators, and epistemic operators. The Keypad allows the user to visually construct properties to be checking on the selected state of the current model. The constructed formula for the property is shown on the text area (“display”) over the key area. To the right of the display are some accessibility buttons that allows the user to browse the history of properties entered on the keypad, and to edit and reuse them.

The “GM” button to the left of the display gives a quick access to the graphical view of the current model. The “CM” button is enabled whenever the model checker returns “false” for a given property. “CM” stands for “Counter Model”, it gives a quick access to the graphical view of a counter model that proves that the given property is false. The “ST” button is used to generate a syntax tree of a given property, this helps the user to cross-check whether the property being checked corresponds with the intended (for seemingly ambiguous properties which may have more than one way of being parsed).

The table below shows the meaning some of the special function keys on the Keypad.
Table 1: Meaning of Symbols on the Epistemic Keypad

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>NOT</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>-&gt;</td>
<td>IMPLIES</td>
</tr>
<tr>
<td>!=</td>
<td>NOT EQUALS</td>
</tr>
<tr>
<td>==</td>
<td>EQUALS</td>
</tr>
<tr>
<td>&lt;=</td>
<td>LESS THAN OR EQUALS</td>
</tr>
<tr>
<td>&gt;=</td>
<td>GREATER THAN OR EQUALS</td>
</tr>
<tr>
<td>&lt;</td>
<td>LESS THAN</td>
</tr>
<tr>
<td>&gt;</td>
<td>GREATER THAN</td>
</tr>
<tr>
<td>mod</td>
<td>MODULUS FUNCTION</td>
</tr>
<tr>
<td>sqr</td>
<td>SQUARE FUNCTION</td>
</tr>
<tr>
<td>sqrt</td>
<td>SQUARE ROOT</td>
</tr>
<tr>
<td>@</td>
<td>KNOWS (K Operator)</td>
</tr>
<tr>
<td>#</td>
<td>EVERYBODY KNOWS</td>
</tr>
<tr>
<td>:&gt;</td>
<td>DISTRIBUTED KNOWLEDGE</td>
</tr>
<tr>
<td>C</td>
<td>COMMON KNOWLEDGE</td>
</tr>
<tr>
<td>true</td>
<td>TRUE</td>
</tr>
<tr>
<td>false</td>
<td>FALSE</td>
</tr>
<tr>
<td>+</td>
<td>ADDITION</td>
</tr>
<tr>
<td>-</td>
<td>SUBTRACTION</td>
</tr>
<tr>
<td>/</td>
<td>DIVISION</td>
</tr>
<tr>
<td>*</td>
<td>MULTIPLICATION</td>
</tr>
</tbody>
</table>

4.4 The vdel Model Checker: Test and Evaluation

The Epistemic Keypad passes the constructed formular to the vdel model checker. The property specified in the formular could be checked on a selected state, or checked on the entire model. The model checker returns “true” if the property is true on the state (or model), or “false”, with a counter model, if otherwise.

The epistemic properties which can be checked by the model checker are the same as the epistemic formulars that can be expressed in the epistemic constraints declaration of a vdel code (see Appendix B).

In the following sections, we present results obtained from the vdel model checker in studying some well-known epistemic scenarios.
### Table 2: Model Checking 2-Atom-2-Agent (GLO) - Inputs and Outputs

<table>
<thead>
<tr>
<th>Epistemic Formular</th>
<th>vdEl Construct</th>
<th>Output from Model Checker</th>
</tr>
</thead>
<tbody>
<tr>
<td>M, 1 (\vdash) (K_{\text{gron}} q \lor K_{\text{gron}} \neg q)</td>
<td>(\text{gron}@[q])</td>
<td>true</td>
</tr>
<tr>
<td>M, 1 (\vdash) (K_{\text{liv}} p \lor K_{\text{liv}} \neg p)</td>
<td>(\text{liv}@[p])</td>
<td>true</td>
</tr>
<tr>
<td>M, 1 (\vdash) (K_{\text{gron}} \neg(K_{\text{liv}}p \lor K_{\text{liv}}\neg p))</td>
<td>(\text{gron}@\text{liv}!@[p])</td>
<td>false Figure 15: cm1</td>
</tr>
<tr>
<td>M, 1 (\vdash) (K_{\text{gron}} K_{\text{liv}} K_{\text{gron}} \neg(K_{\text{liv}}p \lor K_{\text{liv}}\neg p))</td>
<td>(\text{gron}@\text{liv}@\text{gron}!@[p])</td>
<td>false Figure 16: cm2</td>
</tr>
<tr>
<td>M (\vdash) (K_{\text{gron}} K_{\text{liv}} K_{\text{liv}}(K_{\text{gron}} q \lor K_{\text{gron}} \neg q))</td>
<td>(\text{gron}@\text{liv}!@[q])</td>
<td>true</td>
</tr>
<tr>
<td>M (\vdash) (K_{\text{gron}} K_{\text{liv}} K_{\text{liv}}(K_{\text{gron}} q \lor K_{\text{gron}} \neg q))</td>
<td>(\text{gron}@\text{liv}!@[q])</td>
<td>true</td>
</tr>
<tr>
<td>M, 2 (\vdash) EEE (\neg(K_{\text{gron}} q \lor K_{\text{gron}} \neg q))</td>
<td>(###\text{gron}@[q])</td>
<td>true</td>
</tr>
<tr>
<td>M (\vdash) C ((K_{\text{gron}} q \lor K_{\text{gron}} \neg q))</td>
<td>(\text{gron}@[q])</td>
<td>true</td>
</tr>
<tr>
<td>M, 2 (\vdash) D ((p \rightarrow q))</td>
<td>(:(p\Rightarrow true \rightarrow q\Rightarrow true))</td>
<td>true</td>
</tr>
<tr>
<td>M, 2 (\vdash) E ((K_{\text{gron}} q))</td>
<td>#gron@[q]=true</td>
<td>Syntax error: column 9</td>
</tr>
<tr>
<td>M, 2 (\vdash) E ((K_{\text{gron}} q))</td>
<td>#gron@[q]=true</td>
<td>false Figure 16: cm2</td>
</tr>
</tbody>
</table>

**Figure 15: cm1**

1: \(p=false, q=false\)

3: \(p=true, q=false\)

2: \(p=false, q=true\)
4.4.1 Case Study: Muddy Children Puzzle

The following is taken from past exam paper of COMP521: Knowledge Representation (January 2011, Question 4).

We model the muddy children puzzle for the case of three children \{c1, c2, c3\}, where child c1 and c2 are muddy. Let c1M be an atom denoting that child 1 is muddy: similar for c2M and c3M. Let onemuddy = (c1M \lor c2M \lor c3M): it says that there is at least one muddy child. Similarly, define twomuddy as (c1M \land c2M) \lor (c1M \land c3M) \lor (c2M \land c3M): it says that at least two children are muddy.

Description of the initial scenario:

```
begin
    boolean c1M;
    boolean c2M;
    boolean c3M;

    agent c1;
    agent c2;
    agent c3;

    c1 observes c2M;
    c1 observes c3M;
    c2 observes c1M;
    c2 observes c3M;
    c3 observes c1M;
    c3 observes c2M;
end
```

Code Listing 16: vDel Description of the Initial Scenario of the Muddy Children Puzzle
The generated model of the initial scenario is as follows:

Figure 17: Generated Model of the Initial Situation in the Muddy Children Puzzle

Table 3: Model Checking Muddy Children Puzzle (Initial Scenario) - Inputs and Outputs

<table>
<thead>
<tr>
<th>Epistemic Formular</th>
<th>vdel Construct</th>
<th>Output from Model Checker</th>
<th>Counter-Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M, 7 ⊨ K_{c_1}(c_2M==true)$</td>
<td>$c_1@c_2M==true$</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>$M, 7 ⊨ K_{c_2}(c_2M==true)$</td>
<td>$c_2@c_2M==true$</td>
<td>false</td>
<td>Figure 18: cm3</td>
</tr>
<tr>
<td>$M, 7 ⊨ E(c_2M==true)$</td>
<td>#@c_2M==true</td>
<td>Syntax error: column 2</td>
<td></td>
</tr>
<tr>
<td>$M, 7 ⊨ E(c_2M==true)$</td>
<td>#c_2M==true</td>
<td>false</td>
<td>Figure 18: cm3</td>
</tr>
<tr>
<td>$M, 7 ⊨ K_{c_2}(\neg c_1M==true)$</td>
<td>$c_2@(c_1M==true</td>
<td></td>
<td>c_2M==true</td>
</tr>
<tr>
<td>$M, 7 ⊨ E(\neg c_1M==true)$</td>
<td>#(c_1M==true</td>
<td></td>
<td>c_2M==true</td>
</tr>
<tr>
<td>$M, 7 ⊨ C(\neg c_1M==true)$</td>
<td>$(c_1M==true</td>
<td></td>
<td>c_2M==true</td>
</tr>
<tr>
<td>$M, 7 ⊨ C(\neg c_2M==true)$</td>
<td>$(c_1M==true&amp;&amp;c_2M==true)</td>
<td></td>
<td>$(c_1M==true&amp;&amp;c_3M==true)</td>
</tr>
</tbody>
</table>
Now the father announces that at least one child is muddy, and asks the muddy children to come forward if they know they are muddy. To represent this announcement we insert the following constraint into the original description.

```plaintext
constraint ((c1M == true) || (c2M == true) || (c3M == true));
```

We then recompile the program to obtain the following model. We see that the state in which all three children are not muddy has been deleted from the puzzle because all the children would consider it absurd since at least one child is muddy.

**Agent “Liv” is red; Agent “gron” is black**

*Figure 18: cm3*

*Figure 19: Generated Model the Muddy Children Puzzle after the First Announcement*

*We then repeat the following model-checking on the updated model.*
Table 4: Model Checking Muddy Children Puzzle – After the First Announcement - Inputs and Outputs

<table>
<thead>
<tr>
<th>Epistemic Formular</th>
<th>vdel Construct</th>
<th>Output from Model Checker</th>
<th>Counter-Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M, 6 \vdash K_{c_1}(c_2M==true)$</td>
<td>$c_1@c_2M==true$</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>$M, 6 \vdash K_{c_2}(c_2M==true)$</td>
<td>$c_2@c_2M==true$</td>
<td>false</td>
<td>Figure 20 : cm4</td>
</tr>
<tr>
<td>$M, 6 \vdash E(c_2M==true)$</td>
<td>$#c_2M==true$</td>
<td>false</td>
<td>Figure 20 : cm4</td>
</tr>
<tr>
<td>$M, 6 \vdash K_{c_2}($onemuddy==true$)</td>
<td>$c_2@(c_1M==true</td>
<td></td>
<td>c_2M==true</td>
</tr>
<tr>
<td>$M, 6 \vdash E($onemuddy==true$)</td>
<td>$(c_1M==true</td>
<td></td>
<td>c_2M==true</td>
</tr>
<tr>
<td>$M, 6 \vdash C($onemuddy==true$)</td>
<td>$(c_1M==true</td>
<td></td>
<td>c_2M==true</td>
</tr>
<tr>
<td>$M, 6 \vdash C($twomuddy==true$)</td>
<td>$(c_1M==true &amp;&amp; c_2M==true</td>
<td></td>
<td>(c_1M==true &amp;&amp; c_3M==true)</td>
</tr>
</tbody>
</table>

2: $c_1M=false$, $c_2M=true$, $c_3M=false$

4: $c_1M=true$, $c_2M=false$, $c_3M=false$

6: $c_1M=true$, $c_2M=true$, $c_3M=false$

7: $c_1M=true$, $c_2M=true$, $c_3M=true$

Figure 20 : cm4

Now the father announces again that at least one child is muddy, and asks the muddy children to come forward if they know they are muddy. To represent this announcement we insert the following constraint into the original description.

constraint ((c1M ==true) || (c2M ==true) || (c3M==true));
epistemic constraint (c1 !@ c1M==true) && (c2 !@ c2M==true) && (c3 !@ c3M==true);
The epistemic constraint above specifies the aftermath of the first announcement by the father, which resulted in no child stepping forward. It means none of the children knew they were muddy, otherwise at least one of them would have stepped forward.

We then recompile the new program to obtain the following model. We see from figure 11 (which resulted from the first announcement by the father) that if any child was muddy, then that child would have known it because it would see that no other child is muddy, knowing that at least one child is muddy. So all the children reason the same, but no child stepped forward. This means that states 1, 2 and 4 in figure 11 are in fact unrealistic, and so they are deleted to obtain the model in figure 12. At this point we see that we have common knowledge that at least two children are muddy, so any child that sees only one muddy child will step forward.

We then repeat the following model checking on the updated model.

We then repeat the following model checking on the updated model.

Table 5: Model Checking Muddy Children Puzzle – After the Second Announcement - Inputs and Outputs

<table>
<thead>
<tr>
<th>Epistemic Formular</th>
<th>vdel Construct</th>
<th>Output from Model Checker</th>
</tr>
</thead>
<tbody>
<tr>
<td>M, 6 ⊩ K_{c2}(c2M==true)</td>
<td>c1@c2M==true</td>
<td>true</td>
</tr>
<tr>
<td>M, 6 ⊩ K_{c2}(c2M==true)</td>
<td>c2@c2M==true</td>
<td>true</td>
</tr>
<tr>
<td>M, 6 ⊩ E(c2M==true)</td>
<td>#c2M==true</td>
<td>true</td>
</tr>
<tr>
<td>M, 6 ⊩ K_{c2}(onemuddy==true)</td>
<td>c2@(c1M==true</td>
<td></td>
</tr>
<tr>
<td>M, 6 ⊩ E(onemuddy==true)</td>
<td>#(c1M==true</td>
<td></td>
</tr>
<tr>
<td>M, 6 ⊩ C(onemuddy==true)</td>
<td>(c1M==true</td>
<td></td>
</tr>
<tr>
<td>M, 6 ⊩ C(twomuddy==true)</td>
<td>(c1M==true&amp;c2M==true)</td>
<td></td>
</tr>
</tbody>
</table>
4.4.2 Case Study: Consecutive Numbers Puzzle

Two agents, a (Anne) and b (Bill) are facing each other. They see a number on each other's head, and those numbers are consecutive numbers $n$ and $n+1$ for a certain $n \in \mathbb{N}$. They both know this, and they know that they know it, etc. However, they do not have any other apriori knowledge about their own number. Let us assume we have 'atoms' $a_n$ and $b_n$ for every $n \in \{0, 10\}$, expressing that the number on Anne's head equals $n$, and that on Bill's head reads $n$, respectively.

Suppose that in fact $a_3$ and $b_2$ are true. We assume that the agents only see each other's number and that it is common knowledge that the numbers are consecutive (Ditmarsch, Hoek & Kooi 2007).

VDEL Description of Consecutive Numbers

```
begin
  int a(0,11); //here we limit n to 11
  int b(0,11);

  agent anne;
  agent bill;

  constraint sqrt(sqr(a-b))=1;

  anne observes b;
  bill observes a;
end
```

Code Listing 17: vdel Description of the Initial Scenario of the Consecutive Numbers Puzzle

We compile the above program to produce the model in Figure 22.
We then proceed to the following model-checking on the current model.

Table 6: Model Checking the Initial Scenario of the Consecutive Numbers Puzzle - Inputs and Outputs

<table>
<thead>
<tr>
<th>Epistemic Formular</th>
<th>vdel Construct</th>
<th>Output from Model Checker</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M, 6 ⊧ Kanne (b==2)</td>
<td>anne@b==2</td>
<td>true</td>
</tr>
<tr>
<td>M, 6 ⊧ Kanne (a==1 V a==3)</td>
<td>anne@(a==1</td>
<td>a==3)</td>
</tr>
<tr>
<td>M, 6 ⊧ Kanne K_bill (b==0 V b==2 V b==4)</td>
<td>anne@bill@b==0</td>
<td>b==2</td>
</tr>
<tr>
<td>M, 6 ⊧ K_bill Kanne (a==1 V a==3 V a==5)</td>
<td>bill@anne@(a==1</td>
<td>a==3</td>
</tr>
</tbody>
</table>

Figure 22: Generated Model of the Consecutive Numbers Puzzle
| M, 6 ⊨ Kanne K_{b111}K_{anne} (b==0 V b==2 V b==4) | anne@bill@anne@(b==0 | b==2 | b==4) | true | Figure 23: cm5 |
| M, 6 ⊨ K_{b111}K_{anne}K_{b111}(a==1 V a==3 V a==5) | bill@anne@bill@(a==1 | a==3 | a==5) | true | Figure 23: cm5 |
| M, 6 ⊨ ¬K_{anne}¬(a==1) ^ ¬K_{anne}¬(a==3) | (anne!@!(a==1))&(anne!@!(a==3)) | true |
| M, 6 ⊨ ¬K_{b111}¬(b==2) ^ ¬K_{b111}¬(b==4) | (bill!@!(b==2))&(bill!@!(b==4)) | true |
| M, 6 ⊨ K_{anne}(a) | anne@[a] | false |
| M, 6 ⊨ K_{b111}(b) | bill@[b] | false |
| M, 6 ⊨ K_{anne}(¬K_{anne} (a) ^ ¬K_{b111}(b)) | anne@(anne!@[a]&&bill@[b]) | true |
| M, 6 ⊨ K_{b111}(¬K_{anne} (a) ^ ¬K_{b111}(b)) | bill@(anne!@[a]&&bill@[b]) | true |
| M, 6 ⊨ E¬(a==5 ^ ¬EE¬(a==5)) | (#(a==5))&(¬(#(a==5))) | true |
| M, 6 ⊨ E¬K_{anne} (a) ^ ¬K_{b111}(b) ^ ¬EE¬K_{anne} (a) ^ ¬K_{b111}(b)) | (anne!@[a]&&bill@[b])&&(¬((anne!@[a]&&bill@[b]))) | true |
| M, 2 ⊨ K_{anne}(a==1) ^ K_{anne}(a V K_{b111}(b)) | (anne@a==1)(&(anne@anne[a] | bill[b])) | true |
| M ⊨ (a==3 ^ b==2) -> ¬C¬(a==1 V a==2 V a==3 V a==5) ^ C(b==1 V b==2 V b==3 V b==5) | ((a==3&b==2) -> !(a==1 | a==2 | a==3 | a==5)) C | false | Figure 23: cm5 |
| | (a==3&b==2) -> (b==1 | b==2 | b==3 | b==5) C | true |
Epistemic Updates in the Consecutive Numbers Puzzle

From the initial scenario (figure 13) we see that if in the real world anne had a ‘0’ or ‘11’ on her forehead, then bill would have known his number. Likewise if bill had a ‘0’ or ‘11’, anne too would have known her number. However in this puzzle we witness the unfolding drama in which both anne and bill continue to repeat the announcement “I don’t know my number”. Let us use vdel to investigate what happens after each announcement.

bill says “I don’t know my number”. So we append the following epistemic constraint to the vdel code:

```
bill ![a];
```

bill says he does not know his number. Anne reasons that if bill had seen a ‘0’ or ‘11’ on her, he would have known his number. anne concludes that the states in which she has ‘0’ or ‘11’ are unrealistic, but still does not know her number. As such states 1 and 22 are deleted from the model for being unrealistic, and so we obtain the updated model shown in Figure 23.
Next *anne* says “I don’t know my number”. We again append the following epistemic constraint to the vdel code:

\[ \text{anne} !@ [a]; \]

In the updated model, if *anne* saw ‘0’ or ‘11’ on *bill* forehead, she would have known her number. In addition, since the world in which *anne* has ‘0’ and *bill* ‘11’ was invalidated, if *anne* then saw ‘10’ on *bill*, she would have known her number (since she would know her number could then only be ‘9’). Likewise at this point, if *anne* saw ‘1’ on *bill’s* forehead, she would know her number could only be ‘2’, since \( a=0 \) had been previously invalidated. *bill* also reasons about these deliberations of *anne*. So they conclude that states 2, 4, 19 and 21 are unrealistic, and we remove those from the model to obtain an updated model (Figure 24). Again we verify this reasoning on the graphical model when we observe in states 2, 4, 19 and 21 of figure 14 that in fact *anne* (black accessibility relation) would have known her since all the *anne*-successor states would be indistinguishable to *anne* in those states.
Figure 25: Consecutive Numbers Puzzle after the Second Announcement

Bill says again “I don’t know my number”. We append

```
bill !@ [a];
```

to the vdel code, and following the same reasoning above we delete states 3, 5, 18 and 20 from the model in Figure 25 to obtain the following updated model (Figure 26).

Figure 26: Consecutive Numbers Puzzle after the Third Announcement
We see that by continuous and alternating announcement of their ignorance, both anne and bill are gradually approaching a scenario in which they both know their number. We see in Figure 26 that anne now knows her number!

### 4.4.3 Solving the Sum and Product Puzzle

We conclude by giving a summary of how the sum and product puzzle is modeled in vdel, and by introducing the right order of epistemic constrains, we generate a model which shows the solution of the puzzle. We will omit the detailed graphical models of the initial scenarios, since they are too large to be included here.

A says to Sum and Prod: “I have chosen two natural numbers x and y such that 1<x<y and x + y ≤ 100. I am now going to announce their sum s= x + y to Sum only, and their product p = x.y to Prod only. The content of these announcements remains a secret.” He acts accordingly. The following conversation between Sum and Prod then takes place.

1. Prod says: “I don’t know the numbers.”
2. Sum: “I knew that. I don’t know my number.”
3. Prod: “Now I know my number.”
4. Sum: “Now I know my number.”

**vdel Description:**

begin

int x(2,99);  
int y(3,99);

agent sum;  
agent prod;

constraint x<y;  
constraint x+y<=100;  
dynamic constraint s = x+y;  
dynamic constraint p = x*y;

sum observes s;  
prod observes p;

epistemic constraint (prod !@ [x && y]) && (sum @ (prod !@ [x && y]));

epistemic constraint sum !@ [x && y];  
epistemic constraint prod @ [x && y];  
epistemic constraint sum @ [x && y];

end

**Code Listing 18 : vdel Description of the Sum and Product Puzzle with all the announcements included**
In the first instance (first epistemic constraint in the vdel description) prod announces “I don’t know my number”. In the second instance, sum begins by announcing “I knew it”, which implies that sum knew prod did not know x and y in the first instance. This is why the first epistemic constraint in the vdel description says that “prod does not know x and y, and sum knows that prod does not know x and y”. Hence, giving the epistemic constraints in the right order is of vital importance in obtaining the correct model following the announcements.

When we compile the above description we obtain the following model which shows the unique solution of the puzzle.

![Figure 27: Unique Solution to the Sum and Product Puzzle](image-url)

199: p=52.0, s=17.0, x=4.0, y=13.0
EVALUATION

Already we have carried out extensive evaluation of the vdel system in previous chapters, especially in chapter 4, by using the system to model, test and study some well-known epistemic scenarios such as the Muddy Children Puzzle, Consecutive Numbers puzzle and Sum and Product puzzle. We see that the implemented tool satisfies all the test cases proposed for use in evaluating this tool according to project proposal and specification. Furthermore vdel was also used to test some other test book dynamic epistemic scenarios that were not included in the project proposal, and the tests were equally successful. Some of these other scenarios include the Prisoners and Hats Puzzle and the Game of Hexa.

In this chapter we present further analysis and evaluation of the vdel system by comparing it briefly with the current state of the art dynamic epistemic modelling tool, DEMO. We will highlight some of their similarities and differences, and outline some of the strengths of both tools.

5.1 DEMO vs. VDEL

In the following subsections we present comparison of DEMO and vdel under broad categories of general functionality, language for epistemic and dynamic epistemic description, some model visualisation capabilities, time complexity of model checking, and other general implementation differences.

5.1.1 General Functionality

Both DEMO and vdel provides facilities for modelling of epistemic scenarios and epistemic updates, generating graphical models of DEL scenarios, facilities for formular evaluation in epistemic models, and model-checking of generated models.

DEMO, which is currently a much more mature DEL modelling tool provides the following additional features: translation of dynamic epistemic formulas to PDL formulas, reduction of dynamic epistemic logic to automata PDL, minimisation of epistemic models under dissimulation, minimization of action models under action emulation.

5.1.2 Epistemic Description

In DEMO, description of epistemic scenarios is done in Haskell, a popular logic programming language. However vdel comes with its own DEL description language which is designed specifically for describing dynamic epistemic logic scenarios. As such vdel often makes for a more succinct and intuitive description of scenarios and public announcements than DEMO.
However, concerning announcements, whereas vdel only implements public announcements (aka Message passing with common knowledge) in which a message is sent to every one of the agents in the scenario, and they also know of each other’s knowledge of the message, in addition DEMO implements more specialized message passing schemes such as individual message passing in which only a proper subset of the agents in the scenario receive an actual message although all agents know of the existence of the message; public message passing in which all the agents in the scenario receive the message but they do not know whether the same message has been passed to others. Secret message passing, non-other than the receiving agent knows that any message passing is going on.

5.1.3 Model Visualisation

Due to the specialized message passing schemes in DEMO, models classes generated in DEMO could be more varied. Whereas vdel generates only S5 models, DEMO could in addition, generate KD45 and K45 models.

In addition to the graphic visualization of models, DEMO also provides textual models following the definitions given in chapter 2. DEMO also allows several views and utility functions on generated models such as checking euclidicity, reflexivity equivalence, finding axiom schemes for logic of communication etc. At the moments, vdel only provides the list of all the states in the model together with their valuation.

Both vdel and DEMO do not yet support the functionality of letting a user to graphically interact with and modify generated graphical models.

5.1.4 Model Checking and Time Complexity

The model checking algorithm implemented by vdel is based on the well-known labelling algorithm (Fagin et al. 1995; Halpern & Vardi 1991). Model checking problem is defined as: given a model $M$, a state $w$, and a formula $\varphi$, check whether $M,w \vDash \varphi$, where $|M|$ is the size of the model as measured in the size of its domain plus the number of pairs in its accessibility relations, and where $|\varphi|$ is the length of the formula $\varphi$. In the labelling algorithm we begin by dividing $\varphi$ into subformulas of which there are at most $|\varphi|$, and placing them in order of increasing length. For each subformula, all states are labelled with either the $\varphi$ formula or its negation, according to the valuation of the model and based on the results of previous steps. This is a bottom-up approach in the sense that the labelling starts from the smallest subformulas. The approach yields a time complexity of $O(|M| \times |\varphi|)$.

In DEMO (v1.02), the algorithm for checking whether $M,w \vDash \varphi$ does not employ the bottom-up approach. Instead it uses a top-down approach, starting with the formula $\varphi$ and recursively checking its largest subformulas. For instance, in checking whether $M,w \vDash K_a \psi$, the algorithm first checks whether $M,w' \vDash \psi$ for all $w'$ such that $w \sim_a w'$, and then recursively checks the subformulas of $\psi$. This algorithm works in time $O(|M|^\varphi)$, since each
subformula may need to be checked $|M|$ times, and there are at most $|\varphi|$ subformulas of $\varphi$. Therefore DEMO’s algorithm is quite expensive, in the worst case.

vdel, follows the bottom-up approach of subformular evaluation, and as such matches the time complexity of $O(|M| \times |\varphi|)$, outperforming DEMO. In addition, vdel’s model-checking algorithm was optimised such that for individual knowledge operators, only the accessibility relations pertaining to the agent in question is considered in the search. This improves the efficiency of the said “bottom-up” model-checking algorithm. However, overall complexity of model-checking in vdel is yet to be determined.

5.1.5 General implementation differences

We see that DEMO leverages the already in-built logical utility functions in Haskell, as such DEMO has much more powerful utility functions than the current version of vdel. But we have also seen that building on Haskell compels a user to describe epistemic scenarios in Haskell, which was not designed with dynamic epistemic logic in mind.

Furthermore, due to DEMO’s dependence on Haskell, the description of epistemic scenarios takes on a mathematical-logical form, whereas vdel provides a programming language formalism which combines elements of object oriented programming with those of logic programming.

We also see that vdel provides a GUI model-checking tool called the Epistemic Keypad. This keypad presents the user with an immediate overview of the available functions on the model checker, as well as the views of the graphical model and graphical counter examples (in the case where the result of model-checking is false). DEMO is strictly a command-line model-checking tool and does not yet provide a graphical counter example where the checked property is false.

Therefore whereas DEMO currently supports many more utilities for research in dynamic epistemic logic, vdel currently provides a useful tool for beginners in dynamic epistemic logic to appreciate and understand modelling and verification of properties in dynamic epistemic scenarios.

Finally both DEMO and vdel rely on the graph-visualisation tool, dot, to render the generated model in graphical form.

5.2 Summary of Strengths, Weaknesses and Recommendations

Compared to the state-of-the-art modeling tool, DEMO, we see that the vdel tool is still budding. However, vdel possesses some strength over DEMO since vdel description language was implemented specifically with epistemic and dynamic epistemic scenarios in mind, whereas DEMO uses Haskell, which is a general purpose logic programming language, to describe epistemic scenarios. Hence vdel constructs are sometimes more intuitive than those in DEMO in describing epistemic scenarios.
In addition vdel’s time complexity also outperforms that of DEMO since it follows (and optimizes for multiagent DEL) the “bottom-up” labeling algorithm in (Fagin et al. 1995; Halpern & Vardi 1991).

Finally vdel is only capable of handling public announcements and generating $\mathcal{S}5$ model only. DEMO is much more robust here, in that it handles a variety message passing mechanisms and generates other classes of models apart from $\mathcal{S}5$. In addition DEMO possesses more utility functions for viewing different characteristics of generated models. vdel needs to implement more utility functions along this line.
PROFESSIONAL ISSUES

In this chapter we discuss some relevant professional and ethical issues considered carrying out this project. We reflect upon the British Computer Society’s Code of Conduct and Code of Practice (British Computer Society. 1976), in relation to the development of the inputs, outputs and execution of this project, as well as considerations of the applications and uses of the project’s outcomes.

6.1 Code of Conduct

This project complies with the British Computer Society (BCS) Code of Conduct. Following the section on public interest, this project has been carried out with due diligence and care for the specifications laid down for carrying out of masters project by the Department of Computer Science, University of Liverpool. Specifically, this project has been carried out in such a way as to accomplish all the set tasks laid out in the project specification, to the level of detail permitted by the available time of completion for the project. Furthermore, although this project was carried out with the intention of creating a useful tool for the visualisation of dynamic epistemic logic, it is not necessarily intended that the output of this project be made publicly available to general users. However, in carrying out this project, it was carefully designed and implemented in such a way as to cater for potential users such as students and researchers in the area of epistemic logic and dynamic epistemic logic.

Bearing the end users in mind, the following special features were incorporated to better serve their interests:

The system was built on a platform (Java) which is easily accessible to both all students and researchers, making it easy for users to easily install and use the tool. The vdel description language was also designed to be intuitive to use, which makes it easy to use for newcomers, as well as advanced researchers in epistemic and dynamic epistemic logic. Intuitive graphical user interfaces were also provided with adequate colour cues and accessible textual displays which allows for disabled users to access the controls of the programme. The graphical user interface was also designed to expose all the functions of the underlying system, make it a powerful tool, no less than the command line interface which constitutes the state-of-the-art prior to this project.

With respect to environmental conservation, we have seen that the time complexity of vdel model checker is better than that of the currently available DEL modelling tool, DEMO. As such more CPU resources will be saved in the event that this tool becomes publicly available for use by both students and researchers, with the consequence of reduction in carbon emissions and energy conservation.

Results presented in this project were honest, and at no stage were the results manipulated for any reason. This project was also carried out in full compliance with Copyright, Designs and
Patents Act (1988), the Computer Misuse Act (1990) and Data Protection Act (1998). In addition, all known national, regional or international laws were complied with. No personal data was stored by this project. This project was also not done under any form of duress whatsoever.

6.2 Code of Good Practice

This project also complies with the BCS Code of Good Practice. Throughout the design and execution of this project, the author’s technical skills and knowledge were greatly enhanced. Particularly the author became more knowledgeable, and gained a better understanding of epistemic and dynamic epistemic logic. In addition, the author gained deeper understanding of the theory and practice of the design of programming languages which he greatly utilised in the design of the vdel language and the vdel interpreter. The author deepened his understanding of model checkers and model checking algorithms, as well as state-of-the-art tools in dynamic epistemic logic modelling and verification. The author’s writing and presentation skills were also greatly enhanced throughout the course of this project. Finally, the author feels a greater confidence in undertaking a tutoring and demonstration role in modules that relate with epistemic logic and dynamic epistemic logic.

The author ensured that the level of his ability in specific areas were indicated prior to crucial stages of the project, and put effort to acquire the requisite skills and knowledge through enlightening discussions with project advisors, and the following of advice provided in the project meetings and discussions. Recommended readings by the advisors were also done diligently and progress reports made and followed by project advisors. The author, together with the project advisors also ensured that the workload undertaken was reasonable for the ability of the author, and that the author had the necessary resources to complete all the work within the specified time period for the project.

This research is in no way damaging to society. It is, on the contrary, very beneficial to students in the study of computer science and artificial intelligence. In addition, the potential applications have been made clear for this project. Furthermore due to the fact that no personal data was employed in carrying out this project, all ethical and legal concerns in the area of personal data protection and privacy have also been complied with and respected. Finally work by other authors such as DEMO, the current state-of-the-art tool for dynamic epistemic modelling have also been evaluated in this project to provide a more complete view of the impact of this project so far within the research field.
CONCLUSIONS

7.1 Summary of Main Outcomes

In this project we have created a tool for modelling and visualising dynamic epistemic logic. The vdel tool allows us to intuitively model dynamic epistemic logic scenarios, and verify epistemic properties on these scenarios using the vdel model-checking tool. Visualisations of initial model of scenarios, updated models, and counter examples thus allow us to see the dynamics and solutions of epistemic scenarios and puzzles.

The vdel description language is intuitive in the sense that introduces a programming language specifically designed for describing dynamic epistemic logic scenarios; the vdel interpreter translates the vdel description into a Kripke model; epistemic properties are also expressed in vdel, and checked on the generated model, using the vdel model-checker. The epistemic keypad provides a user-friendly GUI for interacting with the vdel model-checker.

7.2 Future Work

In vdel the current algorithm for evaluating the common knowledge operator assumes that the model is a single connected graph. As a result, there is no symbol and grammar production for common knowledge. Instead common knowledge operation defaults to an exhaustive checking of the underlying epistemic property in all states of the model.

This approach places a limitation in checking common knowledge properties on models that have more than one connected components, for instance the consecutive numbers puzzle.

As an example, consider a property of the form shown below:

\[ M, s \models C\psi \]

Property \( C\psi \) (above) may be true in the connected component that contains state, \( s \), but not true for the entire model. The existence of models with more than one connected component brings us to the question of reachability. Given an epistemic states, \( s_1 \), is it possible to reach another state, \( s_2 \), from \( s_1 \). What if \( s_1 \) and \( s_2 \) are in different connected components of the model?

Furthermore, we have seen how an announcement, or a sequence of well-ordered announcements, led to a goal epistemic state (as seen already in, for example, the Consecutive Numbers, and Sum and Product Puzzles). We might know for sure that state \( s_2 \) is reachable from state \( s_1 \), but we may not know exactly the sequence of announcements that will take us to the goal state. This brings us to the problem of finding a practicable epistemic protocol.

The vdel system could form the basis of further research into the area of dynamic epistemic model-checker that can provide us with epistemic protocols by which a goal state could be reached.
REFERENCES CITED


Hintikka, J. (1957) 'Modality as Referential Multiplicity', *Ajatus*, vol. 20, pp. 49-64.


APPENDIX A: BNF GRAMMAR FOR VDEL DESCRIPTION LANGUAGE

goal ::= BEGIN del_code END;

del_code ::= atoms agents facts;

atoms ::= atom_declaration_list;
atom_declaration_list ::= atom_declaration_list atom_declaration
| atom_declaration;

atom_declaration ::= bool_declaration SEMI
| int_declaration SEMI
| enum_declaration SEMI;

bool_declaration ::= BOOLEAN ATOM_IDENTIFIER;

int_declaration ::= INT ATOM_IDENTIFIER LPAREN DOUBLE COMMA DOUBLE RPAREN;

enum_declaration ::= ENUM ATOM_IDENTIFIER LPAREN enum_values_list RPAREN;

enum_values_list ::= enum_values_list COMMA enum_value
| enum_value;

enum_value ::= IDENTIFIER;

agents ::= agent_declaration_list;
agent_declaration_list ::= agent_declaration_list agent_declaration
| agent_declaration;

agent_declaration ::= AGENT AGENT_IDENTIFIER SEMI;

facts ::= fact_list;
fact_list ::= fact_list fact
| fact;

fact ::= epistemic_constraint
| propositional_constraint;

epistemic_constraint ::= EPISTEMIC_KW CONSTRAINT_KW epistemic_property SEMI
| IDENTIFIER OBSERVES IDENTIFIER SEMI;

epistemic_property ::= boolean_expr AND epistemic_fact
| boolean_expr OR epistemic_fact
| boolean_expr IMPLIES epistemic_fact
| epistemic_fact;

epistemic_fact ::= IDENTIFIER KNOWS epistemic_fact
| IDENTIFIER NOT KNOWS epistemic_fact
| epistemic_fact AND epistemic_fact
| epistemic_fact OR epistemic_fact
| epistemic_fact IMPLIES epistemic_fact
| LPAREN epistemic_fact RPAREN
| IDENTIFIER KNOWS boolean_expr
| IDENTIFIER NOT KNOWS boolean_expr
| IDENTIFIER KNOWS LBRACKET phi RBRACKET
| IDENTIFIER NOT KNOWS LBRACKET phi RBRACKET
| EVERYBODYKNOWS epistemic_fact
| NOT EVERYBODYKNOWS epistemic_fact
| EVERYBODYKNOWS boolean_expr
| NOT EVERYBODYKNOWS boolean_expr
| EVERYBODYKNOWS LBRACKET phi RBRACKET
| NOT EVERYBODYKNOWS LBRACKET phi RBRACKET
| DISTKNOWLEDGE epistemic_fact
| NOT DISTKNOWLEDGE epistemic_fact
| DISTKNOWLEDGE boolean_expr
| NOT DISTKNOWLEDGE boolean_expr
| DISTKNOWLEDGE LBRACKET phi RBRACKET
| NOT DISTKNOWLEDGE LBRACKET phi RBRACKET;

63
\[
\phi ::= \phi \text{ AND } \phi \\
| \phi \text{ OR } \phi \\
| \phi \text{ IMPLIES } \phi \\
| \text{NOT } \phi \\
| \text{IDENTIFIER} \\
| \text{LPAREN } \phi \text{ RPAREN};
\]

\[
\text{propositional_constraint ::= static_constraint} \\
| \text{dynamic_constraint} ;
\]

\[
\text{static_constraint ::= CONSTRAINT_KW boolean_expr SEMI;} \\
\text{dynamic_constraint ::= DYNAMIC_KW CONSTRAINT_KW IDENTIFIER EQUALS expr SEMI} \\
| \text{DYNAMIC_KW CONSTRAINT_KW IDENTIFIER EQUALS boolean_expr SEMI};
\]

\[
\text{boolean_expr ::= boolean_expr AND boolean_expr} \\
| \text{boolean_expr OR boolean_expr} \\
| \text{boolean_expr IMPLIES boolean_expr} \\
| \text{expr DOUBLEEQUALS expr} \\
| \text{expr DOUBLEEQUALS boolean_expr} \\
| \text{expr LESSTHAN expr} \\
| \text{expr LESSTHANEQUAL expr} \\
| \text{expr GREATERTHAN expr} \\
| \text{expr GREATERTHANEQUAL expr} \\
| \text{expr NOTEQUAL expr} \\
| \text{expr NOTEQUAL boolean_expr} \\
| \text{NOT boolean_expr} \\
| \text{TRUE} \\
| \text{FALSE} \\
| \text{LPAREN boolean_expr RPAREN};
\]

\[
\text{expr ::= expr PLUS term} \\
| \text{expr MINUS term} \\
| \text{term};
\]

\[
\text{term ::= term MULTIPLY unary_expr} \\
| \text{term DIVIDE unary_expr} \\
| \text{term MOD unary_expr} \\
| \text{unary_expr};
\]

\[
\text{unary_expr ::= SQUAREROOT unary_expr} \\
| \text{SQUARE unary_expr} \\
| \text{leaf};
\]

\[
\text{leaf ::= IDENTIFIER} \\
| \text{DOUBLE} \\
| \text{LPAREN expr RPAREN};
\]
APPENDIX B: BNF GRAMMAR FOR THE ARITHMETIC AND BOOLEAN LOGIC EVALUATOR (ABLE)

statement ::= boolean_expr
  | arithmetic_expr;

boolean_expr ::= boolean_expr AND boolean_expr
  | boolean_expr OR boolean_expr
  | boolean_expr IMPLIES boolean_expr
  | arithmetic_expr DOUBLEEQUALS arithmetic_expr
  | arithmetic_expr DOUBLEEQUALS boolean_expr
  | arithmetic_expr LESSTHANEQUAL arithmetic_expr
  | arithmetic_expr LESSTHANEQUAL boolean_expr
  | arithmetic_expr GREATERTHAN arithmetic_expr
  | arithmetic_expr GREATERTHANEQUAL arithmetic_expr
  | arithmetic_expr NOTEQUAL arithmetic_expr
  | arithmetic_expr NOTEQUAL boolean_expr
  | NOT boolean_expr
  | TRUE
  | FALSE
  | LPAREN boolean_expr RPAREN;

arithmetic_expr ::= arithmetic_expr PLUS term
  | arithmetic_expr MINUS term
  | term;

term ::= term MULTIPLY unary_expr
  | term DIVIDE unary_expr
  | unary_expr;

unary_expr ::= SQUAREROOT unary_expr
  | SQUARE unary_expr
  | leaf;

leaf ::= IDENTIFIER
  | INTEGER
  | LPAREN arithmetic_expr RPAREN;
APPENDIX C: BNF Grammar for the Epistemic Property Evaluator

statement ::= epistemic_expr SEMI
| boolean_expr SEMI
| boolean_expr AND epistemic_expr SEMI
| boolean_expr OR epistemic_expr SEMI
| boolean_expr IMPLIES epistemic_expr SEMI;

epistemic_expr ::= IDENTIFIER KNOWS epistemic_expr
| IDENTIFIER NOT KNOWS epistemic_expr
| epistemic_expr AND epistemic_expr
| epistemic_expr OR epistemic_expr
| epistemic_expr IMPLIES epistemic_expr
| LPAREN epistemic_expr RPAREN
| IDENTIFIER KNOWS boolean_expr
| IDENTIFIER NOT KNOWS boolean_expr
| IDENTIFIER KNOWS BRACKET phi BRACKET
| IDENTIFIER NOT KNOWS BRACKET phi BRACKET
| EVERYBODYKNOWS epistemic_expr
| NOT EVERYBODYKNOWS epistemic_expr
| EVERYBODYKNOWS boolean_expr
| NOT EVERYBODYKNOWS boolean_expr
| EVERYBODYKNOWS BRACKET phi BRACKET
| NOT EVERYBODYKNOWS BRACKET phi BRACKET
| DISTKNOWLEDGE epistemic_expr
| NOT DISTKNOWLEDGE epistemic_expr
| DISTKNOWLEDGE boolean_expr
| NOT DISTKNOWLEDGE boolean_expr
| DISTKNOWLEDGE BRACKET phi BRACKET
| NOT DISTKNOWLEDGE BRACKET phi BRACKET;

phi ::= phi AND phi
| phi OR phi
| phi IMPLIES phi
| NOT phi
| IDENTIFIER
| LPAREN phi RPAREN;

boolean_expr ::= boolean_expr AND boolean_expr
| boolean_expr OR boolean_expr
| boolean_expr IMPLIES boolean_expr
| arithmetic_expr DOUBLEEQUALS arithmetic_expr
| arithmetic_expr DOUBLEEQUALS boolean_expr
| arithmetic_expr LESSTHAN arithmetic_expr
| arithmetic_expr LESSTHANEQUAL arithmetic_expr
| arithmetic_expr GREATERTHAN arithmetic_expr
| arithmetic_expr GREATERTHANEQUAL arithmetic_expr
| arithmetic_expr NOTEQUAL arithmetic_expr
| arithmetic_expr NOTEQUAL boolean_expr
| NOT boolean_expr
| TRUE
| FALSE
| LPAREN boolean_expr RPAREN;

arithmetic_expr ::= arithmetic_expr PLUS term
| arithmetic_expr MINUS term
| term;

term ::= term MULTIPLY unary_expr
| term DIVIDE unary_expr
| term MOD unary_expr
| unary_expr;

unary_expr ::= SQUAREROOT unary_expr
| SQUARE unary_expr
| leaf;

leaf ::= IDENTIFIER
| DOUBLE
| LPAREN arithmetic_expr RPAREN;
## APPENDIX D: PROJECT PLAN

<table>
<thead>
<tr>
<th>Start Date</th>
<th>End Date</th>
<th>Duration (weeks)</th>
<th>Task</th>
<th>Deliverables and Milestones</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 1</td>
<td>July 29</td>
<td>8</td>
<td>Background Research</td>
<td></td>
</tr>
<tr>
<td>June 22</td>
<td>June 29</td>
<td>1</td>
<td><strong>Writing of Project Specification</strong></td>
<td><strong>Project Specification Document</strong></td>
</tr>
<tr>
<td>June 29</td>
<td>June 29</td>
<td>-</td>
<td>Submission of Specification</td>
<td></td>
</tr>
<tr>
<td>June 1</td>
<td>July 6</td>
<td>5</td>
<td><strong>Software Design and Prototyping</strong></td>
<td><strong>Description Language</strong></td>
</tr>
<tr>
<td>June 4</td>
<td>June 11</td>
<td>1</td>
<td>Construction of Description Language grammar for the DEL Puzzles</td>
<td>Description Language grammar Design</td>
</tr>
<tr>
<td>June 14</td>
<td>June 28</td>
<td>2</td>
<td>Construction of language grammar for the property checker and for public announcements</td>
<td>Language grammar for the property checker and public announcements</td>
</tr>
<tr>
<td>June 14</td>
<td>June 16</td>
<td>-</td>
<td>Design of the “Epistemic Keypad”</td>
<td>“Epistemic Keypad” Design</td>
</tr>
<tr>
<td>June 30</td>
<td>July 7</td>
<td>1</td>
<td>Design of Algorithms for the DEL Property Checker</td>
<td>DEL Property Checker Algorithms</td>
</tr>
<tr>
<td>June 16</td>
<td>July 22</td>
<td>4</td>
<td><strong>Documentation of Design</strong></td>
<td><strong>Design Documentation</strong></td>
</tr>
<tr>
<td>July 20</td>
<td>July 25</td>
<td>1</td>
<td>Preparation for Design Presentation</td>
<td></td>
</tr>
<tr>
<td>July 25</td>
<td>July 25</td>
<td>-</td>
<td><strong>Design Presentation</strong></td>
<td></td>
</tr>
<tr>
<td>June 1</td>
<td>August 20</td>
<td>7</td>
<td><strong>Software Implementation</strong></td>
<td></td>
</tr>
<tr>
<td>June 6</td>
<td>July 18</td>
<td>6</td>
<td>Implementation of Interpreter for the Description Language</td>
<td>Description-Language-to-Graphical Interpreter</td>
</tr>
<tr>
<td>July 11</td>
<td>July 27</td>
<td>2</td>
<td>Implementation of the “Epistemic Keypad” and DEL Property Checker</td>
<td>“Epistemic Keypad” and DEL Property-checker</td>
</tr>
<tr>
<td>July 27</td>
<td>August 20</td>
<td>3</td>
<td>Integration of Software Components and testing</td>
<td><strong>Integrated Software: DEL Visualiser</strong></td>
</tr>
<tr>
<td>August 20</td>
<td>August 27</td>
<td>1</td>
<td>Preparation for Software Presentation</td>
<td></td>
</tr>
<tr>
<td>August 27</td>
<td>August 27</td>
<td>-</td>
<td><strong>Presentation of Software</strong></td>
<td></td>
</tr>
<tr>
<td>August 1</td>
<td>August 14</td>
<td>2</td>
<td>Experiments with some epistemic puzzles</td>
<td></td>
</tr>
<tr>
<td>August 1</td>
<td>August 8</td>
<td>1</td>
<td>Testing the 2-agent-2-atom scenario</td>
<td></td>
</tr>
</tbody>
</table>

68
<table>
<thead>
<tr>
<th>Date</th>
<th>Date</th>
<th>Duration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>August 9</td>
<td>August 16</td>
<td>1</td>
<td>Testing the Muddy Children Scenario</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>and Sum and Product or Consecutive Numbers scenario</td>
</tr>
<tr>
<td>August 1</td>
<td>September 12</td>
<td>5</td>
<td>Dissertation Write-up</td>
</tr>
<tr>
<td>August 1</td>
<td>August 3</td>
<td>-</td>
<td>Introduction</td>
</tr>
<tr>
<td>August 3</td>
<td>August 10</td>
<td>1</td>
<td>Background</td>
</tr>
<tr>
<td>August 10</td>
<td>August 17</td>
<td>1</td>
<td>Methodology</td>
</tr>
<tr>
<td>August 17</td>
<td>August 20</td>
<td>0.5</td>
<td>Realisation</td>
</tr>
<tr>
<td>August 20</td>
<td>August 30</td>
<td>1.5</td>
<td>Evaluation, Conclusion and Abstract</td>
</tr>
<tr>
<td>August 30</td>
<td>August 31</td>
<td></td>
<td>Professional Issues</td>
</tr>
<tr>
<td>September 1</td>
<td>September 12</td>
<td></td>
<td>Proof-reading of Dissertation</td>
</tr>
<tr>
<td>September 12</td>
<td>September 14</td>
<td></td>
<td>Project Completion and Submission of Dissertation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dissertation Document</td>
</tr>
</tbody>
</table>