EVOLVING KNOWLEDGE BASES
Frontiers in Artificial Intelligence and Applications

Volume 81

Published in the subseries

Dissertations in Artificial Intelligence

Under the Editorship of the ECCAI Dissertation Board
Proposing Board Member: Luís Moniz Pereira

Recently published in this series:

Vol. 80, T. Welzer et al. (Eds.), Knowledge-based Software Engineering
Vol. 79, H. Motoda (Ed.), Active Mining
Vol. 78, T. Vidal and P. Liberatore (Eds.), STAIRS 2002
Vol. 77, F. van Harmelen (Ed.), ECAI 2002
Vol. 76, P. Šinčák et al. (Eds.), Intelligent Technologies – Theory and Applications
Vol. 75, I.F. Cruz et al. (Eds.), The Emerging Semantic Web
Vol. 74, M. Blay-Fornarino et al. (Eds.), Cooperative Systems Design
Vol. 73, H. Kangassalo et al. (Eds.), Information Modelling and Knowledge Bases XIII
Vol. 72, A. Namatame et al. (Eds.), Agent-Based Approaches in Economic and Social Complex Systems
Vol. 71, J.M. Abe and J.I. da Silva Filho (Eds.), Logic, Artificial Intelligence and Robotics
Vol. 70, B. Verheij et al. (Eds.), Legal Knowledge and Information Systems
Vol. 69, N. Baba et al. (Eds.), Knowledge-Based Intelligent Information Engineering Systems & Allied Technologies
Vol. 68, J.D. Moore et al. (Eds.), Artificial Intelligence in Education
Vol. 67, H. Jaakkola et al. (Eds.), Information Modelling and Knowledge Bases XII
Vol. 66, H.H. Lund et al. (Eds.), Seventh Scandinavian Conference on Artificial Intelligence
Vol. 65, In production
Vol. 64, J. Breuker et al. (Eds.), Legal Knowledge and Information Systems
Vol. 63, I. Gent et al. (Eds.), SAT2000
Vol. 62, T. Hruška and M. Hashimoto (Eds.), Knowledge-Based Software Engineering
Vol. 61, E. Kawaguchi et al. (Eds.), Information Modelling and Knowledge Bases XI
Vol. 60, P. Hoffman and D. Lemke (Eds.), Teaching and Learning in a Network World
Vol. 59, M. Mohammadian (Ed.), Advances in Intelligent Systems: Theory and Applications
Vol. 58, R. Dieng et al. (Eds.), Designing Cooperative Systems
Vol. 57, M. Mohammadian (Ed.), New Frontiers in Computational Intelligence and its Applications
Vol. 56, M.I. Torres and A. Sanfeliu (Eds.), Pattern Recognition and Applications
Vol. 55, G. Cumming et al. (Eds.), Advanced Research in Computers and Communications in Education
Vol. 54, W. Horn (Ed.), ECAI 2000
Vol. 53, E. Motta, Reusable Components for Knowledge Modelling
Vol. 52, In production
Vol. 51, H. Jaakkola et al. (Eds.), Information Modelling and Knowledge Bases X
Vol. 50, S.P. Lajoie and M. Vivet (Eds.), Artificial Intelligence in Education
Vol. 49, P. McNamara and H. Prakken (Eds.), Norms, Logics and Information Systems
Vol. 48, P. Návrat and H. Ueno (Eds.), Knowledge-Based Software Engineering
Vol. 47, M.T. Escrig and F. Toledo, Qualitative Spatial Reasoning: Theory and Practice
Vol. 46, N. Guarino (Ed.), Formal Ontology in Information Systems
Vol. 45, P.-J. Charrel et al. (Eds.), Information Modelling and Knowledge Bases IX
Vol. 44, K. de Koning, Model-Based Reasoning about Learner Behaviour

ISSN: 0922-6389
Evolving Knowledge Bases

Specification and Semantics

João Alexandre Leite
Centro de Inteligência Artificial (CENTRIA), Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Caparica, Portugal
Acknowledgements

I must start by thanking Ana for being with me and everything else..., and my parents, as ever, for being the best parents in the world.

On the academic side, I would like to start by thanking my supervisor, Prof. Dr. Luís Moniz Pereira, for everything he has provided me with, during these unforgettable years. His almost full devotion to the academic and scientific causes constitutes an example filled with fruitful lessons to be learnt. His leadership has played the fundamental role in bringing our research group to where it stands now. Thank you for everything you taught me.

A special acknowledgment must go to Prof. Dr. José Júlio Alferes for his collaboration and joint work. His presence, support, and insightful comments, were always of crucial importance.

I would also like to thank both Prof. Dr. Luís Moniz Pereira and Prof. Dr. José Júlio Alferes for sharply proofreading a draft of this work.

For valuable discussions and joint work throughout these years, I would also like to thank Prof. Dr. Antonio Brogi, Dr. Pierangelo Dell’Acqua, Prof. Dr. Teodor Przymusinski and Prof. Dr. Halina Przymusinska.

I would also like to acknowledge the comments and suggestions made by Prof. Dr. Carlos Damásio, Prof. Dr. Thomas Eiter which helped the improvement of the submitted version of this Thesis into this published version.

Thanks are also due to Dr. Miguel Calejo and Prof. Dr. Paulo Quaresma for their comments and suggestions.

The warmest of thanks goes to Carolina Sobral, Dina Luís, Francisco Pereira, Joana Lima, Joana Sobral, Leonor Barata, Luís Trindade, Miguel Ferrand and Vitor Nogueira, for being the best of friends.

A thank you also goes to the guys at FBA, in particular to Alexandre Matos, Cristina Alves, Conceição Miranda and João Bicker.

For always turning scientific conferences into very pleasant experiences, I would like to thank Daniela Carbogim, David Pearce and Paolo Torroni.

Quebra-Costas in Coimbra, and Tertúlia Bar in Lisbon, have always been my places of choice for relaxing from work (although Tertúlia has often served as an alternative working place), for which I thank them.

I would like to acknowledge the Departamento de Informática, and the Centro de Inteligência Artificial, for the working conditions provided. A special word of appreciation goes to Prof. Dr. Pedro Barahona for all his effort in making it possible to write this Thesis in English.

A special word of gratitude goes to the Secretarial staff for being so friendly and efficient.

This work was partially supported by PRAXIS XXI scholarship no. BD/13514/97, and conceived out within the MENTAL project PRAXIS XXI 2/2.1/TIT/1593/95, directed by my supervisor.
Abstract

Most of the work conducted so far in the field of logic programming has focused on representing static knowledge, i.e., knowledge that does not evolve with time. This is a serious drawback when dealing with dynamic knowledge bases in which not only the extensional part (the set of facts) changes dynamically but so does the intensional part (the set of rules).

In this thesis we investigate updates of knowledge bases represented by logic programs, where the updates are far more expressive than a mere insertion and deletion of facts since they can be specified by means of arbitrary program rules and thus they themselves are logic programs. Based on the notion of logic program updates, we define the notion of Dynamic Logic Programming (DLP) which characterizes knowledge given by a sequence of logic programs, each representing a state of the world. This notion is further extended to the case where such modules can be organized according to an acyclic digraph, introducing Multi-dimensional Dynamic Logic Programming (MDLP), thus allowing the concurrent representation of several dimensions of a knowledge system (e.g., time, hierarchy, etc.).

We then investigate the concept of Evolving Knowledge Bases (EKB). An EKB is a knowledge base which can not only be externally updated, but is capable of self evolution by means of its internally specified behaviour. To accomplish a uniform specification of such self and external updates, we extend the language of updates LUPS, otherwise only capable of specifying external updates, to be able to specify both the external updates as well as the internal behaviour and its updates.

The so defined Evolving Knowledge Bases, with an MDLP based knowledge representation, allow: the combination of knowledge from different sources; the specification of external updates to the knowledge of each individual source; the specification of relations between the sources of knowledge according to elaborate precedence relations; the update of such precedence relations; the specification and update of the evolution of such precedence relations; the specification and update of the knowledge base’s internal behaviour; the access to and reasoning about external observations; the specification of multiple entities (sub-agents), within an Evolving Knowledge Base, each independently carrying out part of the internal behaviour; the specification and update of precedence relations among such sub-agents.

Throughout this thesis, we incrementally specify, semantically characterize, and illustrate with examples, the concepts and tools necessary to the development of Evolving Knowledge Bases.
# Contents

Acknowledgements v

Abstract vii

List of Figures xiii

Preface xv

1 Introduction 1
   1.1 Starting Point . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
   1.2 The Roadmap . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
   1.3 The Route . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
      1.3.1 Milestones . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
   1.4 About the Route . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2 Logic Programming for NMR 11
   2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
   2.2 Brief Historical Perspective . . . . . . . . . . . . . . . . . . . . . . . . 12
   2.3 Language of Logic Programs . . . . . . . . . . . . . . . . . . . . . . . 14
   2.4 Model Theory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
      2.5.1 Definite Logic Programs . . . . . . . . . . . . . . . . . . . . . . 16
      2.5.2 Normal Logic Programs . . . . . . . . . . . . . . . . . . . . . . 17
      2.5.3 Extended Logic Programs . . . . . . . . . . . . . . . . . . . . . 18
   2.6 Generalized Logic Programs . . . . . . . . . . . . . . . . . . . . . . . 18
      2.6.1 Language . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
      2.6.2 Semantics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
      2.6.3 Adding Strong Negation . . . . . . . . . . . . . . . . . . . . . . 23
   2.7 Concluding Remarks . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26

3 Dynamic Logic Programming 29
   3.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
   3.2 Logic Program Updates . . . . . . . . . . . . . . . . . . . . . . . . . . 34
      3.2.1 Declarative Semantics . . . . . . . . . . . . . . . . . . . . . . . 34
      3.2.2 Transformational Semantics . . . . . . . . . . . . . . . . . . . . 36
      3.2.3 Properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
   3.3 Dynamic Logic Programming . . . . . . . . . . . . . . . . . . . . . . . 47
      3.3.1 Declarative Semantics . . . . . . . . . . . . . . . . . . . . . . . 48
      3.3.2 Properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
      3.3.3 Transformational Semantics . . . . . . . . . . . . . . . . . . . . . 64
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.6.1</td>
<td>Organizational Decision Making</td>
<td>199</td>
</tr>
<tr>
<td>6.6.2</td>
<td>Multiple Inheritance</td>
<td>200</td>
</tr>
<tr>
<td>6.6.3</td>
<td>Legal Reasoning</td>
<td>203</td>
</tr>
<tr>
<td>6.6.4</td>
<td>Representing Inter- and Intra-Agent Social Viewpoints</td>
<td>204</td>
</tr>
<tr>
<td>6.6.5</td>
<td>MDLP and Multi-agent Systems</td>
<td>209</td>
</tr>
<tr>
<td>6.7</td>
<td>Comparison with other work</td>
<td>210</td>
</tr>
<tr>
<td>6.8</td>
<td>Conclusions and Future Work</td>
<td>212</td>
</tr>
<tr>
<td>7</td>
<td>Multi-dimensional KABUL</td>
<td>215</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>215</td>
</tr>
<tr>
<td>7.2</td>
<td>Core Language</td>
<td>217</td>
</tr>
<tr>
<td>7.2.1</td>
<td>KABUL&lt;sup&gt;m&lt;/sup&gt; - Syntax</td>
<td>218</td>
</tr>
<tr>
<td>7.2.2</td>
<td>KABUL&lt;sup&gt;m&lt;/sup&gt; - Semantics</td>
<td>221</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Update Program Reduct</td>
<td>222</td>
</tr>
<tr>
<td>7.2.4</td>
<td>Executable Commands</td>
<td>223</td>
</tr>
<tr>
<td>7.2.5</td>
<td>Successor State</td>
<td>223</td>
</tr>
<tr>
<td>7.3</td>
<td>Introducing DAG Commands</td>
<td>225</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Hierarchy Commands</td>
<td>225</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Prevalence Mode Commands</td>
<td>228</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Agent Commands</td>
<td>235</td>
</tr>
<tr>
<td>7.4</td>
<td>Multiple Update Programs</td>
<td>236</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Framework and Syntax</td>
<td>237</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Semantics</td>
<td>241</td>
</tr>
<tr>
<td>7.5</td>
<td>Summary and Open Issues</td>
<td>246</td>
</tr>
<tr>
<td>8</td>
<td>Illustrative Examples</td>
<td>247</td>
</tr>
<tr>
<td>8.1</td>
<td>A Dynamic Knowledge Base for Stock Ratings</td>
<td>247</td>
</tr>
<tr>
<td>8.1.1</td>
<td>Scenario Overview</td>
<td>247</td>
</tr>
<tr>
<td>8.1.2</td>
<td>Knowledge Base Specification</td>
<td>248</td>
</tr>
<tr>
<td>8.1.3</td>
<td>Knowledge Base Evolution</td>
<td>256</td>
</tr>
<tr>
<td>8.1.4</td>
<td>Further Elaborations</td>
<td>265</td>
</tr>
<tr>
<td>8.2</td>
<td>An Agent Architecture</td>
<td>266</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Introduction</td>
<td>266</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Overall Architecture</td>
<td>267</td>
</tr>
<tr>
<td>8.2.3</td>
<td>Related Work</td>
<td>271</td>
</tr>
<tr>
<td>8.2.4</td>
<td>Final Remarks</td>
<td>274</td>
</tr>
<tr>
<td>9</td>
<td>Conclusions and Future Directions</td>
<td>277</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>283</td>
</tr>
<tr>
<td>A</td>
<td>List of Symbols</td>
<td>303</td>
</tr>
</tbody>
</table>
List of Figures

5.1 Circuit with four delays .............................................. 161
5.2 Falling Edge JK Flip-Flop ........................................... 162
5.3 Rising Edge JK Flip-Flop ............................................ 163

6.1 A Directed Acyclic Digraph ....................................... 178
6.2 MDLP of Example 56 ................................................ 179
6.3 MDLP of Example 57 ................................................ 180
6.4 A DAG with one student and two supervisors ..................... 181
6.5 A DAG with one student, two supervisors and a sink ............. 183
6.6 DAG of Example 58 ................................................ 184
6.7 An MDLP representing a company .................................. 201
6.8 A Taxonomy ........................................................ 201
6.9 Clyde - The Royal Elephant ....................................... 202
6.10 DAG of Example 63 ................................................ 203
6.11 Two Dimensions of a Multi-Agent System ......................... 204
6.12 Equal Role Representation ....................................... 205
6.13 Time Prevailing Representation .................................. 206
6.14 Hierarchy Prevailing Representation ............................. 207
6.15 Sub-agent Hierarchy ................................................ 208
6.16 Inter- and Intra-Agent Relationship Representation ............. 208
6.17 A Lattice with two dimensions .................................... 209
6.18 The view of the Senior Researcher ............................... 210
6.19 The view of the Student .......................................... 210

7.1 Time Prevalence Mode .............................................. 233
7.2 Set the Past to Time Prevalence Mode ............................ 233
7.3 Hierarchy Prevalence Mode ........................................ 234
7.4 Hierarchy Change under Hierarchy Prevalence Mode ............... 234
7.5 Clear all Past Prevalence Modes .................................. 235
7.6 Recent Time Prevalence with $\delta = 2$ ............................. 235

8.1 Trade Market ..................................................... 254
8.2 Bull Market ....................................................... 255
8.3 Bear Market ....................................................... 256
8.4 The MINERVA agent architecture ................................. 268
This work started out with a belief and a desire. The belief that symbolic based representation and logic reasoning are (still) powerful and useful approaches to the study of Artificial Intelligence (AI), and that to follow such a stance Logic Programming and Non-monotonic Reasoning (LP&NMR) are probably the best available vehicles. The desire to prove LP&NMR to be a useful framework within the context of the latest AI trend: intelligent agents.

When facing with the fact that very little had been done in what concerns the use of Logic Programming to represent and reason about dynamic knowledge, i.e. knowledge that evolves with time, we soon realized that this job was going to be more of a ground breaking bottom-up task instead of a goal oriented top-down one. And this is how things went.

This rather long volume introduces several completely novel approaches, such as the framework of Dynamic Logic Programming to represent knowledge base updates, with no distinction between the extensional and intensional part of a knowledge base, and the introduction of the concept of an Evolving Knowledge Base - a knowledge base which also incorporates a specification of its dynamic behaviour and where both the knowledge and the behaviour can be updated. But most importantly, this volume should be regarded as an intermediate step of a long journey. Even though several solid and stable results are presented throughout, bringing closure to some problems, many other issues are addressed for which a fully satisfactory solution is not achieved, thus opening many doors. Some people may argue that such unpolished results should not be reported in a PhD Thesis. We argue the contrary: this is just a PhD Thesis - the beginning, not the end.

This Thesis, as well as the authors’ M.Sc. Dissertation[130] where some preliminary results on the subject of Logic Program Updates first appeared, grew within the context of our MENTAL agents project¹, led by the authors’ supervisor Prof. Dr. Luís Moniz Pereira, at the Centro de Inteligência Artificial, and Departamento de Informática, Universidade Nova de Lisboa. The aim of the MENTAL project was that of establishing, on a sound theoretical basis, the design of an overall architecture for mental agents based on, and building upon the strengths of logic programming. The MENTAL project will be followed by the FLUX project², led by Prof. Dr. José Júlio Alferes, within which we will carry on our research efforts in these directions.

Parts of this work are based on the following publications³:


¹Portuguese research funding programme PRAXIS XXI, project MENTAL (2/2.1/TIT/1593/95).
²Portuguese research funding programme POSI, project FLUX (POSI/SRI/40958/2001).
³These publications are available at http://centria.di.fct.unl.pt/~jleite/


The order of authors, where alphabetical, is not intended to indicate the extent of the individual contributions. Part of the results presented throughout this work have been obtained in collaboration with the co-authors, and for it I thank them profusely. At the beginning of each Chapter, there is mention to the publications where the results have partially appeared. This work was partially supported by PRAXIS XXI scholarship no. BD/13514/97. The author’s home page is located at http://centria.di.fct.unl.pt/~jleite/, and he can be reached at jleite@di.fct.unl.pt.

Lisbon, 28th of August, 2002

João Alexandre Carvalho Pinheiro Leite
Chapter 1

Introduction

In this introductory Chapter we briefly outline the directions taken throughout the remainder of the work, with special emphasis on the motivation which constitutes the starting point of our explorations. It aims only at providing a birds-eye view of this thesis. Rather more detailed motivation is to be found throughout, as each step is taken.

1.1 Starting Point

The development of machines that exhibit an intelligent behaviour has always been at the very core of Artificial Intelligence (AI). Representing and reasoning about knowledge is one of the most fundamental and challenging tasks of AI. Since the mid-fifties there has been a strong desire to use logic-based languages to deal with such issues. The advantages of the approach were expressed by John McCarthy [161, 164] as follows:

> Expressing information in declarative sentences is far more modular than expressing it in segments of computer programs or in tables. Sentences can be true in a much wider context than programs can be used. The supplier of a fact does not have to understand much about how the receiver functions or how or whether the receiver will use it. The same fact can be used for many purposes, because the logical consequences of collections of facts can be available.

This stance originated a wide and lasting research effort, initially leading to two main independent streams: Non-Monotonic Reasoning (NMR), and Logic Programming (LP).

Non-Monotonic Reasoning, diverging from the use of classical logic and its closed and monotonic domains, aims at establishing formal reasoning methods that allow an adequate representation of common-sense reasoning, inherently non-monotonic: we are constantly forced to withdraw conclusions in face of new information, something that is not possible to represent with the intrinsically monotonic classical logic.

Logic Programming (LP) combines logic as a representation language, with the theory of automated deduction. LP introduced in Computer Science the important
CHAPTER 1. INTRODUCTION

class of declarative, as opposed to procedural programming. A procedural language can be seen as one to specify "how", whereas a declarative language is one to specify "what". According to this declarative view, summarized by Robert Kowalski [120] as 
Algorithm=Logic+Control, a programmer should only be concerned with the declarative meaning of the program, while the procedural aspects of its execution should be handled automatically.

The declarative nature of logic programming together with its amenability to implementation made it an important candidate for knowledge representation and non-monotonic reasoning, based on the so called "logical approach to knowledge representation". The cross-fertilization with Non-Monotonic Reasoning gave rise to the field of Logic Programming for Non-monotonic Reasoning, this being the area of research of this work.

Most of the work conducted so far in the field of Logic Programming has focused on representing static knowledge, i.e., knowledge that does not evolve with time. If we are to move to a more open and dynamic environment, typical of these web and agent days, we need to consider ways of representing knowledge which may evolve in time. If we are to accept the Logic Programming based approach to knowledge representation and reasoning, it must prove to be an appropriate framework capable of representing not only static knowledge, i.e. knowledge that does not evolve with time, but also dynamic knowledge, i.e. knowledge that does.

When dealing with modifications to a knowledge base represented by a propositional theory, two kinds of abstract frameworks have been distinguished both by Keller and Winslett [117] and by Katsuno and Mendelzon [116]. One, theory revision, deals with incorporating new knowledge about a static world whose previous representation was incomplete or incorrect. The other deals with changing worlds, and is known as theory update. Theory revision has been extensively studied in the context of Logic Programming (cf. [3, 10, 11, 33, 87, 88, 184, 202, 229, 230]), but in this work, we are concerned with representing and reasoning about worlds that change, i.e. we will investigate the problem of theory update.

Several authors have addressed the issue of updates of logic programs and deductive databases (see e.g. [12, 56, 102, 103, 157, 194, 197]), most of them following the so called "interpretation update" approach, originally proposed in [116, 228].

This approach is based on the idea of reducing the problem of finding an update of a knowledge base $DB$ by another knowledge base $U$ to the problem of finding updates of its individual interpretations (models). More precisely, a knowledge base $DB'$ is considered to be the update of a knowledge base $DB$ by $U$ if the set of models of $DB'$ coincides with the set of updated models of $DB$, i.e., 

$$
\text{the set of models of } DB' = \text{the set of updated models of } DB.
$$

The update of models is governed by the update rules specified in $U$, and also by inertia applied to those literals of the models not directly affected by the update program. Thus, according to the interpretation update approach, the problem of finding an update of a deductive database $DB$ is reduced to the problem of finding individual updates of all of its relational instantiations (models) $M$.

Unfortunately, such an approach suffers, in general, from several important drawbacks:

- In order to obtain the update $DB'$ of a knowledge base $DB$ one has to first compute all the models $M$ of $DB$ (typically, a daunting task) and then individually compute their (possibly multiple) updates $M_U$ by $U$. An update $M_U$ of a given

---

1The notion of a model depends on the type of considered knowledge bases and on their semantics. Here, we are considering (generalized) logic programs under the stable model semantics.
interpretation $M$ is obtained by changing the status of only those literals in $M$ that are “forced” to change by the update $U$, while keeping all the other literals intact by inertia (see e.g. [157, 194, 197]).

- The updated knowledge base $DB'$ is not defined directly but, instead, it is indirectly characterized as a knowledge base whose models coincide with the set of all updated models $M_U$ of $DB$. In general, there is therefore no natural way of computing $DB'$, which may even not exist at all, because the only straightforward candidate for $DB'$ is the typically intractably large knowledge base $DB''$ consisting of all clauses that are entailed by all the updated models $M_U$ of $DB$.

- Most importantly, while the semantics of the resulting knowledge base $DB'$ indeed represents the intended meaning when just the extensional part of the knowledge base $DB$ (the set of facts) is being updated, it leads to strongly counter-intuitive results when also the intensional part of the database (the set of rules) undergoes change, as the following example shows.

**Example 1** Consider the logic program $P$ consisting of the following rules:

$$
\begin{align*}
sleep & \leftarrow \neg tv\_on \\
tv\_on & \leftarrow \\
watch\_tv & \leftarrow tv\_on.
\end{align*}
$$

Clearly $M = \{tv\_on, watch\_tv\}$ is its only stable model. Suppose now that the update $U$ states that there is a power failure, and if there is a power failure then the TV is no longer on, as represented by the logic program $U$:

$$
\begin{align*}
\neg tv\_on & \leftarrow power\_failure \\
power\_failure & \leftarrow
\end{align*}
$$

According to the above mentioned interpretation approach to updating, we would obtain $M_U = \{power\_failure, watch\_tv\}$ as the only update of $M$ by $U$. This is because $power\_failure$ needs to be added to the model and its addition forces us to make $tv\_on$ false. As a result, even though there is a power failure, we are still watching TV. However, by inspecting the initial program and the updating rules, we are likely to conclude that since “watch\_tv” was true only because “tv\_on” was true, the removal of “tv\_on” should make “watch\_tv” false by default. Moreover, one would expect “sleep” to become true as well. Consequently, the intended model of the update of $P$ by $U$ is the model $M_U = \{power\_failure, sleep\}$.

Let us look at another example:

**Example 2** Consider the logic program:

$$
\begin{align*}
free & \leftarrow \neg jail \\
jail & \leftarrow jail\_for\_eutanasia \\
jail\_for\_eutanasia & \leftarrow eutanasia
\end{align*}
$$

whose only stable model is $M = \{free\}$. Suppose now that the update $U$ states that “eutanasia becomes true, i.e. $U = \{eutanasia \leftarrow\}$. According to the interpretation approach to updating, we would obtain $\{free, eutanasia\}$ as the only update of $M$ by $U$. 
CHAPTER 1. INTRODUCTION

However, by inspecting the initial program and the update, we are likely to conclude that, since “free” was true because “jail” could be assumed false, which was the case because “eutanasia” was false, now that “eutanasia” became true “jail_for_eutanasia” and “jail” should also have become true, and “free” should be removed from the conclusions.

These two examples, we believe, are sufficient evidence to support the claim that existing methods, based on the notion of interpretation updates, are not adequate when we also wish to update the intensional part of a knowledge base. This constitutes the starting point to this work, its goal being to investigate the problem of updating knowledge bases represented by (generalized) logic programs.

1.2 The Roadmap

Why are we obtaining these somewhat unintuitive results when performing updates using the interpretation based approach? To answer this question we must first consider the role of inertia in updates.

Newton’s first law, also known as the law of inertia, states that: “every body remains at rest or moves with constant velocity in a straight line, unless it is compelled to change that state by an unbalanced force acting upon it” (adapted from [174]). Commonsense often tends to interpret this law as things keeping as they are unless some kind of force is applied to them. This is true but it doesn’t exhaust the meaning of the law. It is the result of all applied forces that governs the outcome. Take a body to which several forces are applied, and which is in a state of equilibrium due to those forces cancelling out. Later one of those forces is removed and the body starts to move.

The same kind of behaviour presents itself when updating programs. Before obtaining the truth value, by inertia, of those elements not directly affected by the update program, one should verify whether the truth of such elements is not indirectly affected by the updating of other elements.

These examples illustrates that, when updating knowledge bases, it is not sufficient to just consider the truth values of literals figuring in the heads of its rules because the truth value of their rule bodies may also be affected by the updates of other literals. In other words, it suggests that the principle of inertia should be applied not just to the individual literals in an interpretation but rather to entire rules of the knowledge base.

The first part of this work will be devoted to the problem of updating knowledge bases represented by generalized logic programs, where we propose a new solution that attempts to eliminate the drawbacks of the previously proposed approaches. Specifically, given one generalized logic program $P$ (the initial program) and another logic program $U$ (the updating program) we characterize the result of updating $P$ with $U$, denoted by $P \oplus U$. Due to the application of the inertia principle not just to literals but to entire program rules, the semantics of our updated program $P \oplus U$ avoids the drawbacks of interpretation updates and seems to properly represent the intended semantics. Nevertheless, while our notion of program update significantly differs from the notion of interpretation update, it coincides with the latter (as originally introduced in [157] under the name of revision program and later reformulated in the language of logic programs in [194, 197]) when the initial program $P$ is purely extensional, i.e., when the initial program is just a set of facts.

Subsequently, we introduce the paradigm of Dynamic Logic Programming (DLP). The idea behind this paradigm is simple and quite fundamental. Suppose that we are
given a set of program modules $P_s$, indexed by a sequence of states $s$, each representing a time period. Each program $P_s$ contains some knowledge that is supposed to be true at the state $s$. Consequently, the individual program modules may contain mutually contradictory as well as overlapping information. The role of dynamic logic programming is to use the mutual relationships existing between different states (and specified in the form of the ordering relation) to precisely determine, at any given state $s$, the declarative as well as the procedural semantics of the combined program, composed of all modules, denoted by $P_1 \oplus \ldots \oplus P_s$. The introduction of Dynamic Logic Programming extends Logic Programming, making possible for a logic program to undergo a sequence of modifications, opening up the possibility of incremental design and evolution of logic programs, therefore significantly facilitating modularization of logic programming and, thus, modularization of non-monotonic reasoning as a whole.

Whereas Dynamic Logic Programming provides a meaning to sequences of logic programs, it says nothing about how to obtain them. Each logic program can represent newly incoming information, totally independent from the previous states, but can also depend on the previous state of affairs. To allow the specification of such logic programs whose construction can depend on the previous state, Alferes et al. introduced the language of updates LUPS [15]. The language LUPS is based on a notion of update commands that allow the specification (construction) of logic programs. Each command in LUPS, which can be issued in parallel with other such commands, specifies an update action which, in its basic form, encodes the assertion or retraction of a logic program rule. A collection of such commands, whose execution can be made dependent on the semantics of the current sequence of logic programs, specifies the next logic program to be added to such sequence. An example of one such commands is:

\[
\text{assert } L \leftarrow L_1, \ldots, L_k \text{ when } L_{k+1}, \ldots, L_m
\]

This command, when issued at some state $s$, means that if $L_{k+1}, \ldots, L_m$ holds at state $s - 1$, then the rule $L \leftarrow L_1, \ldots, L_k$ should belong to the logic program at state $s$. Besides such basic commands that specify the next logic program in the sequence, LUPS also allows for so called persistent commands. These, when issued, will not only contribute to the specification of the next logic program, but they will contribute as well to the specification of every future logic program, until cancelled.

This notion of allowing commands to specify more than one state transition leads us to introduce the concept of Evolving Knowledge Bases, i.e. transform an otherwise static knowledge base which is only capable of responding to outside stimuli into a dynamic one where update commands are made part of the knowledge base, making its evolution viable according to its own behaviour specification, possibly without any external updates.

An Evolving Knowledge Base is a knowledge base which can change due to external update commands, but which also contains an internal specification encoding its behaviour. According to this notion, an evolving knowledge base at some state $s$, consists of a tuple $\langle P_s, SU_s \rangle$ where $P_s = P_1 \oplus \ldots \oplus P_s$ is a dynamic logic program (DLP), and $SU_s$ is the self update, i.e. a specification that partially encodes its future evolution. At each state transition, the knowledge base receives a set of external updates $\langle EU_{s+1} \rangle$ and perceives a set of external observations $\langle EO_{s+1} \rangle$. The self and external updates are combined, evaluated against the dynamic logic program and external observations, to determine the next state of the evolving knowledge base $\langle P_{s+1}, SU_{s+1} \rangle$:

\[
\langle P_s, SU_s \rangle \xrightarrow{\langle EU_{s+1}, EO_{s+1} \rangle} \langle P_{s+1}, SU_{s+1} \rangle
\]
CHAPTER 1. INTRODUCTION

To specify such knowledge bases, we need an update language capable of not only specifying the assertions and retractions of the object level rules in the DLP, but also capable of updating the behaviour specified by the self update. One way to extend the existing languages of updates in order to express such statements, is to allow the update commands not only to specify the set of logic program rules that belong to the KB produced by the “current” state transition, but also to specify a set of commands that belong to the self update that will govern the next state transition, i.e. we need commands to assert and retract the **assert** and **retract** commands, and we need to assert and retract these, in turn, i.e., we need nested (or embedded) commands.

Given that $P_s$ encodes the object level knowledge and $SU_s$ encodes the behaviour, we need to define a language capable of updating them both. To this purpose, we introduce the **Knowledge And Behaviour Update Language (KABUL)** as a language to fully specify such **Evolving Knowledge Bases**.

Even though the main motivation behind the introduction of **DLP** was to represent the evolution of knowledge in time, the relationship between the different states can encode other aspects of a system. In fact, since its introduction, **DLP** (and **LUPS**) has been employed to represent a stock of features of a system, some of which explored in this work, namely as a means to represent and reason about the evolution of knowledge in time; combine rules learnt by a diversity of agents; reason about updates of agents’ beliefs; model agent interaction; model and reason about actions; resolve inconsistencies in metaphorical reasoning. The common property among these applications of **DLP** is that the states associated with the given set of theories encode a single one of several possible representational dimensions (e.g. time, hierarchies, domains,...). This is so inasmuch **DLP** is defined for linear sequences of states alone.

For example, **DLP** can be used to model the relationship of a hierarchical related group of agents, and **DLP** can be used to model the evolution of a single agency over time. But **DLP**, as it stands, cannot deal with both settings at once, and model the evolution of one such group of agents over time. An instance of such a multi-dimensional scenario can be found in legal reasoning, where the legislative agency is divided conforming to a hierarchy of power, governed by the principle **Lex Superior (Lex Superior Derogat Legi Inferiori)** according to which the rule issued by a higher hierarchical authority overrides the one issued by a lower one, and the evolution of law in time is governed by the principle **Lex Posterior (Lex Posterior Derogat Legi Priori)** according to which the rule enacted at a later point in time overrides the earlier one. **DLP** can be used to model each of these principles separately, by using the sequence of states to represent either the hierarchy or time, but is unable to cope with both at once when they interact.

In order to overcome this limitation, we introduce **Multi-dimensional Dynamic Logic Programming (MDLP)**. According to **MDLP**, a generalization of **DLP**, knowledge is given by a set of logic programs, indexed by collections of states organized into arbitrary acyclic directed graphs (**DAGs**) representing precedence relations. **MDLP** assigns semantics to sets and subsets of logic programs, depending on how they stand in relation to one another, as defined by the acyclic digraph (**DAG**) that represents the states and their configuration. The extra flexibility afforded by the **DAG** allows **MDLP** to provide a semantic framework for the representation, in a unified manner and with precise declarative and procedural semantics, to not only knowledge represented by logic programs related according to some hierarchy (possibly involving multiple inheritance), but also to represent its inner dynamics, i.e. its evolution in time, as well as any other aspect of a knowledge system that is representable in **DLP**. By dint of such natural
generalization, $MDLP$ affords extra expressiveness, thereby enlarging the latitude of logic programming applications unifiable under a single framework. The generality and flexibility provided by a $DAG$ ensures a wide scope and variety of new possibilities. By virtue of the newly added characteristics of multiplicity and composition, $MDLP$ provides a “societal” viewpoint in Logic Programming, important in these web and agent days, for combining knowledge in general.

Whereas $MDLP$ provides a declarative and procedural semantics of a sequence of program updates, based on a hierarchy determined by a $DAG$, as it was the case with $DLP$, it does not provide any language for the specification of such multi-dimensional updates, which depend on the knowledge acquired in the intervening states and on the topology of the $DAG$. It also does not provide any language for the specification (and possible updating) of the precedence topology itself. To address this issue, we extend $KABUL$ with the powerful capability of specifying and updating evolving knowledge bases which enjoy the extra capabilities provided by $MDLP$.

This last piece of the puzzle, which embeds all previously introduced ones, allows the specification, with a precise semantical characterization, of rather elaborate $Evolving Knowledge Bases$ with features such as the abilities to:

- combine knowledge from different sources;
- specify external updates to the knowledge of each individual source;
- relate the sources of knowledge according to elaborate precedence relations;
- update such precedence relations;
- specify and update the evolution of such precedence relations;
- specify and update the internal behaviour of the knowledge base;
- access and reason about external observations;
- specify multiple entities (sub-agents), within an evolving knowledge base, each independently carrying out part of the internal behaviour;
- specify and update precedence relations among such sub-agents;

1.3 The Route

The remainder of this work is structured as follows:

Chapter 2 - Logic Programming For Non-Monotonic Reasoning: Along this Chapter we provide a bird’s-eye view of the field of Logic Programming for Non-Monotonic Reasoning. We start with a brief historical perspective before presenting the syntax and semantics of logic programs used throughout this work. In particular, we present a class of logic programs with default negation both in the premises and conclusions of clauses, and present its stable semantics. These programs, dubbed generalized logic programs, play an important role within the context of updates of knowledge bases and will constitute our main knowledge representation vehicle.
**Chapter 3 - Dynamic Logic Programming:** In this Chapter we investigate updates of knowledge bases represented by logic programs. In order to represent negative information we use generalized logic programs. We start by introducing the notion of an update $P \oplus U$ of one logic program $P$ by another logic program $U$. Subsequently, we provide a precise semantic characterization of $P \oplus U$, and study some basic properties of program updates. In particular, we show that our update programs generalize the notion of interpretation update. We then extend this notion to compositional sequences of logic programs updates $P_1 \oplus P_2 \oplus \ldots$, introducing the paradigm of *Dynamic Logic Programming*. We also study some properties of $DLP$, present some illustrative examples and compare $DLP$ with other approaches to updates.

**Chapter 4 - Knowledge Update Language:** Whereas $DLP$ provides a framework and semantics to determine the meaning of sequences of logic programs, it does not provide a mechanism to construct such programs. Languages of updates accomplish this goal, and are the subject of this Chapter. We start with an overview of the language of updates LUPS [15] and its extension EPI [75]. Then we identify an intuitively incorrect behaviour of LUPS semantics and one possible, important, extension to its syntax. To address these issues, the *Knowledge Update Language* ($KUL$) is introduced and compared to its predecessors. Finally, we present some illustrative examples and show how $KUL$ can be used to specify the effects of actions.

**Chapter 5 - Knowledge and Behaviour Update Language:** In this Chapter we extend the language of updates $KUL$ with several features, the most important being the possibility to allow the knowledge base to evolve not only due to external updates but also due to self updates, thereby introducing the *Knowledge And Behaviour Update Language* ($KABUL$) that allows the specification of updates to both the object level knowledge base and to the self-updates that encode the behaviour of the knowledge base. Examples and comparisons are presented.

**Chapter 6 - Multi-Dimensional Dynamic Logic Programming:** In this Chapter we introduce $MDLP$, a generalization of $DLP$ that allows knowledge to be given by a set of logic programs, indexed by collections of states organized into arbitrary acyclic directed graphs ($DAG$s) representing precedence relations among those states. Several illustrative examples are also presented.

**Chapter 7 - Multi-Dimensional Update Language:** In this Chapter, we extend $KABUL$ with the powerful capability of specifying and updating evolving knowledge bases which enjoy the extra capabilities provided by $MDLP$.

**Chapter 8 - Illustrative Examples:** In this Chapter we present two applications of the framework set forth, rather more elaborate than the examples proffered throughout. The first concerns the modelling of a financial advisory knowledge base which combines the advice with provenance in different advisors, together with stock market data, to produce stock acquisition recommendations. The second concerns the use of $MDLP$ and $KABUL$ as the basis for representing agents’ epistemic states.

**Chapter 9 - Concluding Remarks:** In this Chapter we wrap up and conclude, pointing towards future research directions.
1.3. **THE ROUTE**

The best way to digest this document is to go through it in sequence. Most of the Chapters build on the previous one, and each is necessary to understand the subsequent one. The only exception concerns Chapter 6 which builds on the theories set forth in Chapter 3, and can be read before moving to Chapters 4 and 5. We have therefore the following two reading paths:

1. $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$
2. $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 8 \rightarrow 9$

For the reader interested in the semantics of logic program modules, arranged either as a sequence or as a DAG, and who is not concerned with the way such programs are obtained, we can also suggest the following short reading path:

$1 \rightarrow 2 \rightarrow 3 \rightarrow 6$

Appendix A contains a List of Symbols.

### 1.3.1 Milestones

The main contributions of this thesis, supported on extensive joint work, are:

- definition of a precise semantics for rule based updates, based on the application of the principle of inertia to rules rather than model literals, leading to the notion of updates based on causal rejection of rules, characterizing the result of updating a (generalized) logic program $P$ by another (generalized) logic program $U$;
- introduction of the paradigm of Dynamic Logic Programming (DLP), characterizing knowledge provided by sequences of (generalized) logic programs;
- extension of DLP to deal with programs indexed by acyclic digraphs, introducing Multi-dimensional Dynamic Logic Programming (MDLP);
- introduction of the Knowledge and Behaviour Update Language (KABUL), by specifying its syntax and DLP based semantics, as a framework to specify evolving knowledge bases;
- extension of KABUL to deal with the evolution of MDLP based evolving knowledge bases;
- presentation of several examples, along the entire work, to illustrate the applicability and power of all contributions;
- coherent, consolidated and progressive presentation of the above.

Several important issues could not be covered here, such as for example establishing three valued well founded semantics for updates, or integrate them with other existing non-monotonic reasoning frameworks such as preferences, revision, to name a few. Throughout this thesis we will comment on such issues, that were left unexplored, opening doors for future research paths.
1.4 About the Route

Over recent years, the notion of agency has claimed a major role in defining the trends of modern research. Influencing a broad spectrum of disciplines such as Sociology, Psychology, among others, the agent paradigm virtually invaded every sub-field of Computer Science (c.f. [114] for a survey). Although commonly implemented by means of imperative languages, mainly for reasons of efficiency, the agent concept has recently increased its influence in the research and development of computational logic based systems. Since efficiency is not always the crucial issue, but clear specification and correctness is, Logic Programming and Non-monotonic Reasoning have been revived back into the spotlight [4, 114, 123, 160, 208, 211].

The Logic Programming paradigm provides a well-defined, general, integrative, encompassing, and rigorous framework for systematically studying computation, be it syntax, semantics, procedures, or attending implementations, environments, tools, and standards. LP approaches problems, and provides solutions, at a sufficient level of abstraction so that they generalize from problem domain to problem domain. This is afforded by the nature of its very foundation in logic, both in substance and method, and constitutes one of its major assets. To this accrues the recent significant improvements in the efficiency of Logic Programming implementations for Non-monotonic Reasoning [69, 176, 216, 231]. Besides allowing for a unified declarative and procedural semantics, eliminating the traditional wide gap between theory and practice, the use of several and quite powerful results in the field of non-monotonic extensions to Logic Programming (LP), such as belief revision, inductive learning, argumentation, preferences, abduction, etc.[211] can represent an important composite added value to the design of rational agents. These results, together with the improvement in efficiency, allow the referred mustering of Logic Programming and Non-monotonic Reasoning to accomplish a felicitous degree of combination between reactive and rational behaviours of agents, the Holy Grail of modern Artificial Intelligence, whilst preserving clear and precise specification enjoyed by declarative languages.

Until recently, Logic Programming could be seen as a good representation language for static knowledge. If we are to move to a more open and dynamic environment, typical of the agency paradigm, we need to consider ways of representing and integrating knowledge from different sources which may evolve in time. Moreover, an agent not only comprises knowledge about each state, but also some form of knowledge about the transitions between states. This knowledge about state transitions can represent the agent’s knowledge about the environment’s (or other agents’) evolution, as well as its own behaviour and evolution.

The introduction of Evolving Knowledge Bases allows for a unified declarative specification of such states, state transitions and their evolution, whilst preserving the underlying Logic Programming based representation, thus making it amenable to be the melting pot for combining existing non-monotonic extensions to Logic Programming. We therefore hope that this work contributes to the opening of Logic Programming for Non-monotonic Reasoning to these otherwise unreachable dynamic worlds, typical of the agency paradigm.