COLLABORATIVE LOGIC PROGRAMMING
VIA DEDUCTIVE-INDUCTIVE RESOLUTION

Jian Huang
March 2009

Submitted in total fulfilment of the requirements
of the degree of Doctor of Philosophy

Department of Computer Science and Software Engineering
The University of Melbourne
Victoria, Australia

Produced on archival quality paper
Abstract

This thesis presents a powerful deductive-inductive resolution technique, by combining deductive theorem proving with inductive logic programming, for solving a new class of multi-agent problems—namely the collaborative logic programming (CollabLP) problems.

In essence, the CollabLP formulation captures a wide range of problems in multi-agent settings where knowledge is typically distributed, private and possibly incomplete. Meanwhile, communication is allowed among the agents but restricted only to be in the form of simple logic programming queries. CollabLP captures not only problems requiring induction in multi-agent environments, but also deductive problems requiring collaboration in general.

Under the deductive-inductive resolution (DIR) approach to the CollabLP problem, induction is viewed as an integral component and natural extension of an agent’s deductive process. The DIR approach tightly integrates processes of deduction and induction among agents, where communication is limited to inductive hypotheses and deductive consequences.

Based on a modal treatment, the DIR approach is proven to be both sound (in general) and complete (under a separably inducible assump-
tion) with respect to solving the CollabLP problem.

In the thesis, the DIR approach to the CollabLP problem is not only theoretically analyzed but also empirically evaluated using multi-agent implementations of two well-known problems: distributed path planning and collaborative network fault diagnosis.

Experiments demonstrate the effectiveness of the DIR approach for overcoming the restrictions of distributed knowledge while avoiding the need for centralization. Empirical results have shown promise of the new approach for significantly reducing inter-agent communication while enhancing collaboration and improving network fault tolerance, when compared with competitive distributed strategies that invoke multiple (separate) instances of resolution.
Declaration

This is to certify that:

(i) the thesis comprises only my original work towards the PhD except where indicated in the Preface,

(ii) due acknowledgement has been made in the text to all other material used,

(iii) the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Signed,

_____________________________________________________________________

Jian Huang

12th March 2009
Preface

The content of this thesis comprises only my original work which was conducted solely during my PhD candidature and has not been submitted for any other qualifications.

Parts of this thesis have been extracted and published in various venues in collaboration with my supervisor, Dr. Adrian R. Pearce, as noted below:


In all the above works, I developed most of the ideas, conducted all of the
experiments and written most of the contents, while Adrian was actively involved in the discussion, verification, argumentation and editing of these works.

Jian Huang
12th March 2009
This thesis would not have existed without the tremendous support, direct or indirect, of many, to whom I am strongly obliged and would like to acknowledge with my most immense gratitude.

First of all, I am deeply indebted to my supervisor Dr. Adrian Pearce, who has not only been an invaluable source of knowledge and wisdom but also a patient mentor, advisor, audient and discusser and even, from time to time, a meticulous proofreader and spelling checker. It is difficult to imagine a higher level of patience, guidance and support one could receive.

I would also like to extend my sincere gratitude to two other resourceful references of mine, Prof. Leon Sterling and Dr. James Bailey, from whom I constantly receive assistance, encouragements, motivations, inspirations as well as challenges. In particular, I thank Prof. Leon Sterling for his careful proofreading of the final draft of this thesis.

This journey in the pursuit of truth would have undoubtedly been more arduous without the generous financial support that I have received over the long period of time from both the University of Melbourne and NICTA – Australia’s ICT Centre of Excellence. I take this opportunity to acknowledge their kind support, in the forms of two scholarships and a number of travel allowances.

Moreover, I would like to thank Mohammed Arif, Michelle Blom, Peter...
Hebden, Ryan Kelly, Bin Lu, Andrea Luo, Pedro Tao and Vincent Zhou for always being around to share my excitement as well as boredom and for making the working place a warm environment and a memorable experience.

Finally, I would like to express my absolute appreciation to my beloved parents who have supported me in every possible way during my studies. I also thank my wife, Kathy, for her incredible understanding, tolerance and sacrifice throughout this journey. The same gratitude also goes to many of my close relatives and friends, whose continuous belief, expectation and spiritual support have sustained me for traveling thus far.

Jian Huang
12th March 2009
## Contents

Abstract iii

Declaration v

Preface vii

Acknowledgements ix

1 Introduction 1
   1.1 Aim and Scope ............................................. 1
   1.2 The CollabLP Problem ...................................... 3
   1.3 The DIR Approach to CollabLP ............................ 4
   1.4 Thesis Contributions ...................................... 6
   1.5 Thesis Structure .......................................... 7

I Background 11

2 Background 13
   2.1 Overview .................................................... 13
   2.2 Deductive and Inductive Logic Programming .............. 13
   2.3 The Logic for Epistemic Reasoning ....................... 23
### CONTENTS

2.4 Inductive Learning in Multi-Agent Systems ............................................. 27

3 Literature Review .......................................................................................... 33

3.1 Overview ...................................................................................................... 33
3.2 Integration of Deduction and Induction ....................................................... 34
3.3 Collaborative Problem Solving .................................................................... 37
3.4 Induction in Distributed Settings ................................................................. 42

II Theory ............................................................................................................ 47

4 The CollabLP Problem ..................................................................................... 49

4.1 Overview ...................................................................................................... 49
4.2 Preliminaries and Notation ......................................................................... 50
4.3 The Basic CollabLP Problem ...................................................................... 51
4.4 Sorting Example: Basic CollabLP Case ...................................................... 52
4.5 The Generalized CollabLP Problem ............................................................. 54
4.6 Sorting Example: Generalized CollabLP Case .......................................... 55
4.7 Distributed Path Planning Example ............................................................ 57
4.8 Summary ...................................................................................................... 58

5 The DIR Framework ....................................................................................... 61

5.1 Overview ...................................................................................................... 61
5.2 Interleaving Deduction and Induction .......................................................... 64
5.3 Semantics of Deductive-Inductive Inferences ............................................. 66
5.4 $\Sigma_2$ Inference and RichProlog ................................................................. 69
5.5 Deductive and Inductive Resolvents ............................................................ 71
5.6 Deductive-Inductive Resolution Example .................................................. 73
5.7 Deductive-Inductive Resolution with Collaboration ................................... 76
5.8 Collaboration and Relationship to CollabLP .............................................. 77
### CONTENTS

#### 6  DIR from a Modal Perspective
- 6.1 Overview .................................................. 83
- 6.2 The Universe Structure ..................................... 84
- 6.3 Representing Collaborative Inference ...................... 86
- 6.4 Compressing the Universe ................................... 87
- 6.5 Soundness of Deductive-Inductive Resolution ............ 90
- 6.6 Completeness of Deductive-Inductive Resolution ....... 91

#### III  Practice

#### 7  Application: Distributed Path Planning
- 7.1 Overview .................................................. 100
- 7.2 Deductive Capability: Checking Reachability .......... 100
- 7.3 Inductive Capability: Hypothesizing a Path ............. 101
- 7.4 Interactive Capability: Collaborative Path Planning .... 103
- 7.5 Remarks on Communication Strategies .................. 107
- 7.6 Remarks on Alternative Approaches .................... 109
- 7.7 Experimental Results ...................................... 111

#### 8  Application: Network Fault Diagnosis
- 8.1 Overview .................................................. 115
- 8.2 Deductive-Inductive Diagnostic Procedure .............. 116
- 8.3 Knowledge-based Diagnostic Algorithm .................. 121
- 8.4 Remarks on Alternative Approaches .................... 122
- 8.5 Experimental Results ...................................... 124

#### 9  Conclusion
- 9.1 Summaries and Discussions ............................... 129
- 9.2 Contributions ............................................ 130
- 9.3 Directions for Further Research ......................... 131
## CONTENTS

A  Supplementary Experimental Details  133
   A.1 Overview ........................................ 133
   A.2 Output of the Sorting Example ................ 133
   A.3 Data on Network Fault Diagnosis .............. 134

Bibliography  139

Index  153
List of Figures

1.1 The thesis structure .................................................. 9
2.1 Inferential rules for inverse resolution .............................. 21
2.2 Hypothesis formation using the V-operators ....................... 23
4.1 A collaborative path planning scenario .............................. 57
5.1 A bi-directional resolution example ................................. 63
5.2 Inductive step using inductive resolution ......................... 75
6.1 The universe structure visualized as a directed graph. ........... 85
6.2 An extended universe structure for epistemic analysis ......... 86
6.3 Representing collaborative deductive inference .................. 87
6.4 A universe compression example .................................... 90
6.5 Proof of Completeness: when \( \mathcal{H} = \emptyset \) .................. 93
6.6 Proof of Completeness: when \( \mathcal{H} \neq \emptyset \) .................. 93
6.7 Proof of Completeness: when \( \mathcal{H} \) is separably inducible .... 94
7.1 Hypothesis formation for collaborative path planning .......... 102
7.2 Interaction during collaborative path planning .................. 106
7.3 Experimental results for collaborative path planning .......... 112
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>A collaborative fault diagnosis scenario</td>
<td>117</td>
</tr>
<tr>
<td>8.2</td>
<td>Interaction during collaborative fault diagnosis</td>
<td>119</td>
</tr>
<tr>
<td>8.3</td>
<td>Experimental results for collaborative fault diagnosis</td>
<td>127</td>
</tr>
<tr>
<td>A.1</td>
<td>Software interface of collaborative fault diagnosis</td>
<td>137</td>
</tr>
</tbody>
</table>
List of Tables

7.1 Deliberation during collaborative path planning . . . . . . . . 105
8.1 Deliberation during collaborative fault diagnosis . . . . . . . . 120
A.1 Sample output of the sorting example . . . . . . . . . . . . 135
A.2 Experimental data for collaborative fault diagnosis . . . . . . . 136
## List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Generic algorithm for finding hypothesis $H$</td>
<td>19</td>
</tr>
<tr>
<td>2</td>
<td>Algorithm for compressing the universe</td>
<td>88</td>
</tr>
<tr>
<td>3</td>
<td>Algorithm for hypothesizing paths through induction</td>
<td>104</td>
</tr>
<tr>
<td>4</td>
<td>Algorithm for collaborative path planning</td>
<td>108</td>
</tr>
<tr>
<td>5</td>
<td>Algorithm for collaborative fault diagnosis</td>
<td>123</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Aim and Scope

A great deal of work on learning in multi-agent settings has frequently employed multiple instances of induction separately, as opposed to learning that tightly integrates the processes of induction among agents (Stone & Veloso, 2000). These types of learning strategies often fail in complex domains because individual agents do not necessarily possess sufficient global knowledge of the environment, nor knowledge of other agents, resulting in system level behaviors that do not converge—as a result of uncoordinated learning happening in isolation.

This gives rise to not only the problem that no individual agent is capable of accomplishing the learning task alone any more but also the problem of knowing what knowledge needs to be communicated, given that sharing complete knowledge is often not feasible in such environments. Due to the above two constraints, neither of the the two extremes of collaboration scheme would work, i.e. learning in isolation or communicating everything.

According to Kazakov and Kudenko (Kazakov & Kudenko, 2001), the problem of true multi-agent learning has far more complexity than simply having each
agent perform localized induction in isolation or sharing everything. As Weiß and Dillenbourg have put, in those problems “interaction does not just serve the purpose of data exchange, but typically is in the spirit of a cooperative, negotiated search for a solution of the learning task” (Weiß & Dillenbourg, 1999).

Learning in multi-agent environments thus demands an approach that natively supports interaction, which tightly integrates not only the deductive and inductive processes within one agent, but among a group of agents as well.

Incorporating inductive capabilities into deductive systems has long been proven a useful strategy for a wide range of purposes (Shapiro, 1983; Flach, 1998; Jacobs, Driessens, & De Raedt, 1998; Martin, Nguyen, Sharma, & Stephan, 2002; Nanni, Raffaetà, Renso, & Turini, 2005). However, these systems have either been developed for very specific applications or do not target multi-agent environments, or both. RichProlog (Martin et al., 2002), for instance, is a promising approach that unifies deductive and inductive inferences under one logical framework on the basis of an alternation between compact and weakly compact consequences. However, RichProlog only handles queries of a restricted form and is not sufficient for solving collaborative problems in multi-agent settings in general.

On the other hand, works in collaborative problem solving domains concern the integration of isolated problem solving processes among distributed agents. Frameworks such as multi-agent answer set programming (Vos & Vermeir, 2004; Nieuwenborgh, Vos, Heymans, & Vermeir, 2007; Sakama & Inoue, 2008), for instance, have shown promise for collaborative execution of logic programs among interactive logic-based agents, through the communication of answer sets. However, typically these frameworks often assume a complete knowledge of the problem domain and are thus inadequate for problems necessarily requiring induction.

Recent progress has also been made towards distributing stand-alone inductive processes over multiple agents from both inductive logic programming and abductive logic programming disciplines (Huang & Pearce, 2006b; Ma, Russo,
1.2. THE COLLABLP PROBLEM

Broda, & Clark, 2008). However, these works often focus on dedicated learning systems and do not target general logic programming tasks.

This thesis overcomes the limitations of earlier research and presents a solution named deductive-inductive resolution that combines landmark deductive theorem proving (Kowalski & Kuehner, 1971) and inductive logic programming (Muggleton & De Raedt, 1994) techniques, for a wide range of multi-agent problems—namely collaborative logic programming (CollabLP) problems. Under deductive-inductive resolution (DIR), the induction process is no longer employed as a module separate to an agent’s main deductive reasoning process. Instead, the former is viewed as an integral component and natural extension of the latter such that an agent may switch to and from one form or the other when necessary. The DIR approach tightly integrates processes of deduction and induction among agents, through conservative communication that is limited to inductive hypotheses and deductive consequences.

1.2 The CollabLP Problem

Collaborative logic programming (CollabLP) involves solving logic programming tasks by a group of agents acting collaboratively as a single reasoning system, without sharing complete knowledge.

The CollabLP problem categorizes a class of multi-agent collaborative problems where typically:

(i) The global theory is distributed among a number of collaborative agents, such that each agent has part of the theory but not enough for any of them to solve the problem individually.

(ii) Agents are unable to reveal their internal theories directly. For example, this may result from privacy policies or communication restrictions due to bandwidth, power consumption, reliability or propagation considerations.
Agents interact with each other by issuing (and answering) queries, and not through any other means.

The agents’ combined theory may be insufficient for solving the problem without some hypotheses being generated (necessarily requiring induction).

To help understand the CollabLP problem and its constraints, imagine a number of pirates searching for some buried treasure. Each of them has part of the clue, encoded as logic programs, which will give the location of the treasure once executed. Being inspired by the common goal, i.e. to execute the programs and to find the treasure, the pirates desperately want to collaborate but none of them is willing to reveal his entire part to the rest. Worse still, some fragments of the program have gone missing so that they have to be induced based on the rest of the program and/or some guesses.

In essence, the CollabLP problem captures the fact that knowledge is distributed, private and (possibly) incomplete. Communication is allowed among the agents but restricted syntactically to be in the form of simple logic programming queries. In this sense, the CollabLP formulation captures a wide range of logic programming problems in multi-agent systems, whether or not involving induction.

1.3 The DIR Approach to CollabLP

This thesis develops an approach to the CollabLP problem, namely the deductive-inductive resolution (DIR) approach, based on an integration of both deductive and inductive logic programming techniques.

Reasoning that combines the two forms of inquisition, deductive and inductive, is ubiquitous in daily lives. We shall first consider a real life scenario which demonstrates this form of integration in action.

Imagine a real life situation where the bookshelf in your study sud-
1.3. THE DIR APPROACH TO COLLABLP

denly starts to shake wildly. What immediately comes into your mind maybe ‘the books are going to fall off’. This reasoning step is deductive. You probably would also draw a conclusion such as ‘it is an earthquake’. This reasoning step is not deductive in nature, but inductive or abductive. Based on this hypothesis, you decide that ‘I need to leave the building’, which is again, deductive.

The last reasoning step deserves a closer look. Notice that the conclusion that ‘I need to leave the building’ does not follow from the original observation that the bookshelf is shaking, but follows from the hypothesis made out of the previous reasoning step—‘it is an earthquake’. Therefore, the second and third reasoning steps can be viewed together as an atomic “inductive-deductive” reasoning step.

As can be seen, human agents are capable of interleaving both forms of reasoning in a truly seamless fashion, and so should artificial agents. This way of switching between deductive and inductive reasoning processes can yield complex reasoning scenarios, which requires an approach that natively supports the integration of the two.

The deductive-inductive resolution (DIR) approach provides a new paradigm for multi-agent programming that integrates both forms of reasoning. The DIR approach abstracts away the details of the actual deduction and induction algorithms employed but focuses on the integration of the two, from an agent’s perspective, which allows simple queries to be recursively expanded into potentially more complex forms, via a set of elementary inferential relations. This corresponds to recursive applications of deductive and inductive inferences and effectively results in a bi-directional traversal of the resolution tree of a logic program.

On the basis of the underlining deductive-inductive inferencing mechanism, the DIR approach also provides effective mechanism to support inter-agent com-

---

1 The distinction between inductive and abductive reasoning will be elaborated later in this thesis.
CHAPTER 1. INTRODUCTION

communication, through the use of inductive hypotheses and deductive consequences. In the DIR approach, induction is used not only as a supplement to deductive reasoning but also as an alternative to communication, while enabling inductive processes among agents to be interconnected and local knowledge to be shared (refer to Section 5.7). This opens the possibility of (i) better integration of induction during the execution of logic programs by multiple agents, and (ii) demand-driven communication, only when truly required.

The DIR framework is also assigned a semantics based on the possible-world structure. Besides allowing an epistemic analysis of agents during inferencing, this also enables some theoretical results of DIR to be established. Based on the possible-world semantics, the DIR approach is subsequently proven to be sound (in general) and complete (under the separably inducible assumption) with respect to solving the CollabLP problem.

1.4 Thesis Contributions

Specifically, this thesis makes the following contributions:

- A formal definition of the collaborative logic programming (CollabLP) problem, which captures not only problems involving learning in multi-agent environments, but also deductive problems requiring collaboration in general;

- A new, bidirectional deductive-inductive resolution (DIR) approach for solving instances of the CollabLP problem;

- An modal treatment of the DIR approach, based on which DIR is proven to be sound and complete (under the separably inducible assumption) with respect to solving the CollabLP problem, and

- Experimental evaluations of two applications that both illustrate solutions to instances of the CollabLP problem and empirically demonstrate the ad-
1.5. THESIS STRUCTURE

Vantages of the DIR approach, such as for avoiding centralization, reducing inter-agent communication and enhancing routing accuracy.

This thesis demonstrates, through the use of selected illustrative examples, the applicability of the new approach to a broad range of problems. For the distributed path planning problem, experimental results have shown promise for DIR in reducing communication when compared to (multiple instance of) single agent-based induction over varying distributions of data. When applied to network fault diagnosis—as an extension to existing routing techniques—experiments demonstrate that the diagnostic approach based on DIR is effective in improving network fault tolerance and responsiveness with only a moderate computational overhead.

1.5 Thesis Structure

This thesis is organized as follows:

Chapter 2 provides background on a number of areas, especially the large body of research on inductive logic programming and its techniques. It then provides the basics on modal and epistemic logic which paves the way for understanding some of the results presented in Chapter 6. It also describes the difficulty of inductive learning when extended from single- to multi-agent paradigm. This chapter may be referred back to while reading the remaining parts of the thesis, or skipped over entirely for readers familiar with these research domains.

Chapter 3 identifies research works in closely related areas and describes how this thesis is positioned among those works. This chapter first surveys existing attempts to combine deductive and inductive logic programming, such as, RichProlog. It then reviews existing logic programming approaches for collaboration in multi-agent environments. Some notable examples based on multi-agent answer set programming are covered. This chapter also describes some recent efforts in extending stand-alone inductive processes to distributed settings.
CHAPTER 1. INTRODUCTION

Chapter 4 defines the CollabLP problem and uses illustrative examples to show how it captures a wide range of collaborative problems in multi-agent environments. It does this in two steps. A basic, or simplified, version of the problem is presented first before progressing towards the more general problem definition, in which not only collaboration but also induction are dealt with.

Chapter 5 introduces the core of the deductive-inductive resolution (DIR) framework. The five elementary inferential relations are defined, identifying the elementary inference scenarios which can be used as building blocks for more complicated deductive-inductive inferencing scenarios. The DIR formalism is then extended to multi-agent setting where interaction and collaboration among agents are incorporated, through the sharing of deductive consequences and inductive hypotheses. This chapter concludes with a high-level outline of the relationship between the DIR framework and the CollabLP problem.

Chapter 6 presents an alternative perspective of the DIR approach based on modal logic. A Kripke structure is defined, which not only allows for an epistemic analysis of agents during deductive-inductive resolution, but also enables the establishment that DIR is both sound and complete with respect to solving the CollabLP problem. This chapter may be skipped without affecting the understanding of the remaining chapters.

Chapter 7 and Chapter 8 demonstrates the DIR approach when applied in two practical real life problems of distributed path planning and collaborative network fault diagnosis. These two chapters also detail the experimental investigation conducted as well as present empirical results when compared with various competitive approaches. These two chapters may be read in any order.

Figure 1.1 outlines the structure of the thesis and the dependencies between its chapters.
1.5. THESIS STRUCTURE

Figure 1.1: The thesis structure: outlining the flow and the dependencies between the chapters. Chapter 2 may be skipped over for readers familiar with the relevant research domains. Chapter 6 can be skipped without affecting the understanding of the remaining chapters. Chapter 7 and Chapter 8 may be read in any order.
Part I

Background
Chapter 2

Background

2.1 Overview

This chapter provides the necessary background material for understanding the upcoming chapters. This chapter first contrasts the two forms of reasoning, deductive and inductive, identifying their respective instantiations in logic programming, especially the large body of research on inductive logic programming and its techniques. Some basics on modal and epistemic logic is then provided, which is necessary for the technical results presented in Chapter 6. Finally, this chapter provides some background on the issues in relation to inductive learning in multi-agent settings. In particular, it highlights the intrinsic difficulties and restrictions imposed when extending inductive learning from single- to multi-agent paradigm.

2.2 Deductive and Inductive Logic Programming

2.2.1 Deductive and Inductive Reasoning

The two forms of reasoning, deductive and inductive, are often viewed as duals of each other. Among the two forms of reasoning, the former is better under-
stood and developed. It has been an area of philosophical investigation ever since Aristotle’s time, in the form of categorical syllogism (Aristotle, 1989). What deductive reasoning systems have in common, from the ancient Euclidean geometry to the modern Prolog programming language, is they all start from a finite number of propositions that are believed to be correct, called axioms. In Euclidean geometry, they are the five postulates and in Prolog, they make up the logic program. Deductive reasoning is then the process of deriving subsequent propositions following some sound reasoning steps.

In spite of being a sound form of reasoning, deductive reasoning suffers from the limitation of not being able to reason about generalized properties. In other words, everything that can be reasoned about for being true or false has to be derivable from these axioms. There is no space for reasoning beyond those axioms. For instance, if we know that swan A is white, swan B is white and etc. If we also know that X is a swan, what can we say about its color? Prolog will have absolutely no idea because this type of reasoning, which humans are very comfortable doing, is not deductive but inductive in nature.

Being able to generalize and establish clausal relationships from co-occurring events is a natural capability to human (and animals) and is thus believed to be an important aspect of intelligence (Russell, 1912, §6). Inductive reasoning also lies at the heart of scientific discoveries, as scientific discoveries are conjectural and hypothetical in nature and often involve inferring general rules from finite observations.

However, unlike its deductive counterpart, inductive reasoning does not have a truth-preserving nature, in a sense that the inductive conclusions may still be false even though the premisses are all true. If every marble taken from a bag so far has been black, it is tempting to anyone to jump into the conclusion that such bag contains black marbles. Hume first formulated this problem, which was later known as the ‘problem of induction’, and argued that the supposition that the future resembles the past is not founded on any logical arguments but is derived entirely
2.2. DEDUCTIVE AND INDUCTIVE LOGIC PROGRAMMING

from habit (Hume, 1772, §2). No matter how many instances of white swans have been observed, it does not confirm the general statement that every swan is white. Popper attempted this problem and developed his philosophy of science based on the principle of falsifiability, under which he claims that scientific theories can never be proven. A theory remains tentatively true until it is falsified (Popper, 1959, §1).

There are ongoing discussions in the literature on distinguishing different forms of inductive reasoning, and whether they should be called ‘inductive’, ‘abductive’ or ‘non-deductive’ reasoning in general. According to Lachiche (Lachiche, 2000), inductive reasoning can be further classified into descriptive (or inductive) and explanatory (or abductive). The former is often understood as inferring general rules from specific data. Examples include the swan example and the marble example. For instance, given that all swans have been seen so far are white, we infer that ‘all swans are white’.\(^1\) Abduction, on the other hand, is often understood as reasoning from effects to causes. A doctor, for example, performs abduction when she infers that a patient has cavity in the teeth given he has got a toothache and the fact that he eats a lot of sweets and never brushes his teeth. Similarly, a detective infers abductively from the given evidences that the murderer must have entered the room from the window. Scientists too perform abduction: all phenomena in nature can be explained well if we assume the earth is round, not flat.

Much work have been done to contrast the two forms of inductive reasoning, against syntactic forms or based on model theories (Denecker, Martens, & De Raedt, 1996; Denecker & Kakas, 2002). Equally large amount of work are done to bring them together (Mooney, 1997; Lamma, Mello, Milano, & Riguzzi, 1999). This thesis takes no position in the discussion but refer the readers to (Flach & Kakas, 1996, 1997, 1998, 2000). In this thesis, the word ‘inductive’ is used in the wide sense to mean ‘non-deductive’ in general, which captures both

\(^1\)Replace ‘white’ by ‘black’ if one happens to live in Australia.
CHAPTER 2. BACKGROUND

common abductive reasoning strategies while opening the possibility to capture more general inductive learning tasks.

Whichever of the two forms inductive reasoning may take, the essence of it is to construct a hypothesis \( H \) such that the evidences \( E \) can be explained, together with some existing background theory \( T \). In logic notation, the aim of inductive reasoning is to find \( H \) such that \( T \land H \models E \).

2.2.2 Logic Programming and SLD-Resolution

Deductive and inductive reasoning have their respective instantiations in logic programming, in which logic formalisms have been used as the language for computing, learning and problem solving.

Given program \( T \) and query \( \varphi \), logic programming aims at answering if \( \varphi \) is logically entailed by \( T \), i.e. if \( T \models \varphi \). A logic programming system can often be broken down into two aspects: representation and proof procedure. Prolog, the best known logic programming system for instance, uses first-order Horn clauses as its representation (for both \( T \) and \( \varphi \)) and SLD-resolution for efficient theorem proving. In this subsection, a brief description is provided about Prolog (Colmerauer & Roussel, 1996) and the proof procedure it is based on, i.e. SLD-resolution (Kowalski & Kuehner, 1971).

Although the decision problem of whether \( T \models \varphi \) is undecidable (or semi-decidable) for first-order logic in general (Church, 1936; Turing, 1937), the resolution technique (Robinson, 1965) provides a sound and complete proof procedure which guarantees a proof whenever \( T \models \varphi \) is indeed the case.

Resolution in propositional logic involves deriving clause \( C \) from two clauses \( C_1 \) and \( C_2 \), where \( C_1 \) contains the literal \( l \) and \( C_2 \) contains the literal \( \neg l \). The resulting clause \( C \), called the resolvent, is then defined according to the following rule:

\[
C = (C_1 \setminus \{l\}) \cup (C_2 \setminus \{\neg l\})
\] (2.1)
2.2. DEDUCTIVE AND INDUCTIVE LOGIC PROGRAMMING

Resolution in first-order logic is defined in a similar fashion but requires an additional unification step. In first-order case, Equation (2.1) becomes Equation (2.2), where $\theta_1 \theta_2$ is the mgu of $\neg l_1$ and $l_2$ such that $\neg l_1 \theta_1 = l_2 \theta_2$:

$$C = (C_1 \setminus \{l_1\})\theta_1 \cup (C_2 \setminus \{l_2\})\theta_2$$  \hspace{1cm} (2.2)

The resolution step can be recursively applied to construct a derivation defined as follows:

**Definition 1** (Derivation). Let $T$ be a set of clauses and $\varphi$ a clause. A derivation of $\varphi$ from $T$ is a finite sequence of clauses $R_0, \ldots, R_k = \varphi$, such that each $R_i$ is either in $T$, or a resolvent of two clauses in $\{R_0, \ldots, R_{i-1}\}$.

SLD-derivation is then a restricted form of derivation in two ways. First, it restricts the language (for both $T$ and $\varphi$) to be Horn clauses. Second, SLD-derivation requires every $R_i$ to be a resolvent of the previous resolvent $R_{i-1}$ and a clause taken directly from $T$, hence a form of linear and input resolution. SLD-derivation is defined as follows:

**Definition 2** (SLD-derivation). Let $T$ be a set of Horn clauses and $\varphi$ a Horn clause. An SLD-derivation of $\varphi$ from $T$ is a finite sequence of Horn clause $R_0, \ldots, R_k = \varphi$, such that $R_0$ is in $T$ and each $R_i$ ($i > 0$) is a resolvent of $R_{i-1}$ and a clause $T' \in T$.

Due to these restrictions, SLD-resolution is more efficient than the unconstrained resolution and, meanwhile, still enjoys the property of being sound and refutation-complete (Kowalski, 1974; Lloyd, 1987), unlike other input resolution techniques in general.

Prolog uses SLD-resolution with ‘negation as failure’ to establish refutation (Apt & van Emden, 1982). When given a program $T$ and a query $\varphi$, both in the form of Horn clauses, what Prolog does in essence is to perform a depth-first search to find if there exists an SLD-derivation of the empty clause $\Box$ from $T \cup \{\neg \varphi\}$. If the SLD-tree is finite, Prolog succeeds iff $T \models \varphi$. 

17
CHAPTER 2. BACKGROUND

2.2.3 Inductive Logic Programming and Inverse Resolution

From a machine learning perspective, inductive logic programming (ILP) overcomes two major limitations associated with other inductive learning techniques, such as decision tree learning, neural network and reinforcement learning. First, ILP naturally supports the utilization of substantial background knowledge in the learning process. Second, it allows for knowledge to be represented using an expressive formalism, i.e. in first-order Horn clauses, and is thus compatible with many logic programming techniques. An ILP problem is generally formulated as follows (Muggleton, 1999):

**Definition 3** (Inductive Logic Programming). Given theory (background knowledge) \( T \), positive examples \( E^+ \) and negative examples \( E^- \), represented as logic formulae, the aim of ILP is to find a hypothesis \( H \) such that the following conditions hold:

1. Necessity: \( T \not\models E^+ \)
2. Sufficiency: \( T \land H \models E^+ \)
3. Weak Consistency: \( T \land H \not\models \Box \)
4. Strong Consistency: \( T \land H \land E^- \not\models \Box \)

In the definition of ILP problems, the *necessity* condition captures the idea that the theory alone is insufficient to explain the positive examples and the *sufficiency* condition states that the (induced) hypothesis must entail the positive examples. *Weak consistency* ensures that the hypothesis must be consistent with the theory and *strong consistency* ensures that the hypothesis must not cover the negative examples. The strong consistency condition is often relaxed for practical and/or efficiency purposes.

ILP consequently concerns techniques for constructing the hypothesis, \( H \), systematically and efficiently. Like deductive theorem proving techniques, a wide
2.2. DEDUCTIVE AND INDUCTIVE LOGIC PROGRAMMING

Algorithm 1 Generic algorithm for finding hypothesis $H$.

Input: $T$, $E^+$ and $E^-$.

Output: $H$.

1: Start with some initial (possibly empty) $H$.

2: repeat

3: if $T \land H$ is too strong then

4: specialize $H$.

5: end if

6: if $T \land H$ is too weak then

7: generalize $H$.

8: end if

9: until all four conditions are met.

10: return $H$.

A range of techniques have been developed for formulating the hypothesis. The generic algorithm for finding the hypothesis can be described as follows: if the hypothesis $H$ found so far is too strong (such that it covers not only the positive examples but also some negative examples), weaken it by specializing $H$; if it is too weak (such that it does not cover all positive examples), strengthen it by making it more general. Repeat until $H$ is just right. The generic algorithm for finding $H$ is specified in Algorithm 1 (Nienhuys-Cheng & de Wolf, 1997, §9).

Existing approaches on inductive hypothesis formation are based on either generalization or specialization. Generalization techniques search the hypothesis space from the most specific clauses until it cannot be further generalized without covering negative examples. Generalization techniques include relative least general generalization (RLGG) (Plotkin, 1969)—as implemented by Golem (Muggleton & Feng, 1992)—and inverse resolution—as implemented by Cigol (Muggleton & Buntine, 1988). In (Muggleton & De Raedt, 1994), it has been shown that inductive inference can be done by inverting resolution backwards.
CHAPTER 2. BACKGROUND

from the existing theorems and examples using a number of inductive inference rules. Most specialization techniques are based on top-down search of refinement graph—as implemented by FOIL (Quinlan, 1990). The inverse entailment technique (Muggleton, 1995) is later proposed which is a more fundamental approach than inverse resolution, as inverse entailment is based on model theory instead of inverting proofs. Inverse entailment is implemented by Progol and its successor Aleph (Srinivasan, 2001).

The following paragraphs provide a brief description of inverse resolution (Muggleton & Buntine, 1988) as one fundamental ILP technique, which will be used in later parts of the thesis to illustrate hypothesis formation examples.

Since inductive logic programming is often viewed as an inverse of (deductive) logic programming, it is not surprising that the former can be performed by inverting operators of the latter. As resolution (Robinson, 1965) is one powerful technique for deductive theorem proving which provides a basis for most logic programming systems, inverse resolution explores the inverse operation of it, hence the name ‘inverse’ resolution.

The following set of inferential rules have been defined for inverse resolution in propositional logic (Muggleton & De Raedt, 1994):

**Absorption**:

\[
\begin{align*}
q &\leftarrow A \\
p &\leftarrow A, B
\end{align*}
\]

\[
\begin{align*}
q &\leftarrow A \\
p &\leftarrow q, B
\end{align*}
\]

**Identification**:

\[
\begin{align*}
p &\leftarrow A, B \\
p &\leftarrow A, q
\end{align*}
\]

\[
\begin{align*}
q &\leftarrow B \\
p &\leftarrow A, q
\end{align*}
\]

**Intra-Construction**:

\[
\begin{align*}
p &\leftarrow A, B \\
p &\leftarrow A, C
\end{align*}
\]

\[
\begin{align*}
q &\leftarrow B \\
p &\leftarrow A, q \\
q &\leftarrow C
\end{align*}
\]

**Inter-Construction**:

\[
\begin{align*}
p &\leftarrow A, B \\
q &\leftarrow A, C
\end{align*}
\]

\[
\begin{align*}
p &\leftarrow r, B \\
r &\leftarrow A \\
q &\leftarrow r, C
\end{align*}
\]

Each inference rule inverts a single-step application of resolution, as given in Equation (2.1). By applying these set of rules, theories (the leaves) can be con-
2.2. DEDUCTIVE AND INDUCTIVE LOGIC PROGRAMMING

Figure 2.1: Inferential rules for inverse resolution. LEFT: Absorption and Identification are collectively known as the V-operators. RIGHT: Intra- and Inter-construction are collectively called the W-operators.

structured backwards from the examples (the root). These rules can be visualized as in Figure 2.1. Because of the appearance of their resolution trees, Absorption and Identification are collectively referred to as the V-operators. Absorption involves deriving $C_2$ given $C$ and $C_1$ while Identification involves deriving $C_1$ given $C$ and $C_2$. In both cases, $C_1$ contains the positive literal $l$ and $C_2$ contains the negative literal $\neg l$. Intra- and Inter-construction are collectively referred to as the W-operators. Both Intra- and Inter-construction derive $C_1$, $C_2$ and $A$ given $B_1$ and $B_2$. In Intra-construction, $C_1$ and $C_2$ contain the positive literal $l$, and $A$ contains the negative literal $\neg l$. The case for Inter-construction is exactly the opposite. Resulting from the W-operators, new proposition symbols not found in the examples are effectively ‘invented’.

In (Muggleton & Buntine, 1988), inverse resolution has been extended to first-order logic. Recall that resolution involving first-order predicate logic requires unification, as given in Equation (2.2). Since $\neg l_1 \theta_1 = l_2 \theta_2$, thus $l_2 = \neg l_1 \theta_1 \theta_2^{-1}$, Equation (2.2) can be rearranged to obtain $C_2$ from $C$ and $C_1$ for Absorption:

$$C_2 = (C - (C_1 \setminus \{l_1\}) \theta_1 \cup \{\neg l_1\} \theta_1 \theta_2^{-1}} \quad (2.3)$$

Because the least general $C_2$ occurs when $\theta_2$ is empty and $C_1$ is minimal, i.e. $C_1 = \{l_1\}$, Equation (2.3) can be simplified to obtain the least general $C_2$,
CHAPTER 2. BACKGROUND

denoted \( C_2 \downarrow \), shown in Equation (2.4). Similarly, if we replace all subscripts in Equation (2.4), we obtain the Identification rule for finding the least general clause \( C_1 \downarrow \) in Equation (2.5).

\[
C_2 \downarrow = (C \cup \{ \neg l_1 \}) \theta_1 \\
C_1 \downarrow = (C \cup \{ \neg l_2 \}) \theta_2
\] (2.4) (2.5)

In (Muggleton, 1992), Muggleton also showed the equivalence of Plotkin’s notion of RLGG (Plotkin, 1969) and the least general inverse derivation resulted from iterative applications of Absorption and Identification.

In the remainder of this subsection, a reachability example is used to illustrate hypothesis formation using the V-operators, given

\[
E = \text{reachable}(a, c) \\
T_1 = \text{reachable}(a, b) \\
T_2 = \text{reachable}(A, C) \leftarrow \text{reachable}(A, B) \land \text{reachable}(B, C)
\]

As in logic programming convention, capital letters are used to denote free variables and lower-case letters bound. The term \( \text{reachable}(a, b) \) stands for ‘\( b \) is reachable from \( a \)’. \( E \) can be viewed as the example to be explained and \( T = T_1 \cup T_2 \) is the background theory defining the known reachability as well as the transitive nature of the reachability relation. Figure 2.2 shows the hypothesis formation process in two steps. The first step is an Absorption step. \( C_2 \) is in fact the least general generalization of \( E \) and \( T_1 \), obtained by a direct application of Equation (2.4). The second step is an Identification step with unification. For \( C_1 \), however, there are many possible alternatives. The least general clause, \( C_1 \downarrow \), is \{\text{reachable}(b, c), \neg \text{reachable}(a, b), \neg \text{reachable}(a, c)\}. The most general one is \{\text{reachable}(b, c)\}, as shown in the figure. The actual \( C_1 \) chosen depends on the implementation and application. Any such \( C_1 \) is an inductive hypothesis, with which the theory (\( T \)) entails the example (\( E \)), i.e. \( T \land C_1 \models E \).
2.3. THE LOGIC FOR EPISTEMIC REASONING

Figure 2.2: Example showing the use of the V-operators in forming the hypothesis $C_1$ based on example $E$ and theory $T = T_1 \cup T_2$, such that $T \land C_1 \models E$. The first step uses the Absorption operator (to get $C_2$) whereas the second step uses the Identification operator (to get $C_1$) (see Figure 2.1 LEFT). Note the choice for $C_1$ in this case is not unique.

2.3 The Logic for Epistemic Reasoning

2.3.1 The Possible-World Semantics

Classical logic suffers from the property of extensionality (van der Hoek, 2001) which makes it undesirable for modeling many reasoning constructs that are not extensional, e.g. causal effects and motivational attitudes. This is what modal logic attempts to circumvent. In a nutshell, modal logic extends classical logic by introducing one or more unary operators $\Box$ to the language, where $\Box \varphi$ can be used to model ‘$\varphi$ is known’, ‘$\varphi$ is always the case’, ‘$\varphi$ is a desire’ or ‘$\varphi$ is a result of executing program $\pi$’, etc.

Since (Hintikka, 1962), the semantics of the $\Box$ operator is often defined based on the possible-world structure, or Kripke structure (Kripke, 1963). A Kripke structure $M$ is typically in the form of an n-tuple, $M = (S, \pi, R_1, \cdots, R_n)$, where

- $S$ is a set of states;
• \( \pi \) is called the interpretation which associates with each state in \( S \) a truth assignment of primitive propositions in the language \( \mathcal{L} \), i.e. \( \pi : (S, \mathcal{L}) \mapsto \{ \text{true} | \text{false} \} \)

• \( R_i \) is a set of binary relations over \( S \). \((s, t) \in R_i\) if and only if \( t \) is accessible from \( s \).

If \( s \) is the actual true state then, for all \( t \) that \((s, t) \in R_i\), \( t \) is viewed as the alternative possible state. The formula \( \square \varphi \) is subsequently defined to be true in a model \( M \) and a state \( s \), written \( M, s \models \varphi \), if \( M, t \models \varphi \) for all \( t \) accessible from \( s \). The \( \lozenge \) operator is defined as the dual of the \( \square \) operator such that \( \lozenge \varphi = \neg \square \neg \varphi \).

This possible-world semantics turns out to be ideal for representing epistemic aspects of a reasoning agent. The intuition behind the approach is that besides the actual state of affairs, there could be a number of alternative states of affairs which is indistinguishable to an agent and are considered as the possible states of affairs by the agent. Based on this model, an agent is said to know a fact \( \varphi \) if \( \varphi \) is true in all worlds the agent considers possible.

Many formalization of epistemic logic, or logic of knowledge, are based on modal logic and the possible-world structure. When modeling agent’s mental status, conventionally the modal operators \( K_i \) (in replace for \( \square_i \)) and \( B_i \) (in replace for \( \lozenge_i \)) are used to denote knowledge and belief, where \( K_i \varphi \) and \( B_i \varphi \) respectively stand for agent \( i \) knows about \( \varphi \) and agent \( i \) believes \( \varphi \).

### 2.3.2 Axiomatization of Epistemic Logic

Works on axiomatizing the logic of knowledge has been extensive. The following gives a list of the most commonly seen axioms for epistemic logic systems.

A1 All tautologies of propositional calculus

A2 \((K_i \varphi \land K_i (\varphi \Rightarrow \psi)) \Rightarrow K_i \psi\)

A3 \(K_i \varphi \Rightarrow \varphi\)
2.3. THE LOGIC FOR EPISTEMIC REASONING

**A4** \[ K_i \varphi \Rightarrow K_i K_i \varphi \]

**A5** \[ \neg K_i \varphi \Rightarrow K_i \neg K_i \varphi \]

**R1** From \( \varphi \) and \( \varphi \Rightarrow \psi \) infer \( \psi \)

**R2** From \( \varphi \) infer \( K_i \varphi \)

**A1** and **R1** are obviously carried over from classic propositional logic. **A2** is called the Distribution Axiom, which asserts that an agent's knowledge is closed under implication. **A3** is referred to as the Knowledge Axiom (or Veridicality Axiom), which corresponds to the natural understanding of what ‘knowing something’ means. That is, when an agent is said to know something, that thing is necessarily true, otherwise it is a mere belief. **A4** and **A5** are the Positive and Negative Introspection Axioms respectively, which account for the aspects that an agent knows what it knows and what it does not know. **R2** is sometimes referred to as the Knowledge Generalization Rule, which says an agent knows about all tautologies.

The simplest axiomatic system for knowledge is the K system, which is a simple and direct extension from classical logic with the knowledge operator included. The K system consists of the axioms **A1** and **A2** as well as the derivation rules **R1** and **R2**. Axioms **A3**, **A4** and **A5** are then progressively added on top of the K system to form the T (\( =K+A3 \)), S4 (\( =T+A4 \)) and S5 (\( =S_4+A5 \)) axiomatic systems for various purposes.

As the epistemic reasoning power increases with the insertion of additional axioms, the agent gradually becomes omnisciently rational. For example, there are concerns about whether it makes sense for a resource bounded agent to know all valid formulae (as in **R2**), or to know what it doesn’t know about (as in **A5**). These unrealistic expectations of resource-bounded reasoning agents lead to the well-known logical omniscience problem (Hintikka, 1975). Although there is no consensus on which axiomatic system best captures the aspects of rationality,
CHAPTER 2. BACKGROUND

for many applications with bounded knowledge space, the system $S_5$ (axioms $A1$ to $A5$ plus derivation rule $R1$ and $R2$) seems appropriate for many practical purposes.

Various properties of these axiomatic systems have been proven to hold. For example, it turns out that these axioms also impose structural properties to the Kripke structure associated. The axiom $A3$, for instance, has the correspondence to structures that are reflexive, while $A4$ corresponds to structures that are transitive and $A5$ corresponds to euclidean (i.e. transitive and symmetrical) ones. All of $K, T, S_4$ and $S_5$ are shown to be sound and complete with respect to their respective classes of Kripke structures.

2.3.3 Epistemic Logic for Multi-Agent Systems

As it is becoming increasingly necessary and important to reason about not only what an agent knows about the status of the world, but also what it knows about other agents, the epistemic logic formalisms have subsequently been enriched to accommodate group knowledge for a team of agents. The language $L$ of the logic for a group of $m$ agents is extended to be

$$
\varphi, \psi \in L \Rightarrow \neg \varphi, (\varphi \land \psi), K_i \varphi, E\varphi, C\varphi, D\varphi \in L
$$

in which $E\varphi$ stands for ‘everybody knows that $\varphi$', $C\varphi$ stands for ‘it is common knowledge that $\varphi$’ and $D\varphi$ stands for ‘it is distributed knowledge that $\varphi$’. Epistemic logic has been successfully applied to the study of distributed systems (Halpern & Moses, 1990) and protocol verification (Halpern & Zuck, 1987) involving multiple agents.

Giving a broad and in-depth coverage of the technical results in the vast research area of epistemic logic is beyond the scope of this thesis. Readers are referred to (Halpern, 1995) for a survey paper and introductory texts such as (Fagin, 1995) and (Meyer & Hoek, 1995) for a comprehensive treatment of epistemic
logic. (Blackburn, de Rijke, & Venema, 2001) is a thorough introductory text and a good reference on modal logic.

2.4 Inductive Learning in Multi-Agent Systems

2.4.1 The Multi-Agent Paradigm

The research of multi-agent systems (MAS) concerns the study of interaction and coordination of homogenous or heterogeneous entities that are autonomous, goal-oriented and reactive to the environment they are situated in (Jennings, Sycara, & Wooldridge, 1998).

Recent advances in agent technology have witnessed an increasing amount of success of the multi-agent paradigm, in spite of the fact that every task that can be performed by a group of agents can potentially be performed by a well designed single agent. For example, a distributed constraint satisfaction problem (Yokoo, Durfee, Ishida, & Kuwabara, 1998) can be trivially solved by gathering all constraints into one leader agent which then executes a centralized constraint satisfaction algorithm.

According to Jennings et al. (Jennings et al., 1998), the multi-agent paradigm is often adopted because of its ability to: (i) provide robustness and efficiency; (ii) allow inter-operation of existing legacy systems; and (iii) solve problems in which data, expertise, or control is distributed.

Although there are results (e.g. (Sen, Sekaran, & Hale, 1994)) demonstrating that enabling interaction and collaboration among agents does not necessarily lead to a better performance at system level, but generally speaking, the true benefits of adopting multi-agent paradigm comes from the ability to combine and share a diversified range of resources, knowledge and expertise among the agents which facilitates a collaborative effort for problem solving.

The application of MAS has been proven useful in a wide range of areas in-
including manufacturing, task scheduling, information gathering, network management and as a new paradigm for software engineering (Chalupsky, Gil, Knoblock, Lerman, Oh, Pynadath, Russ, & Tambe, 2002; Bradshaw, 1997; Weiß, 1999; Wooldridge & Ciancarini, 2001). The advantages of structuring applications as MAS rather than as single agent system include: speed up due to concurrency, less communication due to local processing, higher reliability and responsiveness (Lesser, 1999).

### 2.4.2 From Single-Agent to Multi-Agent Induction

Multi-agent systems are complex and dynamic, in which it is often difficult to fully specify the behavior and knowledge of all agents at design stage. They therefore benefit by being equipped with the ability to actively improve their performance over time. Although extensive work has been done in learning from a single agent perspective, it is only until a decade ago that the needs to equip multi-agent systems with learning capabilities have been acknowledged. Examples include collections of papers in (Weiß, 1997; Weiß & Sen, 1996; Imam, 1996; Sen, 1996).

In spite of this, in the existing body of multi-agent learning literature, agents are typically modeled as 0-level or 1-level entities (according to Vidal and Durfee’s awareness classification model (Vidal & Durfee, 1997)). That is, agents are either not aware of the existence of other agents at all or are only able to predict the behavior of other agents through environmental feedback. As a result, it is often the case that learning techniques developed from a single agent perspective are directly applied to multi-agent situation and multi-agent learning are thus only viewed as an emergent property (Alonso, D’Inverno, Kudenko, Luck, & Noble, 2001).

Since these learning strategies often attempt to improve the global behavior through uncoordinated efforts made locally, they typically fail in multi-agent settings. Since then, many have begun to adopt the view that once the learning
2.4. INDUCTIVE LEARNING IN MULTI-AGENT SYSTEMS

process is distributed from single agent to a number of agents, current techniques need to be modified significantly and new techniques need to be invented (Weiß, 1996).

Weiß has classified multi-agent learning strategies into three main categories: multiplication, division and interaction (Weiß & Dillenbourg, 1999). In multiplication learning, each agent learns the global hypothesis independently with others. Agents interact with others only via perceiving the changes of the environment. The advantage of this type of learning mechanism is that existing single agent learning techniques can be easily applied without major modification, with the expense that a significant amount of learning effort is duplicated and that achieving optimal learning outcome is difficult.

In division learning, each agent learns a specific aspect of the hypothesis as if they collaborate on an assembly line. This type of approach is efficient in terms of both time and resource. However, this approach either requires an extra coordinator agent to split the work or requires the agents to negotiate the split among themselves. Likewise, individual hypotheses eventually need to be assembled into a global hypothesis in a similar fashion, which can be a nontrivial task.

In interaction learning, the agents learn their individual hypotheses or the global hypothesis collaboratively by exchanging knowledge and data with each other. Each individual agent’s learning process is affected (and improved) by the knowledge of other agents through close interaction with other agents.

Majority of the existing learning approaches are based on the multiplication strategy, according to Weiß’s classification. Although such learning strategy has been successfully applied to various learning problems (Stone & Veloso, 2000), learning based on multiple isolated instances of induction is insufficient in multi-agent settings in general.

On the other hand, the interaction strategy has been shown to yield much better learning outcome and it is widely accepted that in order to take full advantage of multi-agent system, learning with the aim of improving the system performance
as a whole would have to involve significant interaction among the participants.

2.4.3 Example: Inducing the Definition of Sort

To see why interaction and collaboration is inevitable during induction in multi-agent settings, consider the following example in a logic programming context:

Suppose agent $a_1$ requires a definition for the predicate $\text{min}(L, M)$ (for finding the minimum number in a list). It knows that if it can sort a list (in ascending order), then the first element will be the minimum of the list. However, agent $a_1$ doesn’t know how to sort a list (but it does have a list of positive and negative examples of sorted lists) so its knowledge about $\text{min}$ depends on another agent’s knowledge about $\text{sort}$. Suppose another agent $a_2$ knows how to generate permutations of a list and how to check the ordering of a list. Given that sorting can be performed by generating permutations and checking if the permutation is ordered, agent $a_2$ is already capable of performing sorting as long as information can be communicated from agent $a_1$.

In the above example, only agent $a_1$ knows what $\text{sort}$ means at the start, which can be viewed as agent $a_1$ knowing positive/negative examples of sorted lists along with the name of the predicate ($\text{sort}$). Although agent $a_2$ knows everything it needs to perform $\text{sort}$, it doesn’t know there exists such a thing called sorting. Nevertheless, it can induce the definition of $\text{sort}$, based on its theory and examples given by agent $a_1$. In other words, the two agents have to work in a collaborative manner in order to complete this inductive task.

There are, potentially, several different possible collaboration scenarios that may arise between agents during induction. In (Huang & Pearce, 2006a) it has been demonstrated that the first three (of the following four) scenarios can be han-
dled through the communication of only positive/negative examples (i.e. without the need for theory to be transferred).

(i) The simplest case is that agent $a_2$ already has the knowledge when agent $a_1$ asks for it. Therefore, it is only a matter of communication.

(ii) Alternatively, agent $a_2$ needs to induce a hypothesis based on positive and negative examples received from agent $a_1$ and its own theory.

(iii) Furthermore, agent $a_2$ may require agent $a_3$ to induce some extra knowledge first before it can induce the hypothesis required by agent $a_1$.

(iv) Finally, the theory required for inducing the hypothesis may even be distributed over different agents.

In summary, close interaction and collaboration among the agents during induction is often a prerequisite for successful learning outcomes.
3.1 Overview

In this chapter, this thesis is positioned amid research works in three related areas: (i) integrated deductive-inductive systems; (ii) logic based collaborative problem solving and (iii) inductive learning in distributed settings. If we visualize these three areas of research as three neighboring but non-overlapping circles, the empty region enclosed by them is where this thesis is positioned.

This research, however, does overlap with all above three areas of research to various extent. In fact, it can be viewed that this research largely brings together efforts made from these disjoint research areas, towards solving multi-agent logic programming and learning problems as defined in Chapter 4.

This chapter surveys each of the three areas of research in order, identifying the gap left amid them and how this thesis bridges the gap to overcome the limitations of the existing research.
3.2 Integration of Deduction and Induction

3.2.1 Deductive-Inductive Systems

Incorporating inductive capability into deductive systems has been proven useful for a wide range of purposes. For example, induction has long been successfully applied as tools for design, query processing as well as data mining in deductive databases (Dzeroski & Lavrac, 1993; Flach, 1998), and in pattern recognition and data analysis tasks (Nanni et al., 2005). Inductive extensions have also been applied to traditional logic programming systems in various different ways. For example, to assist traditional programming tasks such as verification and debugging (Jacobs et al., 1998; Shapiro, 1983), to assist high level planning tasks (Missiaen, Bruynooghe, & Denecker, 1995; do Lago Pereira & de Barros, 2004a) and to allow active acquisition of missing knowledge during deductive theorem proving (Huang & Pearce, 2006a). It has also been shown that inductive hypotheses are an effective mechanism for coming up with communication efficient solutions for deductive problems in a distributed fashion (Huang & Pearce, 2007).

Although there have been many promising works in bringing deductive and inductive reasoning together for various applications, incorporating one form of reasoning into the other is frequently an afterthought. This results in induction often being a module separate to an agent’s deductive reasoning process, as opposed to systems that have the two forms of reasoning tightly integrated.

There are recent endeavors on performing both deductive and inductive reasoning natively under one logic framework and attempts have been made to integrate the two forms of reasoning, from both theoretical and implementation perspectives.

Flach (Flach, 2000) argues for using logics to model the reasoning process of inductive inference, in a similar way it does in deductive inference. He claims that logic is the science of reasoning—not necessarily the science of correct reasoning—and deductive logic, which happens to have a nice truth-preserving feature, is just
3.2. INTEGRATION OF DEDUCTION AND INDUCTION

a special case. In this view, his work provides semantics and proof systems for rewriting inductive rules at a meta level. Inductive reasoning systems, in practice, can instantiate these meta-rules for specific applications.

Martin et al. (Martin, Sharma, & Stephan, 2001) provide a logical framework which attempts to unify the logics of deduction and induction. Their framework views interleaved deductive and inductive inference as an alternation between compact and weakly compact consequences. In their generalized logical consequence framework, deductive consequence is defined as “ϕ is a deductive consequence of theory T if it can be established using a finite subset of T, T′, that entails ϕ—where ϕ is true in every model of T′”. Inductive consequence is subsequently defined on the basis of deductive consequence as “ϕ is an inductive consequence of T, if the negation of ϕ, ¬ϕ, is not a deductive consequence of T′”. That is, ϕ is an inductive consequence unless it is known to conflict the theory. Thus, being a deductive consequence is by definition also an inductive consequence. In other words, those sentences that can be proven to contradict the theory are not admitted as generalized logical consequences. All the rest of the sentences are. Among them, there is a special class that can actually be proven—they are the deductive consequences.

3.2.2 RichProlog

RichProlog (Martin et al., 2002) is a promising recent approach to deductive-inductive logic programming that bases itself on the aforementioned generalized logical consequence theory. RichProlog is a logic programming system that joins together the processes of deductive theorem proving and inductive logic programming while maintaining the declarative nature of Prolog and facilitates the answering of a broader range of queries than Prolog. Given a generalized logic program, T, and an atomic formula, ϕ, all of whose free variables occur in the disjoint sequences of variables ¯x and ¯y, RichProlog determines whether ∃x∀yϕ is a generalized logical consequence of T. Whenever this is indeed the case, RichProlog
outputs a sequence of terms, \( \bar{t} \), of the same length as \( \bar{x} \) as a witness for \( \exists \bar{x} \forall \bar{y} \varphi \).

RichProlog allows for the integration of deduction and induction in one particular way. RichProlog answers queries in this particular format: Is there a pattern \( x \) that matches all individuals \( y \)? or \( \exists x \forall y \text{pattern}(x) \land \text{matches}(x, y) \). For example, what pattern do instances \( \text{aaa}, \text{aab}, \text{aba} \) and \( \text{abb} \) exhibit? The first part of the query involves hypothesizing \( x \), which can be viewed as an inductive task, while the second part involves proving that \( x \) indeed matches all \( y \), which is deductive. Moreover, RichProlog offers its own way to solve ILP problems since ILP problems in general can be formulated as: is there a hypothesis that logically entails all examples? As can be seen, this is a just an instantiation of the query that RichProlog handles. However, RichProlog differs from an ILP algorithm in that it clearly separates the deductive component from the inductive component of the query and potentially allows more complicated queries to be built from alternating between the two components. In other words, RichProlog allows the \textit{interconnection} between deduction and induction.

However, RichProlog is less flexible in the sense that although it handles queries with both deductive and inductive parts, it only accepts queries in strict alternating form, \( \exists \bar{x} \forall \bar{y} \varphi \). It does not offer any way to embed one query \textit{into} another, or recursively execute one form of reasoning \textit{as a result} of the other. In other words, RichProlog does not offer a way to \textit{intraconnect} the two processes, which is believed to be a necessary further step to achieve the aim of developing a reasoning engine for deductive/inductive reasoning. Ideally, one would expect an agent to actively transform a given query into a series of deductive/inductive inferences when necessary as part of its reasoning process, rather than fully specifying the query in a format corresponding to the exact reasoning steps the agent is to follow. It is thus foreseeable that from a reasoning agent’s perspective, it would really be beneficial to perform the two forms of reasoning in a truly integrated fashion such that an agent can switch between deductive and inductive inference when it deems necessary.
The deductive-inductive resolution (DIR) strategy presented in this thesis (refer to Chapter 5), on the other hand, approaches the integration of deduction and induction differently by providing inferential relation rules that allow simple queries to be recursively transformed into more complex ones, corresponding to a recursive application of deductive and inductive inferences. In this way, the DIR framework allows not only for interconnecting deduction and induction but also for intraconnecting the two processes, such that induction is embedded into deduction as well as executed alongside deduction, and vice versa.

3.3 Collaborative Problem Solving

3.3.1 Various Forms of Collaboration

Hannebauer (Hannebauer, 2002) has summarized four key reasons which prevent individual agents from solving problems solely by themselves and make collaboration a desirable property for problem solving. The four key reasons are: knowledge, competence, scalability and reliability. According to Nwana and Jennings (Nwana, Lee, & Jennings, 1996), there are many others: (i) Preventing anarchy or chaos; (ii) Dependencies between agents’ actions; (iii) Meeting global constraints; (iv) Distributed expertise, resources or information and (v) Efficiency.

In spite that research on collaboration among problem solvers have taken vastly different approaches, they can nevertheless be categorized into the following three key areas: distributed computing, distributed problem solving and collaborative problem solving. Differentiating these three forms of collaboration is important for understanding what problem solving involving multiple agents is really about.

In distributed computing, the major concern is efficiency. A centralized task is partitioned and given to a number of processors or problem solvers with the aim of decreasing the processing time. In distributed problem solving, however, given a
distributed situation to start with, the concern is how to reach a solution efficiently without gathering information into one single agent. In *collaborative problem solving*, on the other hand, the problem setting is somewhat similar to distributed problem solving but the collaboration among agents are not precisely defined by the designer of the system. Agents choose to collaborate based on their judgement that doing so will make them more likely to achieve their individual goals.

The term ‘collaborative problem solving’ is first used by Hannebauer (Hannebauer, 2002). In addition, he made it clear the distinction between that and ‘distributed problem solving’:

> The entities of such systems (distributed problem solving systems) are typically altruistic, i.e. they willingly accept tasks assigned to them in a client-server manner. The form of organization in distributed problem solving systems is usually restricted since collaboration relations are often predetermined and fixed.

Thus far, the distinction between these three forms of collaboration has become clearer. The distinction comes from the level of autonomy of the entities participating in the problem solving process. In distributed computing, individual entities are not autonomous and serve no purpose alone. They are parts of a central computational entity that is physically distributed. The entities in distributed problem solving systems enjoy a higher level of autonomy but are not self-interested. They do not have their individual goals, not to mention act according to them, hence do not qualify as true agents. Durfee’s remark (Durfee, 1999) makes this clear, “distributed problem solving typically assume a fair degree of coherence is already present: the agents have been designed to work together.” Collaborative problem solving, on the other hand, concerns about collaboration among entities with high degree of autonomy that make their own decisions about how to act, such that there is no predefined script telling them they need to collaborate and how.
3.3.2 Collaboration Models

Models for collaboration among agents with high degree of autonomy have attracted research attentions from various different perspectives and for vastly different applications over a long period of time. Much work have laid the foundations, inspired by which other works aim to build collaborative systems in practice.

Some of these works focus on the cognitive (Fagin, Moses, Halpern, & Vardi, 1997; Halpern & Shore, 1999; Singh, Rao, & Georgeff, 1999) or motivational aspects (Rao & Georgeff, 1991; Hustadt, Dixon, Schmidt, Fisher, Meyer, & van der Hoek, 2001), while others focus on the coordination aspects (Jennings, 1995, 1996; Cox, Durfee, 2005), organizational aspects (Conte & Sichman, 2002), communicational aspects (Cohen, Levesque, 1995; Aknine, Pinson, & Shakun, 2004), plan execution aspects (Giacomo, Lespérance, & Levesque, 2000; Kelly & Pearce, 2006) and programming aspects (Rao, 1996; Hindriks, Boer, Hoek, & Meyer, 1999; van Roy, Brand, Duchier, Haridi, Schulte, & Henz, 2003).

In particular, multi-agent collaboration modeled as solving distributed constraint satisfaction problems (DCSP) have been prominent (Yokoo et al., 1998). When a multi-agent collaboration problem can be represented in terms of satisfying a set of constraints on variables distributed among a group of agents, various algorithms can be applied to solve it, such as (Yokoo, Durfee, Ishida, & Kuwabara, 1992; Yokoo, 1995; Hannebauer, 2000; Jung & Tambe, 2005; Modi, Shen, Tambe, & Yokoo, 2005). In those collaborative approaches, problem solving is based on assigning values to local variables and exchanging values of those variables. Although this has privacy and communication benefits, it imposes significant restriction for problems involving agents with diversified expertise represented in richer formalism.

Modeling collaboration as DCSP has other advantages such as simplicity, extensibility and efficiency but these approaches often assume a high degree of coherence and homogeneity among the agents. That is, these agents somehow know that they all have the same objective of satisfying their respective constraints and
communicating their choices of value to other agents. In other words, DCSP approaches require agents to be designed to collaborate. In addition, the variables need to be assigned to the agents to start with, presumably by some centralized agents.

Undoubtedly, giving a thorough coverage of those numerous collaboration models in the field is beyond the scope of this thesis. In the following section, a description is provided on a recent collaborative approach based logic programming. The multi-agent answer set programming approach is of high relevance to and shares many similarities with the deductive-inductive resolution (DIR) approach presented in this thesis. Both approaches are based on the logic programming paradigm and concern about deliberation, interaction and information exchange with multiple logic-based collaborative agents.

3.3.3 Multi-Agent Answer Set Programming

Recent progresses on extending answer set programming (Vos & Vermeir, 2004) to multi-agent settings have shown promise for collaborative execution of logic programs among interactive logic-based agents with high degree of autonomy. In answer set programming, a problem is described by an extended disjunctive logic program (Gelfond & Lifschitz, 1991; Niemelä, 1999; Marek & Truszczynski, 1999) and solutions are computed by answer sets of the program.

According to (Lifschitz, 2002), an answer set is defined as follows:

**Definition 4.** Let \( \Pi \) be a logic program without negation of failure, and let \( X \) be a consistent set of literals. We say that \( X \) is closed under \( \Pi \) if, for every rule in \( \Pi \), \( \text{Head} \cap X \neq \emptyset \) whenever \( \text{Body} \subseteq X \). We say that \( X \) is an answer set for \( \Pi \) if \( X \) is minimal among the sets closed under \( \Pi \).

For example, the logic program \( \Pi = \{ p; q, \neg r \leftarrow p \} \) has two answer sets \( X = \{ p, \neg r \} \) or \( X = \{ q \} \).
Models based on the multi-agent answer set programming framework have been developed for solving different kinds of collaborative problems. For instance, in (Nieuwenborgh et al., 2007), a model has been proposed for tackling the hierarchical decision problem. In those problems, the decision making procedure involves the participation of a group of agents with diversified (and sometimes inconsistent) knowledge and expertise. In this work, individual agent’s knowledge and expertise are modeled by the logic program it is equipped with.

When given a query to a group of agents to answer, each agent comes up with a solution to the query based on its logic program and the constraints received from other agents, through the communication of the answer sets. They collaborate in a hierarchical way, such that when one agent passes its own answer set(s) to an agent higher up in the (predefined) hierarchy, the latter selects or refines the answer set(s) to meet its own restrictions and passes the refined answer set(s) further up. At the end of the execution, a solution, if one is found, reflects a compromise among individual agents in the system with diversified knowledge and expertise. The hierarchical interaction scheme thus joins together isolated reasoners towards solving logic programming problems in collaboration.

While collaboration in the above work takes the form of progressively refining answer sets in order to satisfy all agents’ views, work in (Sakama & Inoue, 2008) has taken an opposite approach to accommodate diversity. In (Sakama & Inoue, 2008), conflicting beliefs within a single agent and between multiple agents, represented by different answer sets, are compromised to form a new program that maximizes agreement. Two ways to coordinate different agents’ views have been proposed. In the generous form of coordination, the resulting program has an answer set equivalent to the union of the answer sets of all individual programs, thus retaining all original beliefs of each agent. In the rigorous form of coordination, the resulting program has an answer set equivalent to the intersection of the answer sets of all individual programs, thus retaining only the beliefs that are in common among the agents. Either way, the resulting program accommodates the
Multi-agent collaborative frameworks based on answer set programming, such as the ones described above, share many similarities with the DIR framework to be presented in this thesis (as we shall see in Chapter 5). First, both the multi-agent answer set programming and the DIR approaches are based on logic programming, which provides rich formalisms and techniques for modeling diversified agent beliefs and reasoning processes. In addition, both approaches allow agent interaction during collaboration but, in the mean time, both avoid unrestricted sharing of agents’ internal knowledge through communicating only the answer sets or the logical consequences of an agent’s knowledge.

However, although extensions of answer set programming techniques have demonstrated its potential for collaborative execution of logic programs among interactive logic-based agents, existing frameworks do not integrate induction and are thus inadequate for problems necessarily involving learning. The hierarchical decision problem, for example, only captures the deductive aspects in decision making among collaborative agents. Collaboration problems in multi-agent settings, in general, often exhibit a higher level of uncertainty and involve inductive aspects which is typically not supported by extensions of deductive logic programming paradigms, such as answer set programming. In comparison, the CollabLP problem as defined in this thesis (refer to Chapter 4) accommodates inductive aspects (as well as deductive ones) and thus captures a much broader class of logic programming problems in multi-agent collaborative settings.

3.4 Induction in Distributed Settings

3.4.1 Collaborative Induction through Interaction

Work on multi-agent learning often employ multiple instances of induction separately—as opposed to learning that tightly integrates processes of induction among
agents. Although such learning strategy, which involves multiple separate instances of induction, has been successfully applied to various learning problems (Stone & Veloso, 2000), this type of learning often fails in complex domains. In these approaches, agents do not necessarily require the direct participation of other agents during learning, resulting in systems level behavior that does not converge—due to uncoordinated learning happening in isolation.

Researchers have recently started to distinguish this type of learning, or single agent learning in multi-agent environments, with true multi-agent learning. According to Kazakov and Kudenko (Kazakov & Kudenko, 2001), the problem of true multi-agent learning has far more complexity than simply having each agent perform localized learning in isolation.

In multi-agent learning problems, interaction and collaboration during learning are often prerequisites to successful learning outcome as knowledge necessary in learning a hypothesis is typically distributed over a number of agents, as we have see in the sorting example in Section 2.4.3.

This restriction gives rise to two major complications. First, no individual agent can accomplish the learning task alone any more (without interaction). Second, careful exchange of information between agents is required while generating hypotheses, given that sharing complete knowledge is often infeasible in those environments for many reasons (such as resource and privacy concerns). In other words, due to these constraints, neither of the two extremes of the collaboration scheme would work, i.e. learning in isolation or communicating everything.

Therefore, one central challenge in the design of multi-agent inductive learning algorithms is to design those agents (with isolated, inaccurate and sometimes inconsistent views) such that they collaborate in a disciplined manner to learn complex hypotheses in a distributed fashion. Weiβ and Dillenbourg clearly identify this problem “interaction does not just serve the purpose of data exchange, but typically is in the spirit of a cooperative, negotiated search for a solution of the learning task (Weiβ & Dillenbourg, 1999)”. 

43
Research that extends inductive processes to distributed settings is relatively young.

### 3.4.2 Collaborative ILP and ALP

It can be observed from earlier discussions that close interaction and collaboration among agents during induction is often a prerequisite for successful learning outcomes. Research have recently started to investigate in distributing single-agent inductive processes over multiple agents. Notably, there are early attempts on distributing logic-based inductive approaches from both inductive logic programming (ILP) and abductive logic programming (ALP) domains. Not only algorithms have been specifically and completely redeveloped for induction in multi-agent settings, the problems of collaborative induction have also been reformulated.

According to (Huang & Pearce, 2006b), ILP problems, once distributed in multi-agent settings, should be generalized to allow hypotheses formation based on the collective theory using a collaborative approach. More formally, the problem of ILP in multi-agent systems has been defined as follows:

**Definition 5** (Collaborative ILP). The collaborative ILP problem is defined by a finite set of agents $\mathcal{A}$; the theory $T(a_i)$ of each agent $a_i \in \mathcal{A}$; and the set of positive and negative examples $E^+(a_i)$ and $E^-(a_i)$ of each agent $a_i \in \mathcal{A}$. Further, $T = \bigcup_{a_i \in \mathcal{A}} T(a_i)$ is the set of all theories and $\mathcal{E}^+ = \bigcup_{a_i \in \mathcal{A}} E^+(a_i)$ and $\mathcal{E}^- = \bigcup_{a_i \in \mathcal{A}} E^-(a_i)$ are the set of all positive/negative examples. Then collaborative ILP can be viewed as the process of collaboratively generating the hypothesis $H$ such that the following conditions hold:

1. **Necessity:** $T \not\models \mathcal{E}^+$

2. **Sufficiency:** $T \land H \models \mathcal{E}^+$

3. **Weak Consistency:** $T \land H \not\models \Box$
4. **Strong Consistency:** $T \land H \land E^- \not\models □$

5. **Collaborative Necessity:** $\neg \exists a_i \in A \ T(a_i) \land H \models E^+$

The first four conditions are adapted from ILP in a single agent setting (Muggleton, 1999) (refer to Definition 3) and are generalized to allow hypothesis generation over the agents’ *total* theory. The fifth condition asserts that there exists no individual agent who is able to induce the hypothesis based solely on its own theory, thus necessarily requires collaboration.

In a similar vein, recent efforts have been made to distribute abductive logic programming (ALP) (Kakas, Kowalski, & Toni, 1992; Denecker & Kakas, 2002) algorithms over multiple interactive agents.

In (Ma et al., 2008), for instance, a distributed abductive proof algorithm (DARE) is developed which enables a group of agents to cooperate to produce a proof based on a distributed set of knowledge and constraints. Moreover, DARE allows the construction of open systems where agents can come and leave even during proofs.

In DARE, each agent possesses a set of integrity constraints and is capable of performing local abductive reasoning based on its own theory. Just like the work in collaborative ILP as mentioned previously, the aim of the DARE algorithm is to compute a global hypothesis, by integrating local hypotheses formed by individual agents, such that the global hypothesis satisfies the constraints of all agents which have participated in the proof.

The DARE approach assumes a shared ontology as well as the mechanism for advertising the predicates each agent is capable of abducing. While forming the global hypothesis, if a predicate is locally abducible, an agent tries to abduce it by itself. Otherwise, it posts a query to other agent(s) who has publicly advertised the predicate. The advertising and requesting mechanism provides a collaborative interface among the agents as well as ensuring internal states are properly hidden from the outside.
CHAPTER 3. LITERATURE REVIEW

From the perspective of distributing inductive/abductive process, the CollabLP problem defined in this thesis (as we shall see in Chapter 4) encompasses both problems of collaborative inductive and abductive logic programming mentioned earlier. The CollabLP problem captures not only the cases where induction is distributed in multi-agent environments, but also deductive problems requiring collaboration in general.

In a similar vein to the advertising strategy employed by DARE, the deductive-inductive resolution (DIR) approach described in this thesis (refer to Chapter 5) achieves information hiding through the use of deductive and inductive consequences as the only means for communication. Moreover, the DIR approach offers a solution not only to the problems of inductive (and abductive) logic programming in distributed settings, but also to the more general problem of CollabLP. In the DIR approach, the process of distributed hypothesis formation is incorporated into and becomes part of a more general logic programming task.
Part II

Theory
Chapter 4

The Collaborative Logic Programming Problem

4.1 Overview

Given the collaborative and dynamic nature of multi-agent programming problems as previously discussed, we generalize a class of problems—named collaborative logic programming (CollabLP) problems. In this chapter, we first define the so called CollabLP problem and then describe using examples how some practical problems may be formulated as CollabLP problems.

Loosely speaking, collaborative logic programming (CollabLP) involves solving logic programming problems by a group of agents acting collaboratively as a single reasoning system, without sharing complete knowledge. There are a number of aspects to the CollabLP problems which can be informally described as follows:

(i) The global theory is distributed among a number of collaborative agents, such that each agent has part of the theory but not enough for any of them to solve the problem individually.
(ii) Agents are unable to reveal their internal theories directly. For example, this may result from privacy policies or communication restrictions due to bandwidth, power consumption, reliability or propagation considerations.

(iii) Agents interact with each other by issuing (and answering) queries, and not through any other means.

(iv) The agents’ combined theory may be insufficient for solving the problem without some hypotheses being generated (necessarily requiring induction).

The fourth point captures the dynamics and uncertainties that often exist in multi-agent environments, where key knowledge may be missing such that agents must accommodate this by allowing for some informed guesses in order to solve the problem.

4.2 Preliminaries and Notation

Constants, variables, function symbols and predicate symbols are defined in the usual logic programming sense. A term is either a constant, a variable or an \( n \)-place function of \( n \) terms. An atom is an \( n \)-place predicate of \( n \) terms. A literal is either a positive or negative atom (as in \( l \) or \( \neg l \)). A clause is a set of literals. A Horn clause is a clause with at most one positive literal. A definite clause is a Horn clause with exactly one positive literal. A goal clause is a Horn clause with no positive literal. For our purposes, theories and hypotheses are defined as sets of definite clauses. A query is defined as a goal clause with a single literal. In addition, the following symbols are also defined.

- \( A = \{a_1, \ldots, a_n\} \) represents a finite set of agents;
- \( T(a_i) \) represents the finite theory of an individual agent \( a_i \in A \);
- \( T = \bigcup_{a_i \in A} T(a_i) \) is the total theory of all agents in \( A \), or the global theory;
• \( Q \) is the set of all possible queries.

### 4.3 The Basic CollabLP Problem

The CollabLP problem can now be formally defined. The definition is given in two steps. Before presenting the general definition of the CollabLP problem, we first introduce a simplified version in which the global theory is assumed to be complete (cf. the fourth point above). Under this assumption, the problem is deductively solvable through collaboration, and no inductive hypothesis is required. This definition provides a basis which is later generalized to cover problems for which the global theory is not complete and some inductive hypotheses are necessarily required.

**Definition 6 (Basic Collaborative Logic Programming (CollabLP)).** Given a query \( \gamma \in Q \) and a finite group of agents, \( A = \{a_1, \ldots, a_n\} \), each with private theory \( T(a_i) \), that are allowed to communicate messages, \( M \), the basic CollabLP problem decides if \( T \models \gamma \), where \( T = \bigcup_{a_i \in A} T(a_i) \), with the conditions that

1. \( T \not\models \square \);  
2. \( \neg \exists a_i \ T(a_i) \models \gamma \); and  
3. \( M = Q \cup \{\text{true}, \text{false}\} \).

The first condition requires agents’ theories to be consistent, both individually and with each other. Although this is a highly ideal assumption, it is an open possibility that further extensions eliminate this condition and utilize techniques from non-monotonic reasoning to deal with inconsistent theories.

The second restriction demands that the problem necessarily requires collaboration. In other words, no individual agent can answer query \( \gamma \) by itself as no individual theory \( T(a_i) \) is complete.
The third condition restricts the communication of messages, $\mathcal{M}$, between agents to only logic queries, $\mathcal{Q}$. Since ground queries may either succeed or fail, true and false may be optionally used in addition to queries as messages. Hence communication in CollabLP is highly restricted in the sense that agents are not permitted to simply exchange everything they know, nor to reveal their internal knowledge directly.

4.4 Sorting Example: Basic CollabLP Case

Here, we use a sorting example to illustrate the CollabLP problem. Imagine an environment where the overall program is distributed over a number of agents who are willing to collaborate in answering queries. Consider three such agents, $a_1$, $a_2$ and $a_3$. Agent $a_1$ knows the theory for finding the minimum element of a list defined by $T(a_1)$ as follows:

\[
\text{min}(\text{List}, \text{Min}) : - \text{sort}(\text{List}, \text{L}), \text{first}(\text{L}, \text{Min}).
\]

\[
\text{first}([N|\text{Ns}], N).
\]

Agent $a_2$ knows about list permutation and list ordering, in the form of the following theory $T(a_2)$:

\[
\text{permutation}([], []).
\]

\[
\text{permutation}(\text{List}, [\text{First}|\text{Perm}) : - \text{select}(\text{First}, \text{List}, \text{Rest}), \text{permutation}(\text{Rest}, \text{Perm}).
\]

\[
\text{ordered}([]).
\]

\[
\text{ordered}([\_]).
\]

\[
\text{ordered}([X,Y|\text{Tail}]) : - X \leq Y, \text{ordered}([Y|\text{Tail}]).
\]

\[
\text{select}(\text{Elem}, [\text{Elem}|\text{Tail}], \text{Tail}).
\]

\[
\text{select}(\text{Elem}, [\text{Head}|\text{Tail1}], [\text{Head}|\text{Tail2}]) : -
\]
4.4. SORTING EXAMPLE: BASIC COLLALBP CASE

select(Elem,Tail1,Tail2).

Agent $a_3$ knows a list can be sorted by finding an ordered permutation of the list, in the form of $T(a_3)$ as follows:

$$\text{sort}(L,L') := \text{permutation}(L,L'), \text{ordered}(L').$$

Giving the query $\gamma = \min([3, 2, 4, 1, 5], X)$ to any of the agents $a_1$, $a_2$ and $a_3$, with the above theories, CollabLP concerns proving if the query is entailed by the group’s total theory through a collaborative effort. Recall that under the CollabLP problem setting, none of the agents is aware of the theories of other agents, nor is any agent allowed to reveal its internal theory to the others. However, they may query each other and help each other answer some sub-queries.

In this particular example, the global theory is complete, i.e. $T(a_1) \cup T(a_2) \cup T(a_3) \models \gamma$ (with $X$ bound to 1). Hence this is an instance of the basic CollabLP problem. Answering query $\gamma$ in this case requires nothing more than a careful exchange of messages among the agents.

In this case, a solution may simply require agents to pass whichever goals they cannot answer as sub-queries (message) to other agents. For example, when agent $a_1$ is held up by the goal $\gamma' = \text{sort}([3, 2, 4, 1, 5], L')$, it sends $\gamma'$ as a sub-query to agents $a_2$ and $a_3$. Since the global theory is assumed to be complete, there is a guarantee that some agent would be able to answer it. This process goes on until eventually all sub-queries are answered and $a_1$ will come to a positive conclusion that $T(a_1) \cup T(a_2) \cup T(a_3) \models \gamma$.

It is worth pointing out that the exact message passed or the way messages are exchanged is irrelevant to the CollabLP problem. The CollabLP problem only concerns what needs to be answered, i.e. it only defines a problem to be solved. As to how it is solved, that is an implementation issue. Different solution methods may exist for a particular CollabLP problem.
4.5 The Generalized CollabLP Problem

The basic definition is now extended for the generalized problem setting, in which the global theory is no longer assumed to be complete. In order to give the definition in the general problem setting, two additional symbols are introduced:

- \( \mathcal{H} \) represents the global hypothesis required for solving the problem, which may be empty; and
- \( H(a_i) \) represents the local hypothesis induced by an individual agent \( a_i \in \mathcal{A} \), which may also be empty.

At this point we emphasize the omniscient point-of-view taken when discussing the existence of a global hypothesis, which is required to supplement the incomplete theory. The global hypothesis \( \mathcal{H} \) is made up of (but not necessarily a union of) hypotheses formed by individual agents, \( H(a_i) \), in the course of solving the problem, that is, \( \mathcal{H} \subseteq \bigcup_{a_i \in \mathcal{A}} H(a_i) \). In other words, not all individual hypotheses are necessarily required directly in the global hypothesis.

The hypotheses each individual agent can possibly form are dependent on the inductive process employed (besides the theory it is equipped with, \( T(a_i) \), and the query given to it, \( \gamma \)). However, different agents may employ entirely different inductive routines, although the inductive routine employed by each agent has to conform with Definition 3. With this mentioned, CollabLP for the general problem setting can be defined as follows:

**Definition 7** (Generalized Collaborative Logic Programming (CollabLP)). Given a query \( \gamma \in Q \) and a finite group of agents, \( \mathcal{A} = \{a_1, \cdots, a_n\} \), each with private theory \( T(a_i) \), that are allowed to communicate messages, \( \mathcal{M} \), and are capable of formulating hypothesis, \( H(a_i) \), the generalized CollabLP problem decides if \( T \land \mathcal{H} \models \gamma \), for some \( \mathcal{H} \), where \( T = \bigcup_{a_i \in \mathcal{A}} T(a_i) \), with the conditions that

1. \( T \land \mathcal{H} \not\models \square; \)
4.6. Sorting Example: Generalized CollabLP Case

2. \( \neg \exists a_i T(a_i) \land H(a_i) \models \gamma; \text{ and} \)

3. \( M = Q \cup \{\text{true, false}\}. \)

The generalized definition is adequate to capture a wider range of collaborative logic programming problems, whether or not requiring induction. Importantly, the definition captures CollabLP problems with three main types of variations:

- Problems that are purely deductive, with global theory being complete (i.e. \( \mathcal{H} = \emptyset \)) and no induction is required, even though no individual agent can alone find the solution;

- Problems that are deductive per se (global knowledge is complete), but induction serves the purpose of, or as an alternative to, communication (i.e. \( \mathcal{H} = \emptyset, H(a_i) \neq \emptyset \) for some \( a_i \));

- Problems that necessarily require induction, where global knowledge is incomplete (i.e. \( \mathcal{H} \neq \emptyset \)) and the problems are only solvable based on some hypotheses.

4.6 Sorting Example: Generalized CollabLP Case

Again, we build on top of the earlier sorting example with a slight modification to illustrate the CollabLP problem in the generalized setting. Still, we have the same collaborative agents \( a_1 \) and \( a_2 \) with the same theory \( T(a_1) \) and \( T(a_2) \) as before. The same query \( \gamma = min([3, 2, 4, 1, 5], X) \) is given. The only change is agent \( a_3 \) is now missing.

As can be seen, agent \( a_1 \) and \( a_2 \) have almost all the ingredients available to prove \( \gamma \), so long as they can figure out the missing hypothesis that “sorting can be performed by generating permutations and checking if the permutation is ordered”. Ideally, agent \( a_1 \) and \( a_2 \) should accommodate this and prove \( \gamma \) by coming
up with the following hypothesis $\mathcal{H}$ through induction and collaboration:

$$\mathcal{H} = \{ \text{sort}(L, L') :\neg \text{permutation}(L, L'), \text{ordered}(L') \}$$

Obviously, at least one agent has to generate this hypothesis, to support the proof, in the process of answering query $\gamma$. A sample solution approach may proceed like this. Somewhere along the execution, agent $a_1$ would require agent $a_2$ to prove a sub-query $\gamma' = \text{sort}([3, 2, 4, 1, 5], L')$ as agent $a_1$ does not have the definition of $\text{sort}$ to prove it by itself. Although agent $a_2$ does not have the definition either, it can be induced based on its theory of $\text{permutation}$ and $\text{ordered}$.

One way to establish this connection is through the transfer of positive/negative examples of the missing concept. In this case, agent $a_1$ asks agent $a_2$ to prove $\gamma'$ by sending it the positive/negative examples of sorted lists (assume for now it has them), which is effectively stating “Here is the query and here is what I mean by $\text{sort}$ based on these examples. See if you can prove it”. Agent $a_2$ consequently induces the definition of $\text{sort}$ (hypothesis $\mathcal{H}$), which proves $\gamma'$, and responds to $a_1$ with $\text{sort}([3, 2, 4, 1, 5], [1, 2, 3, 4, 5])$.

This collaborative programming idea promises a multi-agent programming paradigm, in which distributed programs (even with some missing fragments) can be executed without being congregated. The user can issue a query to any agent in the team and it is entirely up to the agents to execute the program and work out the missing fragments to answer the query. It also leads to a more flexible programming environment, in which the programmers do not need to fully specify every detail of the program, so long as they are ready to provide necessary positive/negative examples of the missing concepts when needed.

A prototype has been implemented and tested by integrating Prolog with the ILP system Aleph (Srinivasan, 2001). In the implementation, when asked to prove a goal, agents deduce based on their local theory and, if necessary, induce missing knowledge in order to proceed with the theorem proving and may potentially invoke other agents in the team through the transfer of positive/negative examples.
4.7 Distributed Path Planning Example

In this section, we illustrate how practical applications may be viewed as CollabLP problems using a distributed path planning example (Huang & Pearce, 2007). Imagine a futuristic situation in which cars on the streets are able to wirelessly communicate with cars nearby. This could be utilized to allow cars to find out paths among themselves, based on their historical travel information, which no longer relies on a centralized system to keep track of all the information.

Under these circumstances, individual cars are unlikely to have complete knowledge about how to get from one place to all other places. However, if they work together in a collaborative fashion, it is highly likely that a few cars will be able to find the path between any two locations. One obvious way to collaborate is to collect other cars’ knowledge as they meet each other and then compute the path based on this (stored) information. However, for many good reasons (such as communication overhead, storage, privacy and dynamic nature of the information) it is not desirable to do it that way.

A more realistic solution would therefore require a collaborative endeavor through careful negotiation and exchange of information among the cars. In Figure 4.1, for example, assume car A wants to go from a to l, i.e. the query is ‘what is the path from a to l?’ It can be observed from the graphs that one such path is...
a-c-d-g-j-l, however, none of the three cars is able to find the path alone. Assume car A asks car C how to get from $a$ to $l$, instead of replying ‘I don’t know’, car C can reason inductively that ‘if there is a path from $a$ to $g$, then there is a path from $a$ to $l$’.

Upon receiving the reply from car C ‘only if there is a path from $a$ to $g$’, car A infers that ‘car C knows a path from $g$ to $l$’ and then commits itself to find a path from $a$ to $g$. After collaborating with car B, car A eventually finds the path from $a$ to $g$ and thus the full path.

Again, in this example, cars want to collaborate but they do not necessarily wish to expose their entire travel history to the others. Instead, they choose to do their own part of the reasoning and communicate, what seems to be promising, partial results based on their own knowledge and reasoning.

Many aspects of the problem have been abstracted away for illustration purpose. The basic idea may be extended to include potentially complicated and dynamic road information such as distance, capacity, congestion level, throughput, duration for travel, speed limit etc. The approach can thus be adjusted for finding not only any path, but also the shortest, quickest or least congested path etc. But the fundamental restrictions as previously mentioned remain unchanged, if not intensified. More detailed discussion and experimental results in relation to this example is provided in Chapter 7.

4.8 Summary

In essence, the CollabLP problem captures the fact that knowledge is distributed, private and (possibly) incomplete. Communication is allowed among the agents but restricted syntactically to be in the form of simple logic programming queries. In this sense, the CollabLP formulation captures a wide range of logic programming problems in multi-agent systems, whether or not involving induction.

To help understand the CollabLP problem and its constraints, imagine a num-
ber of pirates searching for some buried treasure. Each of them has part of the clue, encoded as logic programs, which will give the location of the treasure once executed. Being inspired by the common goal, i.e. to execute the program and to find the treasure, the pirates desperately want to collaborate but none of them is willing to reveal his entire part to the rest. Worse still, some fragments of the program have gone missing so that they have to be induced based on the rest of the program and/or some guesses. However, we do make the assumption that these pirates are truthful in the messages they communicate and the communication mechanism is reliable.

Through the previous two examples (collaborative sorting and path planning), it can be observed that while tackling CollabLP problems of this kind, the following elements must be considered for deriving effective solutions:

(i) Centralization must be replaced by an appropriate interaction mechanism. This means that program execution and learning must take place based on distributed knowledge, without the need to centralize;

(ii) Conservative communication policies must take the form of exchanging queries, examples and hypotheses in restricted forms, to limit what information may be transferred. This prevents agents revealing unnecessary internal information and also has the potential to lower communication costs;

(iii) Induction plays an important role—not only for coming up with hypothesis but also for abstracting away agents’ internal inference models—toward a more collaborative, synergistic form of inference. Therefore solution strategies must be aware of and incorporate both forms of reasoning—deductive as well as inductive.
Chapter 5

The Deductive-Inductive Resolution Framework

5.1 Overview

The distributed path planning example shares many similarities with the multi-agent sorting example and are both captured by the so called CollabLP problem defined in the earlier chapter. In this chapter, the details of the deductive-inductive resolution (DIR) framework is provided, under which deduction, induction and collaboration are integrated to tackle collaborative problems of this kind.

While logic programming and inductive logic programming techniques focus respectively on traversing the resolution tree upwards and downwards, the DIR approach generalizes them by allowing a bi-directional traversal along the resolution tree. Although the new approach is crowned ‘resolution’, the term here is conceptual rather than specifically referring to a particular algorithm. In fact, the DIR approach abstracts away the details of the actual deductive and inductive algorithms employed and focuses on the integration of the two.

Although the DIR solution shares some of its notation with the work by Martin
et al. on a generalized logical framework (Martin et al., 2001), it approaches the integration of deductive/inductive inferences in a different way. The DIR framework views the integration from an agent’s perspective and allows a simple query to be recursively transformed into a potentially more complex form, corresponding to a recursive application of deductive and inductive inferences. The DIR framework also incorporates agent interaction and collaboration while performing deductive-inductive reasoning.

To start with, here is a revised network reachability example to illustrate how deductive-inductive reasoning, or bi-directional resolution, proceeds. We want to decide if $T \models \gamma$, given background theory $T = T_1 \cup T_2 \cup T_3 \cup T_4$ and query $\gamma$ as follows:

- $T_1 = \text{reachable}(a, c)$
- $T_2 = \text{reachable}(a, b)$
- $T_3 = \text{reachable}(c, d)$
- $T_4 = \text{reachable}(A, C) \leftarrow \text{reachable}(A, B) \land \text{reachable}(B, C)$
- $\gamma = \text{reachable}(b, d)$

This is, of course, not the case under a purely deductive framework. Under a deductive-inductive framework, however, it is permissible to obtain an intensional expansion of the theory through induction first. In this case, the theory can be expanded to incorporate an inductive hypothesis, $H = \text{reachable}(b, c)$, since $\text{reachable}(a, c)$ ($T_1$) can be explained by $\text{reachable}(a, b)$ ($T_2$) and $\text{reachable}(b, c)$ ($H$) via application of $T_4$. With $\text{reachable}(b, c)$ ($H$) and $\text{reachable}(c, d)$ ($T_3$), we can subsequently conclude $\text{reachable}(b, d)$ ($\gamma$). This reasoning process is illustrated in Figure 5.1. The induction process on the left half utilizes the V-operators of the inverse resolution technique as described in Section 2.2.3.

In this example the aim is to prove query, $\gamma$, which is a deductive task. During the proof, however, the inductive process is executed to find inductive hypothesis, $H$, which allows the proof to proceed. This process requires an upward followed
5.1. OVERVIEW

Figure 5.1: Bi-directional resolution for a simple reachability example in a network topology. \( T_n \) is the background theory, \( H \) is the inductive hypothesis and \( \gamma \) is the query. **LEFT:** The inductive (inverse) resolution step is shown, where \( \text{reachable}(a, c) \) is entailed by \( \text{reachable}(a, b) \) and \( \text{reachable}(b, c) \), with \( \text{reachable}(b, c) \) being the inductive hypothesis (as shown in Figure 2.2). **RIGHT:** The deductive (forward) resolution step is shown, from \( \text{reachable}(b, c) \) and \( \text{reachable}(c, d) \), we deductively conclude \( \text{reachable}(b, d) \).

by a downward traversal—thus bi-directional traversal—along the resolution tree. Note that two possible inductive hypotheses can be generated using the inverse resolution technique—the other being \( \text{reachable}(c, b) \), since \( \text{reachable}(a, c) \) and \( \text{reachable}(c, b) \) explains \( \text{reachable}(a, b) \). However, only \( \text{reachable}(b, c) \) allows the subsequent deduction to succeed.

In the rest of this chapter, some preliminary definitions will be given first followed by the definition of the elementary inferential relations, identifying the elementary inference scenarios which can be used as building blocks for more complicated deductive-inductive inferencing scenarios. The DIR formalism is then extended from single-agent to multi-agent setting where interaction and collaboration among agents are incorporated, through the sharing of deductive consequences and inductive hypotheses. At the end of this chapter, the relation between the DIR approach and the CollabLP problem is briefly discussed, leaving the de-
5.2 Interleaving Deduction and Induction

Under a deductive-inductive framework, when confronted with an inferencing task, an agent may require a series of deductive and inductive inferences in order to answer it. This potentially results in a simple inferencing task being expanded into a more complicated one, involving a series of nested deductive-inductive inferences. We first examine the possible forms of nested inference, in terms of relations, before defining their semantics in the following section.

Given a theory, $T$, the following elementary inferential relations are identified under which deductive and inductive inferences may be nested.

$D_{dd}$ Execution of a deductive routine requires another deductive routine to be executed first. This is the most common way inferencing is expanded, without induction even playing a part. A Prolog program is a collection of such rules.

$D_{id}$ Execution of a deductive routine requires an inductive routine to be executed first. This is typically caused by some missing knowledge which prevents the execution of deduction from succeeding without an inductive hypothesis.

$I_{di}$ Execution of an inductive routine requires a deductive routine to be executed first. This is typically required because the execution of an inductive routine depends on some missing knowledge which must first be deduced.

$I_{ii}$ Execution of an inductive routine requires another inductive routine to be executed. This is typically required because the execution of an inductive routine depends on some incomplete theory which must first be induced.
5.2. INTERLEAVING DEDUCTION AND INDUCTION

\( \mathcal{I}_{id} \) Completion of an inductive routine requires a deductive routine to be executed. This is typically required when induction is based on an unsound mechanism such that the resulting hypothesis must be verified. This is an abstraction of what some ILP systems do.

Note that relation \( \mathcal{D}_{di} \) does not exist for deductive inference. This is because the inductive part (following the deductive part) does not serve any purpose. As a result, this relation effectively means carrying out deduction directly and thus does not involve switching between deduction and induction. Likewise, \( \mathcal{D}_{ii} \) and \( \mathcal{I}_{dd} \) do not exist as they are not meaningful either. Consequently, there are only five (of eight possible) different fundamental relations for interleaving the two forms of inference.

These elementary inference relations (involving the two forms of inferences) form the building blocks for more complicated inferencing scenarios. Thus, the above relations provide possible (recursive) transformations from basic deductive or inductive inferences into nested deductive-inductive inferences.

The first two relations summarize how deductive inference may be expanded (corresponding to a downward traversal on the resolution tree) and the last three summarize how inductive inference may be expanded (corresponding to an upward traversal on the resolution tree). A recursive application of these relations consequently leads to bi-directional resolution.

For example, take the relations for inductive inference, \( \mathcal{I}_{di} \), \( \mathcal{I}_{ii} \), and \( \mathcal{I}_{id} \). These relations collectively state that when an agent is confronted with an inductive task, it must do one of the following:

(i) Carry out the induction directly (if the inductive mechanism is sound, uses all available examples and does not rely on any missing knowledge). This does not require an application of the inferential relations.

(ii) Deduce some missing concept first before carrying out the induction, according to relation \( \mathcal{I}_{di} \).
(iii) Induce some missing concept first before carrying out the induction, according to relation $I_{ii}$.

(iv) Carry out the induction but verify the hypothesis later (possibly because the induction mechanism is unsound or does not use all available examples), according to relation $I_{id}$.

Given an inference task, the inferential relations may be recursively applied resulting in a trace of interleaved deduction and induction. Therefore, a theorem prover based on this deductive-inductive framework searches for a legal execution by applying applicable instances of these relations, corresponding to a series of interleaved deductive and inductive inferences, in order to make the query succeed.

5.3 Semantics of Deductive-Inductive Inferences

In this section, the necessary notations and definitions are given, based on which the deductive-inductive resolution framework is developed. All of background theory, $T$, inductive hypothesis, $H$, and deductive consequence (or example), $E$ are sets of definite Horn clauses. We start by formally define deductive and inductive inferences as follows:

**Definition 8** (Deductive Inference). Given a theory, $T$, deductive inference derives consequence, $E$, or proves for a given $E$, that $T \models E$. $\Sigma(T)$ denotes the new theory after applying deductive inference on $T$, where $\Sigma(T) = T \cup E$ iff $T \models E$.

$\Sigma$ can be viewed as an operator which expands theory $T$ through deductive inference, and it can be recursively applied resulting from a series of deductive inferences, i.e., $\Sigma(\Sigma(\cdots(T)))$. This can be viewed as the application of inferential relation $D_{dd}$ identified in Section 5.2.
5.3. SEMANTICS OF DEDUCTIVE-INDUCTIVE INFERENCES

In this work, the deductive consequences and inductive hypotheses derived are incorporated into the theory as a result of the inferences performed, which can be used for later inferences. For example, \( \Sigma(\Sigma(T)) = \Sigma(T) \cup E' \) if \( \Sigma(T) \vdash E' \) where \( \Sigma(T) = T \cup E \) if \( T \vdash E \).

We now turn to the case of induction. For the definition of inductive inference, we take the inductive logic programming perspective on induction, which was defined in Definition 3. Note that this definition is sufficiently general to allow for a broad range of specific refinements.

**Definition 9** (Inductive Inference). Given a theory, \( T = T' \cup E \), where \( T' \nvdash E \), inductive inference derives hypothesis, \( H \), such that \( H \land T' \vdash E \) and \( H \land T' \nvdash \Box \). \( \Pi(T) \) denotes the new theory after applying inductive inference on \( T \), where \( \Pi(T) = T \cup H \) if \( H \neq \emptyset \). \( T \models_1 H \) denotes that \( H \) can be inductively inferred from \( T \).

Note that \( H \) being inductively inferred from \( T \), or \( T \models_1 H \), does not mean \( H \models T \) in general. The only exception to this is in situations where \( T = T' \cup E \) and \( T' = \emptyset \). The notation \( T \models_1 H \) in fact has the underlining meaning of \( T' \land H \models E \). The former notation abstracts \( T' \) and \( E \) and is introduced for simplicity.

Like deductive theorem proving procedures, which provide methods for performing deductive inference, inductive logic programming provides algorithms that can be used to come up with hypothesis, \( H \), systematically and efficiently for inductive inference.

\( \Pi \) may also be recursively applied, in a similar fashion to \( \Sigma \), resulting in a series of hypotheses to be formed, which is an instantiation of the fourth inferential relation, \( I_{ii} \), identified in Section 5.2.

Further, the \( \Sigma \) and \( \Pi \) operators may be nested to yield a sequence of deductive-inductive inferences, such as \( \Sigma(\Sigma(\Pi(\Sigma(T)))) \). To simplify the notation, we define the following shorthand representations.
CHAPTER 5. THE DIR FRAMEWORK

Definition 10 (Deductive-inductive Inference). Given a theory, $T$, $\Sigma(T)$ and $\Pi(T)$ represent the status of the theory after applying deductive and inductive inferences on $T$ respectively, then notations $\Sigma_n(T)$ and $\Pi_n(T)$ represent alternating deductive and inductive inference on $T$, defined recursively as follows:

$$
\Sigma_n(T) = \begin{cases} 
T & \text{if } n = 0 \\
\Sigma(\Sigma_n(T)) \text{ or } \Sigma(\Pi_{n-1}(T)) & \text{if } n > 0 
\end{cases}
$$

$$
\Pi_n(T) = \begin{cases} 
T & \text{if } n = 0 \\
\Pi(\Pi_n(T)) \text{ or } \Pi(\Sigma_{n-1}(T)) & \text{if } n > 0 
\end{cases}
$$

According to the above shorthand notations, the first few terms of $\Sigma_n(T)$ and $\Pi_n(T)$ are as follows:

$$
\Sigma_0(T) = T
$$

$$
\Pi_0(T) = T
$$

$$
\Sigma_1(T) = \Sigma(\Sigma_1(T)) \text{ or } \Sigma(\Pi_0(T)) \quad \text{e.g. } \Sigma(T)
$$

$$
\Pi_1(T) = \Pi(\Pi_1(T)) \text{ or } \Pi(\Sigma_0(T)) \quad \text{e.g. } \Pi(T)
$$

$$
\Sigma_2(T) = \Sigma(\Sigma_2(T)) \text{ or } \Sigma(\Pi_1(T)) \quad \text{e.g. } \Sigma(\Pi(T))
$$

$$
\Pi_2(T) = \Pi(\Pi_2(T)) \text{ or } \Pi(\Sigma_1(T)) \quad \text{e.g. } \Pi(\Sigma(T))
$$

$$
\Sigma_3(T) = \Sigma(\Sigma_3(T)) \text{ or } \Sigma(\Pi_2(T)) \quad \text{e.g. } \Sigma(\Pi(\Sigma(T)))
$$

$$
\Pi_3(T) = \Pi(\Pi_3(T)) \text{ or } \Pi(\Sigma_2(T)) \quad \text{e.g. } \Pi(\Sigma(\Pi(T)))
$$


Both $\Sigma_0(T)$ and $\Pi_0(T)$ stand for the original theory $T$. $\Sigma_1(T)$ (or $\Pi_1(T)$) stands for the status of the theory after applying any number of deductive (or inductive) steps on $T$. $\Sigma_2(T)$ (or $\Pi_2(T)$) stands for applying induction followed by deduction (or deduction followed by induction) on $T$, so on so forth.

Consequently, we can now capture the remaining elemental relations $\mathcal{D}_{id}, \mathcal{I}_{di}$, and $\mathcal{I}_{id}$ identified in Section 5.2.

Each level in the hierarchy captures an infinite number of distinct inferential scenarios. The high level into the hierarchy an inferencing scenario falls into, the higher number of switches between deduction and induction has occurred
and typically the more complex the inferencing task is. Among them, $\Sigma_2$ is a class of logical consequences that deserves special attention as some fundamental deductive-inductive inferencing scenarios fall into this class.

### 5.4 $\Sigma_2$ Inference and RichProlog

$\Sigma_2$-type inference in the deductive-inductive hierarchy simply stands for performing inductive inference on the original theory $T$ and then performing deductive inference on the resulting theory. One way of viewing $\Sigma_2$-type inference is it allows queries with universal quantifiers to be answered, which are often of the following form: ‘Is there a pattern $x$ that matches all individuals $y$?’

This type of query is important, primarily, because any inductive logic programming task can be formulated as a $\Sigma_2$ query: ‘Is there a hypothesis $x$ which logically entails all examples $y$?’ For example, given a theory, $T$, and examples, $E$,

$$\exists H \text{ hypothesis}(H) \land \text{entails}(T, H, E)$$

RichProlog handles $\Sigma_2$-type queries based on a two-stage backtracking algorithm. The first stage involves inducing a hypothesis, $H$, and the second stage verifies that the hypothesis is correct. Backtracking happens during the first stage, the second stage, as well as in-between the first and second stage, in such a way that when verification fails the inductive stage is invoked again to find another hypothesis.

The $\Sigma_2$-type inference is also important for another reason. It represents a deductive reasoning system that executes on an incomplete theory, in such a way that in order to find the deductive consequence of a theory it has to first extend the theory by performing induction—such as in the earlier example in Figure 5.1. This type of inferences can be viewed as a Prolog system that is capable of seeking missing knowledge when it gets stuck in deduction.

Despite the fact that $\Sigma_2$-type inference can manifest in these two different
forms, the difference between the two is obvious. In the first case, the main purpose is to perform induction. The deduction following it is just to verify the inductive hypothesis. In the second case, however, the major concern is to perform deduction. The induction preceding it enables the deduction to be performed. Although, as in the first case, $\Sigma_2$ inferences may be specified as first-order logic queries involving universal quantifiers, in general they can not and thus can not be handled through the process RichProlog is based on.

**RichProlog Vs. Prolog** One could argue that the way RichProlog backtracks between the inductive process and the verification process can be handled by Prolog’s backtracking algorithm too. Indeed, what RichProlog provides over Prolog is just a clear-cut way of splitting the inductive process from the deductive process. In fact, RichProlog can be viewed as a system sitting over two separate modules, an inductive module and a deductive module, and potentially allows more sophisticated induction algorithms to be employed. RichProlog provides an interface to queries of the form $\exists x \forall y \varphi$ and invokes (and switches between) the two modules appropriately.

**RichProlog Vs. ILP** The way RichProlog handles $\Sigma_2$-type queries allows it to simulate any ILP system (the reverse is not true), since an ILP algorithm must somehow generate a hypothesis $H$ and somehow verify it (refer to Algorithm 1). However, RichProlog is more powerful than an ILP system in the sense that it does not just handle the $\Sigma_2$-type queries, which involves learning. It answers all types of queries in the deductive-inductive hierarchy up to $\Sigma_2$-type queries and thus is a general logic programming environment. Although some ILP implementations also involve deduction, it is often integrated to support the inductive process. However, since ILP systems are typically targeted for specific inductive tasks, they are thus more efficient.

**RichProlog Vs. DIR** RichProlog performs deductive-inductive reasoning when the exact order of switching between deduction and induction is fully specified, i.e. it handles queries only in this alternating form. Therefore RichProlog is lim-
5.5. DEDUCTIVE AND INDUCTIVE RESOLVENTS

uted in use if the order is not known in advance. That is, although RichProlog allows deduction to be executed alongside induction, it does not enable deduction to be executed as a result of induction, or vice versa. Therefore, in order to perform deductive-inductive inferences in general—beyond solving $\Sigma_2$-type of queries—what is needed is a flexible way of switching between deductive and inductive processes. In contrast to RichProlog, deductive-inductive resolution (DIR) enables the switching between deduction and induction through a number of inferential relations, detailed in the following section.

5.5 Deductive and Inductive Resolvents

We now turn to the question of whether a given formula $\gamma$ is a result of the recursive application of deductive and inductive inferences. To achieve this, we start by defining the concepts of deductive and inductive resolvent of a given theory $T$, based on nested deductive-inductive inferences.

**Definition 11** (Deductive Resolvent). $\gamma$ is a deductive resolvent of theory $T$, denoted $T \vdash_{dr} \gamma$, iff $\gamma \in \Sigma_n(T) \land \gamma \not\in \Pi_{n-1}(T)$ for some $n > 0$.

**Definition 12** (Inductive Resolvent). $\gamma$ is an inductive resolvent of theory $T$, denoted $T \vdash_{ir} \gamma$, iff $\gamma \in \Pi_n(T) \land \gamma \not\in \Sigma_{n-1}(T)$ for some $n > 0$.

That is, $\gamma$ is a deductive resolvent of $T$ if and only if a deductive inference results in a series of deductive-inductive inference steps, $(\Sigma(T) \Rightarrow \Sigma_n(T))$, via the inferential relations, which subsequently derives $\gamma$. Inductive resolvent is defined in a similar manner. The above two definitions are broad in a sense that for $\gamma$ to be a resolvent, it does not matter in which order inferences are performed, as long as $\gamma$ is derived in the end.

Note that being a deductive resolvent is not equivalent to being entailed in the usual logic sense, i.e., $T \vdash_{dr} \gamma \not\iff T \models \gamma$. Likewise, $T \vdash_{ir} \gamma \not\iff T \models_I \gamma$. 

71
Instead, $T \vdash_{dr} \gamma$ and $T \vdash_{ir} \gamma$ have the respective semantics stated in the following theorems.

Take the example in Figure 5.1, $reachable(b,d)$ is eventually deduced from the theory $(T_1 \cdots T_4)$, via a series of deductive-inductive steps. According to Definition 11 and 12, $reachable(b,d)$ is a deductive resolvent of the theory, i.e. $T_1 \cup T_2 \cup T_3 \cup T_4 \vdash_{dr} reachable(b,d)$.

**Theorem 1 (Deductive Resolvent).** The deductive resolvent, as defined in Definition 11, has the following equivalence relation:

$$
T \vdash_{dr} \gamma \equiv [T \models \gamma] \\
\vee [(T \vdash_{dr} E) \land (E \cup T \vdash_{dr} \gamma)] \\
\vee [(T \vdash_{ir} H) \land (H \cup T \vdash_{dr} \gamma)]
$$

**Proof.** The base case is obtained directly from Definition 8, i.e., when $n = 1$ and $\gamma \in \Sigma_1(T)$. In the first general case, $E \cup T \vdash_{dr} \gamma$ indicates $\gamma$ is a deductive resolvent of $E \cup T$. Since $E$ is also a deductive resolvent of $T$, according to Definition 8, $E$ is incorporated into the theory $T$. Therefore, if $\gamma$ is a deductive resolvent of $E \cup T$, then $\gamma$ is a deductive resolvent of $T$, or $T \vdash_{dr} \gamma$. Similar for the second general case. 

In other words, the first general case says $\gamma$ is a deductive resolvent of $T$ iff $E$ can be deduced from $T$ after a number of steps, and $\gamma$ can be deduced from $E \cup T$ after a number of steps. Similarly, the second general case states that $H$ is induced in an intermediate step, and $\gamma$ is deduced in the final step. The two general cases identify the intermediate $E$ and $H$. They are resulted from nested deductive-inductive inferences and can be viewed as instances of the first two inferential relations identified in Section 5.2. The two general cases collectively state that after a number of intermediate inferential steps, which result in $E$ or $H$ being deduced or induced, $\gamma$ is eventually deduced from $E \cup T$ or $H \cup T$ after a number of inference steps.
5.6. DEDUCTIVE-INDUCTIVE RESOLUTION EXAMPLE

Theorem 2 (Inductive Resolvent). The inductive resolvent, as defined in Definition 12, has the following equivalence relation:

\[
T \vdash_{ir} \gamma \equiv [T \models I \gamma] \\
\lor [(T \vdash_{dr} E) \land (E \cup T \vdash_{ir} \gamma)] \\
\lor [(T \vdash_{ir} H) \land (H \cup T \vdash_{ir} \gamma)] \\
\lor [(T \vdash_{ir} \gamma) \land (\gamma \cup T \vdash_{dr} E)]
\]

Proof. Analogous to the proof of Theorem 1. The base case comes directly from Definition 9. The three general cases are results of \( H \) being incorporated into theory \( T \) as indicated by Definition 9.

The above theorems state that deductive (or inductive) resolvents are obtained through deductive (or inductive) inferences (the base case) or deductive-inductive inferences (the general cases) as a consequence of applying inferential relations. Deductive-inductive resolution thus concerns answering whether a given formula \( \gamma \) is a deductive or inductive resolvent of a given theory \( T \), i.e., whether \( T \vdash_{dr} \gamma \) or \( T \vdash_{ir} \gamma \).

5.6 Deductive-Inductive Resolution Example

If it is known that Alan eats apples, bananas, oranges and grapes, can we infer that Alan eats peaches? Under a deductive reasoning framework, the answer is ‘No, of course not’. However, if scientists have just found a new species in the jungle that happens to eat apples, bananas, oranges and grapes, it is reasonable to assume this new species eats all sorts of fruits and thus peaches too.

This form of reasoning should be allowed under a deductive-inductive reasoning framework. If the hypothesis that Alan eats fruit explains the situation without contradicting any known evidence, it ought to be accepted as a valid one. Therefore, under a deductive-inductive framework, the answer may well be ‘Yes’.
CHAPTER 5. THE DIR FRAMEWORK

Here we define such a logic program under the DIR framework. Assume the predicate \( eat(P, F) \) stands for person \( P \) eats food \( F \) and \( belongs(F, C) \) stands for \( F \) belongs to class \( C \). The logic program is organized around the five elementary inferential relations defined in Section 5.2 as follows:

1. \[ \Sigma(T) \equiv \Sigma(\Sigma(T)) : eat(P, F) \leftarrow hungry(P) \]
2. \[ \Sigma(T) \equiv \Sigma(\Sigma(T)) : eat(P, F) \leftarrow eat(P, C) \land belongs(F, C) \]
3. \[ \Sigma(T) \equiv \Sigma(\Pi(T)) : eat(P, F) \leftarrow induce(eat/2) \land eat(P, F) \]
4. \[ \Pi(T) \equiv \Sigma(\Pi(T)) : induce(eat/2) \leftarrow ILP(eat/2, H) \land verify(H) \]
5. \[ belongs(apple, fruit), \ldots, belongs(peach, fruit) \]
6. \[ eat(alan, apple), eat(alan, banana), \ldots \]
7. \[ \neg hungry(alan) \]

The first two rules are deductive ones. The third rule causes a deductive query to be transformed into an inductive then deductive one, which says “when answering the deductive query \( eat(P, F) \), try inducing the definition that we do not know yet about \( eat/2 \) and then try the same query again”. The fourth rule says “when inducing the definition for \( eat/2 \), execute an ILP process to generate a hypothesis and verify the hypothesis against known evidences”. The rest rules are known facts.

Let’s have a look at how the program executes given \( \leftarrow eat(alan, peach) \) as the query. First, there is no direct evidence that answers the query. The first and second (deductive) rules will both fail for obvious reasons. When it comes to the third rule, it causes an inductive process to be executed, expecting that new definition be learnt about \( eat/2 \) after the execution of the inductive process. After that, the same deductive query is invoked again. The inductive process causes the fourth rule to match and execute which does the actual hypothesis generation and verification. The inductive process deserves a closer look.
5.6. DEDUCTIVE-INDUCTIVE RESOLUTION EXAMPLE

The inverse resolution (IR) algorithm generates hypothesis $H = eat(\text{alan, fruit})$.

Called by induce(eat/2), the ILP(eat/2, $H$) predicate invokes the actual inductive logic programming algorithm that generate a candidate hypothesis $H$. In this example, the ILP algorithm is an implementation of the inverse resolution (IR) algorithm. An IR algorithm requires two ingredients: theory (background knowledge) and examples. IR constructs and outputs the hypothesis $H = eat(\text{alan, fruit})$, based on theory eat/2 (i.e. rules 1 and 2 above) and examples about eat/2 (i.e. eat(\text{alan, apple})), as shown in Figure 5.2.

After the hypothesis, $H$, is generated by ILP(eat/2, $H$) it is subjected to verification, verify($H$). In this example, there are actually two possible hypotheses, $H_1 = eat(\text{alan, fruit})$ and $H_2 = hungry(\text{alan})$, both of which can be derived by the IR process. The negative examples play an important role here since we assume the falsification principle “a hypothesis is valid unless it is proven wrong”. In this example, the only valid hypothesis is $H_1$ ($H_2$ is refuted by the negative example of itself, rule 7).

After the inductive process, rule 3 succeeds and subsequently causes the same query to execute again. This time (deductive) rule 2 succeeds, confirming the success of query $\leftarrow eat(\text{alan, peach})$—in other words Alan does indeed eat peach. During the execution of query $\leftarrow eat(\text{alan, peach})$, all three elementary inferential relations have taken place and the original $\Sigma(T)$ query has been transformed to $\Sigma(\Sigma(\Pi(T)))$, resulting in a series of deductive-inductive inferences.

Here are some further remarks regarding the DIR logic program at the start
of this example. Since the DIR mechanism involves employing logic programming and inductive logic programming techniques to do the respective inference, in practice, a DIR system may be constructed using existing systems, such as Prolog and Aleph, rather than from scratch. Therefore, a DIR program may appear as rules to instruct and switch between the two systems. In this example, this perspective is taken. The first two rules may be viewed as clauses instructing Prolog to do deductive inference. The third and fourth rule invoke the inductive counterpart, Aleph, and switch between the two systems. The rest facts are shared between Prolog and Aleph. Due to this, the third and fourth rule are described at a meta-level.

5.7 Deductive-Inductive Resolution with Collaboration

In the previous sections, the core of the deductive-inductive resolution framework have been given. In this section, we generalize deductive-inductive resolution for collaborative multi-agent settings. We start by restating Theorems 1 and 2, making explicit reference to an individual agent’s theory, to allow for collaborative inference among agents. Throughout this section the assumption is made that agents’ theories are consistent, i.e. $\bigcup_{a_i \in A} T(a_i) \not\models \Box$.

**Theorem 3** (Collaborative Deductive Resolvent). $\gamma$ is a collaborative deductive resolvent of $T(a_i)$ iff $T(a_i) \vdash_{dr} \gamma$, where

$$T(a_i) \vdash_{dr} \gamma \equiv [T(a_i) \models \gamma]$$

$$\lor \exists a_j [(T(a_j) \vdash_{dr} E) \land (E \cup T(a_i) \vdash_{dr} \gamma)]$$

$$\lor \exists a_j [(T(a_j) \vdash_{ir} H) \land (H \cup T(a_i) \vdash_{dr} \gamma)]$$

**Proof.** Omitted. This theorem is a generalization of Theorem 1 for a group of agents, therefore the proof follows closely from the earlier single agent version.
Theorem 4 (Collaborative Inductive Resolvent). $\gamma$ is a collaborative inductive resolvent of $T(a_i)$ iff $T(a_i) \vdash_{ir} \gamma$, where

$$T(a_i) \vdash_{ir} \gamma \equiv [T(a_i) \models_I \gamma]$$

$$\vee \exists a_j [(T(a_j) \vdash_{dr} E) \land (E \cup T(a_i) \vdash_{ir} \gamma)]$$

$$\vee \exists a_j [(T(a_j) \vdash_{ir} H) \land (H \cup T(a_i) \vdash_{ir} \gamma)]$$

$$\vee \exists a_j [(T(a_j) \vdash_{ir} \gamma) \land (\gamma \cup T(a_i) \vdash_{dr} E)]$$

Proof. Omitted (as per above).

The above two theorems express the fact that, in systems involving a group of agents, agents are no longer restricted to answer deductive/inductive queries based only on their own theories. Instead they may, or inevitably have to, engage other agents in the group to tackle the queries together. Since the above two theorems are generalizations of Theorems 1 and 2 for a group of agents, the proof is almost identical to those in the earlier versions and are thus omitted.

Note that when there is only one agent in the group, or when $a_i$ and $a_j$ happen to refer to the same agent, the above theorems collapse to their respective forms in single agent case, which do not require a collaborative effort. Because the above theorems capture both single-agent and multi-agent cases, in later parts of the thesis, the word ‘collaborative’ may be dropped when referring to them.

5.8 Collaboration and Relationship to CollabLP

In the previous section, we have defined collaborative deductive-inductive resolution in multi-agent systems as a collaborative endeavor and we have described from a static viewpoint that answering logic queries in those settings involves a team effort (see Theorems 3 and 4). So far, it has been left unanswered in what
form collaboration actually takes place. In this section, we will first tackle this and then point out the connection with collaborative logic programming (CollabLP) as defined in Chapter 4.

By collaboration, or team effort, we mean either the agents’ distributed knowledge needs to be somehow combined or the task needs to be somehow decomposed such that each individual agent tackles a sub-task and they interact to combine their results in a meaningful way. As far as collaborative logic programming (CollabLP) problems are concerned, collaboration can take place using only the latter approach. More restrictively, communication in CollabLP must only be in the form of logic queries derived from the inductive or deductive consequences of agents’ respective theories (refer to Definition 7).

For a deductive query, for instance, when agent \(a_i\) is chosen to prove \(T(a_i) \vdash_{dr} \gamma\) and while doing so, it discovers that \(T(a_i) \cup \gamma' \vdash_{dr} \gamma\) and \(\gamma'\) can not be proven by itself. It subsequently communicates a sub-query, \(\gamma'\), trying to find an agent \(a_j\) who can prove it, based on its own theory, \(T(a_j) \vdash_{dr} \gamma'\) (relation 1), or, if an agent can induce \(\gamma'\), such that \(T(a_j) \vdash_{ir} \gamma'\) (relation 2). In this way theories themselves are not being communicated, which is forbidden by CollabLP. Instead, agents’ collaborative engagements happen through the communication of deductive/inductive logical consequences of their theories, in the form of logic programming queries that either succeed or fail. The communication involved in answering such deductive queries is shown diagrammatically below:

![Diagram showing communication process](image)

It is worth pointing out some finer aspects that are implicit in the diagram.
5.8. COLLABORATION AND RELATIONSHIP TO COLLABLP

above:

1. Agent $a_i$ and $a_j$ can refer to the same agent;

2. Agent $a_i$ may communicate its query to all other agents until the query is satisfied;

3. Agent $a_j$ may also involve other agents, say $a_k$, in the same way and this process goes on recursively; and

4. Agent $a_j$ may return positive/negative replies (according to whether the query succeeds) or issue a further query $\gamma''$ back to the initiator as a reply.

The next key question to answer is how the collaborative effort based on deductive-inductive resolution offers a solution to the CollabLP problem. In the remaining of this section, we outline some theoretical results and leave the detailed proofs to Chapter 6.

Recall that the CollabLP problem concerns deciding for a given $\gamma$ if $T \land H \models \gamma$, where $T = \bigcup_{a_i \in A} T(a_i)$ (refer to Definition 7). The question is then to ask, if one agent is able to prove (based on DIR) that $\gamma$ is a collaborative deductive resolvent, or $T(a_i) \vdash_{dr} \gamma$, how does that relate to $\gamma$ being a deductive consequence of the group theory $T$? In other words, the question is: what does it mean for $T \land H \models \gamma$ if $\exists a_i T(a_i) \vdash_{dr} \gamma$?

The answer is twofold. First, it is rather anticipated that the latter implies the former, or $\exists a_i T(a_i) \vdash_{dr} \gamma \Rightarrow T \land H \models \gamma$. This is because, if there exists an agent who succeeds in proving $\gamma$, alone or through collaboration (possibly with hypothesis, $H$), $\gamma$ must be a deductive consequence of the group theory (possibly with hypothesis). This leads to the following theorem, which says DIR is sound with respect to solving the CollabLP problem:

**Theorem 5** (Soundness of Deductive-Inductive Resolution). *Deductive-Inductive Resolution is Sound with respect to solving CollabLP problems, such that*

$$\exists a_i T(a_i) \vdash_{dr} \gamma \Rightarrow T \land H \models \gamma$$
CHAPTER 5. THE DIR FRAMEWORK

Proof. Requires modal treatment, the full proof is provided in Section 6.5. □

Hence, the real question is whether the implication goes in both directions. That is to ask, when $\gamma$ is a deductive consequence of the group theory, can it always be proven, through the collaborative effort of DIR, whether $T \land H \models \gamma \Rightarrow \exists a_i T(a_i) \vdash_{dr} \gamma$.

It turns out that this is not always the case. Again, the situation is twofold. In the case when group theory is complete, i.e. hypothesis $H$ is empty, the problem of $T \land H \models \gamma$, or rather $T \models \gamma$, can always be answered through the collaborative effort of DIR. This is because, for a given deductive query $\gamma$, we can always find an agent who can prove part of it. Because the group theory is complete, there must exist another agent who can prove part of the sub-query, even if not the whole sub-query. This process goes on until $\gamma$ is proven in its entirety.

In the general case, however, $T \land H \models \gamma$ does not guarantee $\exists a_i T(a_i) \vdash_{dr} \gamma$ if $H$ is not empty. This has to do with the intrinsic difficulty of induction. Put another way, being able to induce $H$ based on the combined theory, $T$, does not guarantee that the same hypothesis is inducible in a distributed fashion or by an individual agent. In order to guarantee $T \land H \models \gamma$ to be answerable through the collaborative effort of DIR, the global hypothesis $H$ needs to be separably inducible.

**Definition 13** (Separably Inducible). A hypothesis $H$ is separably inducible iff $H = \emptyset$, or $H = H_1 \cup H_2 \cup \cdots \cup H_n$ where $\forall i \exists a_j T(a_j) \vdash_{ir} H_i$.

**Theorem 6** (Completeness of Deductive-Inductive Resolution). Deductive-Inductive Resolution is complete with respect to solving CollabLP problems iff $H$ is separably inducible, such that

$$T \land H \models \gamma \Rightarrow \exists a_i T(a_i) \vdash_{dr} \gamma$$

Proof. Refer to Section 6.6. □
Definition 13 relies on the omniscient view that global hypothesis, $\mathcal{H}$, are separably inducible if and only if there exist agents capable of inducing the set of local hypothesis, $H_1, \ldots, H_n$, where the union of this set is equivalent to $\mathcal{H}$. Note that the inducibility of each hypothesis, $H_i$, from a local perspective, depends on the specific inductive procedure employed and theory equipped by each agent. Being separably inducible, however, does not mean each of $H_i$ must be induced by an individual agent alone. Collaboration in the form of finding the collaborative inductive resolvent is permissible.

It is worth reemphasizing the dual roles that inductive inference plays in the whole DIR strategy. First, it provides a way to accommodate incomplete theory by allowing missing knowledge ($\mathcal{H}$) to be synthesized. Second, and importantly, it facilitates collaboration by establishing connection between agents’ reasoning processes.
Chapter 6

Deductive-Inductive Resolution
from a Modal Perspective

6.1 Overview

In this chapter, a modal treatment of the deductive-inductive resolution (DIR) framework is provided, which allows for an epistemic analysis of agents during deductive-inductive resolution. The DIR approach is subsequently proven to be sound and complete with respect to solving the CollabLP problem.

In order to establish the relation between DIR and CollabLP, we introduce an important intermediate construct called the ‘universe’, based on the possible-world structure or Kripke structure (Kripke, 1963). We start by showing how every DIR inferencing scenario has a corresponding universe structure associated, and subsequently, we show how instances of the universe structure is semantically equivalent to solutions to CollabLP problems. This forms the basis of the proof to the soundness and completeness theorems (refer to Theorem 5 and 6 in Section 5.8).
6.2 The Universe Structure

We define for each agent, \( a_i \), a model \( M(a_i) \) associated with it, based on the possible-world structure, or Kripke structure (refer to Section 2.3). We define the model as a 3-tuple, \( M(a_i) = (S, \pi, R) \), over the language \( L \) where:

- \( S \) is the set of states, each having some logic formulae (ground literals) associated with it (evaluate to \text{true} in that state);

- \( \pi \) is called the interpretation which associates with each state in \( S \) a truth assignment of formulae \( \varphi \in L \), i.e. \( \pi : (S, L) \mapsto \{\text{true}, \text{false}\} \);

- \( R \) is the possibility relation which is a set of binary relations over \( S \). \( (s, t) \in R \) if and only if there exists a transition from \( s \) to \( t \).

For all \( t \) that \( (s, t) \in R \), if \( s \) is the current state then \( t \) can be viewed as the next state agent \( a_i \) considers possible. The transitions move the agent from its current state to one of the possible next states. The transitions can be viewed as a result of either deductive or inductive inferences. At state \( s \), agent considers any state \( t \) as possible next state and does not know in which state it would end up before an inference is performed. \( R \) may be further classified as either deductive or inductive transitions. That is \( R = R_D \cup R_I \) where \( R_D \) is the set of deductive transitions, which are transitions caused by deductive inferences. Similarly, \( R_I \) are the inductive transitions. \( M(a_i) \) is consequently a binary tree structure.

We then define the universe \( U = (M, R) \), where \( M \) is the collection of all models, i.e. \( M = \bigcup_{a_i \in A} M(a_i) \). \( R \) is a set of inter-model transitions which establish connections between states in different models. \( R \) transitions reflect interactions between agents, which could occur at any stage during inferencing and which allow agents’ knowledge to be transferred. A universe structure visualized as a directed graph is shown in Figure 6.1.
6.2. THE UNIVERSE STRUCTURE

The semantics of the universe is characterized by the following four axioms:

\[ A_0 : \ (s, t) \in R \land \pi(s, \varphi) \implies \pi(t, \varphi) \]
\[ A_1 : \ \pi(s, T) \land (s, t) \in R_D \implies \exists \varphi (T \models \varphi \land \pi(t, \varphi)) \]
\[ A_2 : \ \pi(s, T) \land (s, t) \in R_I \implies \exists \varphi (T \models_I \varphi \land \pi(t, \varphi)) \]
\[ A_3 : \ (s, t) \in \mathcal{R} \implies \exists \varphi (\pi(s, \varphi) \land \pi(t, \varphi)) \]

For simplicity, all variables in the axioms are universally quantified by default, unless explicitly stated. \( A_0 \) accounts for the fact that knowledge persists (agents never forget things). \( A_1 \) and \( A_2 \) collectively assert that as an outcome of inference, if \( \varphi \) is the new knowledge inferred, \( \varphi \) holds in the new state. \( A_3 \) states that transition between states in two models allows knowledge to be transferred across.

The basic universe structure is readily extensible to incorporate epistemic reasoning capability. This can be easily done by adding the standard equivalence relation \( K \) from literature (Meyer & Hoek, 1995) into the agent model, such that \( M(a_i) = (S, \pi, R, K) \). This way, agents’ mental status can be analyzed epistemically using the standard set of knowledge axioms, as defined in Section 2.3, during deductive-inductive resolution. An extended universe structure may be visualized as in Figure 6.2, in which all dotted transitions are the \( K \) relations.

For the rest of this chapter, however, the discussion will be based on the basic universe structure.
6.3 Representing Collaborative Inference

We are now in the position to represent collaborative deductive/inductive inferencing scenarios using the defined universe structure. We focus our discussion on collaborative deduction here, as the inductive counterpart is very similar. Recall that collaborative deduction is described by Theorem 3 as follows:

\[ T(a_i) \vdash_{dr} \gamma \equiv [T(a_i) = \gamma] \]

\[ \lor \exists a_j [(T(a_j) \vdash_{dr} E) \land (E \cup T(a_i) \vdash_{dr} \gamma)] \quad (relation \ 1) \]

\[ \lor \exists a_j [(T(a_j) \vdash_{ir} H) \land (H \cup T(a_i) \vdash_{dr} \gamma)] \quad (relation \ 2) \]

Assume \( \forall a_i \pi(s(a_i, 0), T(a_i)) \), i.e. for every agent \( a_i \), theory \( T(a_i) \) holds in its initial state.

Figure 6.3 illustrates how collaborative deductive inference is represented using the universe structure in general. The base case corresponds to a universe with a single agent model and an empty \( R \) (shown on the left), while the two general cases involve a collection of agent models joined by a set of \( R \) transitions (shown in the middle and on the right).

The formulae that hold at each state are denoted in the figure. Note that as a result of \( R_D \) or \( R_I \) transitions, apart from some new formulae being held at the finishing states (due to \( A_1 \) and \( A_2 \)), whatever holds at the starting states still hold...
6.4. COMPRESSING THE UNIVERSE

Figure 6.3: Representing collaborative deductive inference as described by Theorem 3 using the universe structure. **LEFT:** The base case is represented as a universe with a single agent model and an empty \( \mathcal{R} \). **MIDDLE:** Relation 1 is represented as a collection of agent models joined by a set of \( \mathcal{R} \) transitions. **RIGHT:** Relation 2 is represented similarly to Relation 1.

at the finishing states (due to \( A_0 \)). As a result of \( \mathcal{R} \) transitions, some formulae hold in both the starting and finishing states (due to \( A_3 \)). These axioms thus capture the semantics of deductive-inductive resolution.

Note also that the structure is recursive in the general cases, represented by the dotted line at the left, top and bottom of the structure, reflecting that collaborative inference may involve any number of agents in a recursive way.

In all cases, as can be seen in Figure 6.3, \( T(a_i) \vdash_{dr} \gamma \text{ iff } \exists s \exists t (s, t) \in R_D \land \pi(t, \gamma) \). That is, whenever \( \gamma \) is a collaborative deductive resolvent of \( T(a_i) \), there exists in the universe a deduced state \( t \) in which \( \gamma \) holds.

6.4 Compressing the Universe

We now define an algorithmic approach which takes an existing universe and reduces the number of states to two by combining all except the final state into one. Importantly, this process preserves the semantics of the universe.

The process looks at two states joined together by a transition at a time. In
CHAPTER 6. DIR FROM A MODAL PERSPECTIVE

Algorithm 2 Algorithm for compressing the universe

1: repeat
2: choose a root state $s$
3: if $(s, t) \in R_D$ then
4: remove $t, \forall t' (t, t') \in R$ add $(s, t')$ to $R$
5: else if $(s, t) \in R_I$ then
6: remove $s$
7: else if $(s, t) \in R$ then
8: remove $s, \forall \varphi \pi(s, \varphi)$ assert $\pi(t, \varphi)$
9: end if
10: until only two states left
order to maintain generality and thus reasoning capability, the process aims to replace the two old states by a single new state that contains a more general set of formulae than either of the two original states. Recall that a set of formulae $T$ is said to be more general than $T'$ iff $T \models T'$.

For a deductive transition, the starting and finishing states are equally general, such that either can be chosen without sacrificing generality. The starting state, however, carries a subset of formulae of what the finishing state carries and is thus chosen as the resulting state. For an inductive transition, the finishing state is kept as it is more general than the starting state, in the sense that whatever is entailed by the starting state is entailed by the finishing state but not vice versa. For an $R$ transition, states are combined and the combined state is more general than either of the individual state. In all cases, the resulting state entails whatever is entailed by the starting or finishing state. Algorithm 2 defines this reduction process in detail, including rewiring the transitions.

The reduction process starts by choosing an arbitrary root state and repeats until only two states are left. Although the number of states decrease at every step, while applying the algorithm, the semantics of the universe is maintained. That is, the axioms $A_0 - A_3$ still hold at every state in the new universe. In other words, this process does not restrict or broaden the reasoning capability of any agent at any stage by ensuring that each resulting state is no less general than the ones being eliminated.

It can be observed that whenever there exists in the old universe a final state in which $\gamma$ holds, there exists a such state in the corresponding reduced universe (because the process halts before reaching that final state).

Figure 6.4 shows this compression process for a typical collaborative inference scenario (on the left), with some intermediate steps omitted. To keep it concise, not every true formula is listed at each state. Note that as a result of this compression process, only the theories and hypotheses are kept (the $E$s are all dropped).
CHAPTER 6. DIR FROM A MODAL PERSPECTIVE

Figure 6.4: Compressing the universe into two states while maintaining the semantics of the universe. **Left:** a typical collaborative inference scenario. **Middle:** an intermediate step during compression. **Right:** the resulting universe with only two states.

6.5 Soundness of Deductive-Inductive Resolution

*Proof of Theorem 5 (Soundness of Deductive-Inductive Resolution).* Recall that the soundness theorem states that

\[ \exists a_i T(a_i) \vdash_{dr} \gamma \Rightarrow T \land H \models \gamma \quad (6.1) \]

As have been discussed in an earlier section (with reference to Figure 6.3), every collaborative deductive inference scenario corresponds to a unique universe structure in which there exists a deduced state and \( \gamma \) holds in that state. That is, \( \exists a_i T(a_i) \vdash_{dr} \gamma \) implies \( \exists s \exists t (s, t) \in R_D \land \pi(t, \gamma) \). Thus, what we need to prove is: whenever there exists a deduced state in which \( \gamma \) holds, \( \gamma \) is entailed by the group theory \( T \) (plus hypotheses \( H \)), or

\[ \exists s \exists t (s, t) \in R_D \land \pi(t, \gamma) \Rightarrow T \land H \models \gamma \quad (6.2) \]

The proof proceeds as follows. For each of the universes that meet the former condition, we can obtain a reduced truth-preserving version through the compression procedure defined by Algorithm 2. In addition, we know that in the reduced universe there are only two states, joined by an \( R_D \) relation (see Figure 6.4 for example). In the finishing state, \( \gamma \) holds and in the starting state, the formulae
can be expressed as $T \land H$, where $T$ is the union of all $T_i$ that have participated in the collaborative inference and $H$ is a collection of the hypotheses induced in the process. Since the semantics of the universe is carefully maintained through the compression process, the axioms $A_0 - A_3$ still apply. Due to axiom $A_1$, the semantics of the reduced universe reads: $T \land H \models \gamma$.

The remaining question is how $T \land H \models \gamma$ relates to $T \land H \models \gamma$. $H$ is, by definition, the minimum subset of $H$ such that $T \land H \models \gamma$ still holds. Hence, if $T \land H \models \gamma$ then $T \land H \models \gamma$. $T$ is the collection of agents’ theories which have participated in the inferencing and is a subset of $T$. Therefore, if $T \land H \models \gamma$ then $T \land H \models \gamma$. Finally, we have proven that for any universe in which $\exists s \exists t (s, t) \in R_D \land \pi(t, \gamma)$ holds, $T \land H \models \gamma$ also holds, i.e. Equation (6.2) holds and hence Equation (6.1) holds. Therefore, Deductive-Inductive Resolution is Sound.

Note that not all $H$s in $H$ must appear in $H$. In fact, it may be the case that none of the $H$s in $H$ is needed, in which case $T \models \gamma$. In this case, $H$ is simply $\emptyset$ although $H$ is not $\emptyset$. This is due to the communication restriction imposed by CollabLP, such that hypotheses are invented merely for the purpose of interaction and are not needed in proving $T \models \gamma$, once the theories are congregated.

### 6.6 Completeness of Deductive-Inductive Resolution

**Proof of Theorem 6 (Completeness of Deductive-Inductive Resolution).** Recall that the completeness theorem states that

$$T \land H \models \gamma \Rightarrow \exists a_i T(a_i) \vdash_{dr} \gamma$$

iff $H$ is separably inducible \( (6.3) \)

That is, deductive-inductive resolution is complete when $H = \emptyset$, or when $H = H_1 \cup H_2 \cup \cdots \cup H_n$, where $\forall i \exists a_j T(a_j) \vdash_{ir} H_i$, and not complete otherwise. We prove these cases separately.
Case 1: \( \mathcal{H} = \emptyset \). When \( \mathcal{H} = \emptyset \), Equation (6.3) simplifies to

\[
\mathcal{T} \models \gamma \Rightarrow \exists a_i \mathcal{T}(a_i) \vdash_{dr} \gamma
\]

(6.4)

Because \( \mathcal{T} \) is a set of Horn clauses and SLD-resolution is known to be complete for Horn clauses (Lloyd, 1987), therefore, there exists an SLD-derivation of \( \gamma \) from \( \mathcal{T} \) whenever \( \mathcal{T} \models \gamma \). Since \( \mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_n \), this means \( \mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_n \models \gamma \) has an SLD-derivation. Let \( E_1, E_2, \cdots, E_n, \gamma \) be the actual SLD-derivation, where each \( E_i \) is a resolvent of \( E_{i-1} \) and some \( T_i \in \mathcal{T}, E_{i-1} \land T_i \models E_i \) holds for each resolvent. This gives rise to the following argument:

\[
\begin{align*}
\mathcal{T} & \models \gamma \\
\Rightarrow & \quad \mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_n \models \gamma & \text{def. of } \mathcal{T} \\
\Rightarrow & \quad \mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_n \vdash_{sld} \gamma & \text{SLD-RES is complete} \\
\Rightarrow & \quad \mathcal{T}_i \vdash_{sld} E_i \land E_i \cup \mathcal{T}_j \vdash_{sld} E_j \land \cdots \land E_n \cup \mathcal{T}_n \vdash_{sld} \gamma & \text{def. of SLD-RES} \\
\Rightarrow & \quad \mathcal{T}_i \models E_i \land E_i \cup \mathcal{T}_j \models E_j \land \cdots \land E_n \cup \mathcal{T}_n \models \gamma & \text{SLD-RES is sound}
\end{align*}
\]

Notice that the last line of the above derivation consists of a conjunction of entailments, where each conjunct contains exactly one \( T_i \). It can be seen as an instance of collaborative deductive inference (refer to Theorem 3) and this instance uses relation 1 exclusively and is purely deductive. Therefore, we have proven that every instance of \( \mathcal{T} \models \gamma \) has a (purely deductive) instance of \( \mathcal{T}(a_i) \vdash_{dr} \gamma \) which answers it. Hence, Equation (6.4) holds.

This, however, does not necessarily mean that this deductive instance is the unique instance which answers \( \mathcal{T} \models \gamma \). There may be other instances involving a combination of deductive and inductive inferences, even though the induced hypotheses do not contribute to \( \mathcal{H} \). Nevertheless, it is certainly the case that at least one such instance exists, i.e. the purely deductive one.

The last line of the above derivation can also be represented using a universe structure as previously defined, with all \( E \)s being used for communication. That is, every instance of \( \mathcal{T} \models \gamma \) has at least one universe in which all transitions
are either $R_D$ or $\mathcal{R}$ transitions. The universe will look something like the one in Figure 6.5.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure6.5.png}
\caption{Every CollabLP problem where $\mathcal{T} \models \gamma$ has a solution based on collaborative deductive-inductive resolution that can be represented by a universe, in which all transitions are either $R_D$ or $\mathcal{R}$ transitions.}
\end{figure}

Case 2: $\mathcal{H}$ is Separably inducible. Case 1, as we have just seen, is in fact a special case under the separably inducible assumption (refer to Definition 13). Now we prove in general when $\mathcal{H}$ is separably inducible, Equation (6.3) holds. The proof for this case is built on top of the proof for Case 1. Since we have seen in Case 1 that

\begin{equation}
\mathcal{T} \models \gamma \Rightarrow T_i \models E_i \land E_i \cup T_j \models E_j \land \ldots \land E_n \cup T_n \models \gamma \quad (6.5)
\end{equation}

for the same reason, when $\mathcal{H} = H_i \cup H_j \cup \ldots \cup H_n$, the following holds

\begin{equation}
\mathcal{T} \land \mathcal{H} \models \gamma \Rightarrow T_i \cup H_i \models E_i \land E_i \cup T_j \cup H_j \models E_j \land \ldots \land E_n \cup T_n \cup H_n \models \gamma \quad (6.6)
\end{equation}

Of course, the number of $T$s and $H$s need not be equal (some $H$s are dummies). Using the universe structure, Equation (6.6) is represented in Figure 6.6, in the same way as Equation (6.5) is represented in Figure 6.5.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure6.6.png}
\caption{universe representation of Equation (6.6) in the same way as Equation (6.5) is represented in Figure 6.5.}
\end{figure}
Figure 6.7: Every CollabLP problem where $\mathcal{H}$ is separably inducible has a solution based on collaborative deductive-inductive resolution that can be represented by a universe, in which all $E$s and $H$s are derivable and communicated.

Unlike Figure 6.5, Figure 6.6 does not correspond to an instantiation of Theorem 3 by default (since the $H$s are coming from nowhere), unless all $H$s are instantiations of Theorem 4. The separably inducible assumption makes exactly this claim. According to this assumption, $\forall i \exists a_j \ T(a_j) \vdash_r \ H_i$, which in turn suggests for each $H_i$, $\exists s \exists t \ (s, t) \in R_I \land \pi(t, H_i)$. That is, for each $H_i$, there exists two states joined by an $R_I$ relation and $H_i$ holds in the finishing state. Therefore, under the separably inducible assumption, Figure 6.6 becomes Figure 6.7, where all the $H$s are inducible according to Theorem 4 and are communicated. Unlike Figure 6.6, Figure 6.7 can now be viewed as an instantiation of Theorem 3. Hence, whenever $\mathcal{H}$ is separably inducible, there always exists an instance of collaborative deductive-inductive inference that answers the CollabLP problem. Hence, Equation (6.3) holds.

Case 3: $\mathcal{H}$ is not separably inducible. When $\mathcal{H}$ is not separably inducible, all we need to give is a counterexample which falsifies Equation (6.3), in which $T \land \mathcal{H} \models \gamma$ holds but $\exists a_i \ T(a_i) \vdash_{dr} \gamma$ does not. Here is the counterexample, assuming inverse resolution is the inductive procedure employed:
6.6. COMPLETENESS OF DEDUCTIVE-INDUCTIVE RESOLUTION

\[ G = \{a_1, a_2\} \]
\[ T = \{\text{reachable}(A, C) \leftarrow \text{reachable}(A, B) \land \text{reachable}(B, C)\} \]
\[ T(a_1) = \{\text{reachable}(a, b), \text{reachable}(c, d)\} \cup T \]
\[ T(a_2) = \{\text{reachable}(a, c)\} \cup T \]
\[ H = \{\text{reachable}(b, c)\} \]
\[ \gamma = \text{reachable}(b, d) \]

Surely, as we have seen previously in Figure 5.1, \( T \cup H \models \gamma \) holds, where \( T = \bigcup_{a_i \in A} T(a_i) \). Nevertheless, \( \exists a_i \ T(a_i) \vdash_{dr} \gamma \) fails to hold. This is because the hypothesis \( \text{reachable}(b, c) \) is not a collaborative inductive resolvent of either \( T(a_1) \) or \( T(a_2) \), hence not separably inducible. It can be induced only based on \( T(a_1) \) and \( T(a_2) \) combined. Thus, this query can not be answered based on an approach which does not allow theories to be combined. As a result, Equation (6.3) no longer holds in this case. On the other hand, if we swap \( \text{reachable}(c, d) \) and \( \text{reachable}(a, c) \) in this example, \( H \) becomes separably inducible and Equation (6.3) would in fact hold.

In summary, the above results have proven that whenever global hypothesis is separably inducible, every instance of the CollabLP problem can be answered using the collaborative deductive-inductive resolution strategy. In other cases, deductive-inductive resolution does not guarantee that every CollabLP problem can be answered. This is largely due to the intrinsic difficulty that some global hypotheses are not inducible without the theories being combined.

In other words, deductive-inductive resolution is not complete in general, unless either the global hypothesis is separably inducible or we are willing to relax the condition and allow centralization of knowledge to some extent.
Part III

Practice
Application: Distributed Path Planning

An important aspect common to collaborative problem solving is the distribution of knowledge. Consequently, solution strategies must necessarily involve either centralization or interaction during problem solving. CollabLP assumes the latter approach—it avoids centralization of knowledge and prohibits unrestricted communication during interaction.

Apart from knowledge being distributed, another important aspect of these problems is knowledge being incomplete in some, often minor, way. To overcome this, as we have previously seen, induction can be gainfully exploited. In fact, induction can be used not only for coming up with hypotheses when knowledge is incomplete, but also for pinpointing the key pieces of knowledge which require communication.

In this and the following chapters, two applications of DIR will be described for solving instances of CollabLP problems. The problems investigated in these two chapters are (i) the distributed path planning problem and (ii) the collaborative network fault diagnosis problem. These two problems are also used to empirically evaluate the DIR approach, when compared against existing approaches that in-
volving centralization or separate instances of induction.

We first compare the DIR solution to CollabLP against an approach that requires centralization (in Chapter 7 Distributed Path Planning). In the second application, DIR is compared with distributed approaches that do not involve induction (in Chapter 8 Network Fault Diagnosis).

### 7.1 Overview

The idea behind the DIR solution, detailed in the previous chapters, is to equip agents with deductive, inductive and interactive capabilities—as well as rules that agents may use to switch between these actions when undertaking a collaborative endeavor searching for the global solution. In this chapter, the distributed path planning problem is used to demonstrate the DIR approach and show empirically the advantages of the approach in reducing communication costs when compared with a centralized approach. The Distributed Path Planning problem was first introduced in Section 4.7.

Each aspect of the application is explained in terms of checking reachability, hypothesizing paths, collaborative path planning and communication strategies. The experimental results are then presented.

### 7.2 Deductive Capability: Checking Reachability

The term \( \text{reachable}(a, b) \) stands for ‘\( b \) is reachable from \( a \)’. The term \( \text{link}(a, b) \) stands for ‘there exists a link from \( a \) to \( b \)’. As in logic programming convention, capital letters are used to denote free variables and lower-case letters bound variables. Both terms can also include extra arguments containing information about the relation (such as cost, e.g. \( \text{link}(a, b, 5) \)) but for simplicity we illustrate using the two-argument form.

We assume each car is equipped with the following as its theory:
7.3. INDUCTIVE CAPABILITY: HYPOTHESIZING A PATH

\[ T_1 : \text{reachable}(A, B) \leftarrow \text{link}(A, B) \]

\[ T_2 : \text{reachable}(A, B) \leftarrow \text{reachable}(A, X) \land \text{reachable}(X, B) \]

\[ T_3 : \text{a set of ground terms in the general form} \text{link}(A, B), \text{e.g.} \text{link}(a, b) \]

\[ T_1 \text{ and } T_2 \text{ are declared instances of inferential relation } D_{dd} \text{ in the DIR framework, } \Sigma(T) \equiv \Sigma(\Sigma(T)) \text{ (refer to Section 5.2). The first rule simply captures the meaning that if there exists a link from } A \text{ to } B, \text{ then it is reachable from } A \text{ to } B. \]

The second rule specifies the transitivity nature of the reachability relation such that if it is reachable from \( A \) to \( X \) and it is reachable from \( X \) to \( B \), then it is reachable from \( A \) to \( B \). Each car also keeps a history of links it has traversed, in the form of \( \text{link}(A, B) \).

Equipped with this combined theory, cars perform deductive reasoning to infer, given any query in the form \( \leftarrow \text{reachable}(A, B) \), whether one location is reachable from another. For simplicity, we will drop the arrow (\( \leftarrow \)) in front of a query when it is obvious from the context that we are referring to a query.

7.3 Inductive Capability: Hypothesizing a Path

On the other hand, in the cases when \( \text{reachable}(A, B) \) can not be answered deductively, i.e. \( B \) is not known to be reachable from \( A \), an inductive procedure may be executed to formulate a hypothesis that allows the deduction to proceed. The following inductive rule captures this inductive capability.

\[ I_1 : \text{reachable}(A, B) \leftarrow \text{induce}(\text{reachable}/2) \land \text{reachable}(A, B) \]

This rule is a declared instances of inferential relation \( D_{id} \) in the DIR framework, \( \Sigma(T) \equiv \Sigma(\Pi(T)) \) (refer to Section 5.2). This rule enables hypotheses about the \( \text{reachable}/2 \) to be induced.

Using the inverse resolution technique with the theory \( T = \bigcup T_n \) as input, the inductive process constructs hypotheses, \( H \), when given a query \( \gamma \) in the form of...
CHAPTER 7. APPLICATION: DISTRIBUTED PATH PLANNING

Figure 7.1: The complete inductive process based on inverse resolution for formulating hypothesis given query \( \gamma = \text{reachable}(a,l) \) for the collaborative path planning example in Figure 4.1. The theory used is shown along the left branches \((T_1, \ldots, T_n)\) and intermediate results generated by the inductive process are shown on the right, including the final hypothesis arrived, \( H = \text{reachable}(a,g) \).

reachable\((A,B)\), such that \( T \land H \models \gamma \). As a simple illustration, imagine that car C from Figure 4.1 tries to find a path from \( a \) to \( l \).

Figure 7.1 shows the complete inductive process, using inverse resolution (IR), of generating hypotheses to explain the query \( \gamma = \text{reachable}(a,l) \). The hypothesis \( H = \text{reachable}(a,g) \) in the example is interpreted as: given what is already known (the background theory), it is reachable from \( a \) to \( l \) (the query) provided it is reachable from \( a \) to \( g \) (the hypothesis).

Notice that the inductive process may be utilized to find a path from one location to another transitively, since if a path does exist the induction process will eventually generate a hypothesis which evaluates to \( \top \) (i.e. a clause containing \( \{p, \neg p\} \) at the same time). Consequently, the minimum subset of \( T_3 \) which generates \( \top \), forms the actual path from \( A \) to \( B \).

Clearly no hypothesis generated by car agent C, in this example, would indicate a full path because the reachability from \( a \) to \( g \) is not known to car C. In spite
of this, hypothesis $H = \text{reachable}(a, g)$ is an important message that engages collaboration (so long as it can be effectively communicated). It will be shown shortly how this partial solution can be used during future endeavors to uncover the full path.

In general, such as in this example, many alternative hypotheses exist and can be generated by the inductive process. For our purpose, the best hypothesis is chosen based on its syntactic simplicity as well as the amount of information used in the process. The algorithm for systematically generating the hypotheses is detailed in Algorithm 3 including methods for selecting promising hypotheses using variations of the minimum description length (MDL) metric. This selection process is termed ‘scoring’.

In practice, inductively generating hypotheses in an uninformed way described above can make search space intractable very quickly. For this reason, Dijkstra’s algorithm has been employed as a heuristic for identifying promising hypotheses and pruning away search space in a mindful way.

### 7.4 Interactive Capability: Collaborative Path Planning

So far it has been demonstrated how induction allows an agent to not only discover a path if it exists but also guess a hypothetical path which can be pursued further. We now turn to the interactive aspects which enable multiple agents with distributed knowledge to collaboratively discover paths. Unsurprisingly, this requires the agents to carefully maintain their knowledge and systematically exchange information with each other.

When a car tries to induce a hypothesis to prove a query in the form $\text{reachable}(A, B)$, it obtains a hypothesis through induction, which it can not determine the validity of by itself. This hypothesis, similarly, has the form $\text{reachable}(A, B)$. At this point, the agent can involve other agents by passing its hypothesis, as a sub-query,
CHAPTER 7. APPLICATION: DISTRIBUTED PATH PLANNING

Algorithm 3 Algorithm for hypothesis generation utilizing deductive shortest path subroutine and scoring.

\textbf{GENHYPO}(\textit{Query})

1: \textit{HypList} $\leftarrow \{\text{Query}\}, \text{HypHistory} \leftarrow \emptyset$
2: \textbf{while} HypList \neq \emptyset \textbf{do}
3: \hspace{1em} Choose hypothesis \textit{H} from HypList
4: \hspace{1em} \textbf{if} \exists \textit{T} in \textit{H} such that \textsc{Dijkstra}(\textit{T}, \textit{Path}) is true \textbf{then}
5: \hspace{2em} Replace \textit{T} with \textit{Path} and store \textit{H} into HypHistory
6: \hspace{1em} \textbf{else}
7: \hspace{2em} Generate all subsequent hypotheses HypAll based on \textit{H}
8: \hspace{2em} \textbf{if} HypAll = \emptyset \textbf{then}
9: \hspace{3em} HypHistory $\leftarrow \{\textit{H}\} \cup \text{HypHistory}$
10: \hspace{2em} \textbf{else}
11: \hspace{3em} HypList $\leftarrow$ HypAll $\cup$ HypList
12: \hspace{2em} \textbf{end if}
13: \hspace{1em} \textbf{end if}
14: \textbf{end while}
15: \textbf{SCOREHYPO}(HypHistory)
16: \textbf{return} all \textit{H} in HypHistory in the order of score.
7.4. INTERACTIVE CAPABILITY: COLLABORATIVE PATH PLANNING

Table 7.1: Deductive-inductive reasoning and interaction between agent cars in the collaborative path planning example is shown, while collaboratively searching for a path from a to l. The Table includes messages passed and hypotheses induced at each step.

<table>
<thead>
<tr>
<th>Step</th>
<th>Agent</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>$\gamma = \text{reachable}(a,l)$</td>
</tr>
<tr>
<td>2</td>
<td>A ASKS C</td>
<td>$\gamma = \text{reachable}(a,l)$</td>
</tr>
<tr>
<td>3</td>
<td>C INDUCES:</td>
<td>$H = \text{reachable}(a,g)$</td>
</tr>
<tr>
<td>4</td>
<td>C REPLIES:</td>
<td>$H = \text{reachable}(a,g)$</td>
</tr>
</tbody>
</table>
| 5    | A DEDUCES: | $K_1 = K_a(K_c(\text{reachable}(g,l)))$  
|      |       | $K_2 = K_a(\exists i K_i(\text{reachable}(a,g)) \rightarrow K_a(\text{reachable}(a,l)))$ |
| 6    | A     | $\gamma = \text{reachable}(a,g)$ |
| 7    | A ASKS B | $\gamma = \text{reachable}(a,g)$ |
| 8    | B INDUCES: | $H = \text{reachable}(a,c)$ |
| 9    | B REPLIES: | $H = \text{reachable}(a,c)$ |
| 10   | A DEDUCES: | $K_3 = K_a(K_b(\text{reachable}(c,g)))$  
|      |       | $K_4 = K_a(\exists i K_i(\text{reachable}(a,c)) \rightarrow K_a(\text{reachable}(a,g)))$ |
| 11   | A     | $\gamma = \text{reachable}(a,c)$ |
| 12   | A INDUCES: | $H = \top$ — path is found |

Take the example in Figure 4.1 again and assume car A is interested in going from a to l. It can be observed from the graph that one existing path is $a-c-d-g-j-l$. The interaction steps for DIR in this scenario are summarized in Table 7.1, presented in a streamlined way.

Car A starts (step 1) with the query $\gamma = \text{reachable}(a,l)$ which it forwards to car C to answer. Car C performs induction and obtains hypothesis, as shown previously, $H = \text{reachable}(a,g)$. Car C returns $\text{reachable}(a,g)$ back to car A. Car A infers that car C knows $\text{reachable}(g,l)$ (by reasoning about the def-
CHAPTER 7. APPLICATION: DISTRIBUTED PATH PLANNING

Figure 7.2: The interaction and information passing among agent cars, A, B and C, is shown for the collaborative path planning example, when searching for a path from a to l.

inition of reachable) (step 5). Therefore, car A knows that as long as a car knows (or can explain) reachable(a, g), the path can be found. At this stage (step 6), the overall problem has changed. Car A now replaces the old query with \( \gamma = \text{reachable}(a, g) \). Later on, through collaboration with car B in a similar fashion, the cars successfully induce the path from a to g. Since car A remembers that car C knows reachable(g, l), the full path from a to l is thus eventually found. Of course, this requires car A to go back to its knowledge base and retrieve what has been inferred before, and about who knows what. The actual transfer of link information then takes place. The distributed interaction steps among the three cars in this example is shown in Figure 7.2.

The choice of the exact communication strategy is an independent decision,
based on specific implementation assumptions and factors. One particular algo-

rithm that has been implemented can be found in Algorithm 4. Some remarks

concerning the choice of communication strategy is discussed in the following

section.

7.5 Remarks on Communication Strategies

Given that an agent can perform deduction, induction or forward the query to oth-

ers, one natural question arises: what order should actions be taken? Since induc-

tion or interaction is typically only needed if deduction fails, it seems reasonable

that deduction should be attempted first so the question is really: when deduction

fails, should an agent first attempt induction or communicate a sub-query?

From a programming point-of-view, the choice should not matter so long as

the space is completely explored. However, for practical reasons, the action se-

quence depends a lot on the overall objective—and there is often a tradeoff be-

tween communication and computation.

When communication is cheap, communication is generally preferred over in-

duction and it makes sense to query as many agents as possible in order to be able

to compare and choose the most promising hypotheses. In those cases, an agent

should only induce when there is nobody else to ask, or if no sub-queries suc-

ceed. When communication is expensive, induction is preferred over communica-

tion when communication does have an associated cost. In those cases, an agent

should avoid unnecessary communication and should commit to its best hypoth-

esis before communicating sub-queries to other agents to explore the hypothesis

further. When communication is prohibitively expensive, an agent’s knowledge

should be fully explored before moving on to another agent. Under these circum-

stances, communication may happen only between the query initiator and other

agents upon demand.

The communication topology and/or hierarchy also influences possible com-
CHAPTER 7. APPLICATION: DISTRIBUTED PATH PLANNING

Algorithm 4 Deductive-inductive steps for collaborative path planning that combines agent interaction, induction and epistemic reasoning.

CPP($Query, Know$)

1: $Hyp \leftarrow \text{GENHypo}(Query)$ // induce the path by itself
2: if $\text{PATHFOUND}(Hyp)$ then
3: \hspace{1em} $Path \leftarrow \text{RETRIEVEPath}(Know)$
4: \hspace{1em} return $Path$
5: end if
6: for $\forall i \in A$ do
7: \hspace{1em} $Hyps \leftarrow Hyps \cup \text{ASK}(i, Query)$
8: end for
9: if $Hyps = \emptyset$ then
10: \hspace{1em} return $\text{FAIL}$
11: end if
12: if $\text{PATHFOUND}(Hyps)$ then
13: \hspace{1em} $Path \leftarrow \text{RETRIEVEPath}(Know)$
14: \hspace{1em} return $Path$
15: end if
16: while $Hyps \neq \emptyset$ do
17: \hspace{1em} $Query \leftarrow \text{CHOOSEBEST}(Hyps)$ // hypothesis to pursue next
18: \hspace{1em} $Know \leftarrow \text{GENKnow}(Query)$ // generate new knowledge
19: \hspace{1em} $Hyps \leftarrow Hyps - \{Query\}$
20: \hspace{1em} CPP($Query, Know$)
21: end while
22: return $\text{FAIL}$
munication strategies, where communication costs typically depend on the number of hops. For instance, a star communication network assumes an agent can communicate with any other agent in the neighborhood directly, with constant cost. A tree type network may require communication in only one direction (say top-down). A ring type network requires agents to only communicate through neighboring agents etc.

Finally, prior knowledge provides agents with additional information when choosing the communication strategy. Based on what is known, agents can discriminate information from one source against another; or know in advance which agent is more likely to know the answer to a particular query; or to know whom to avoid directing particular queries in the first place. Therefore, more sophisticated strategies may be implemented, in which the choice of communication strategy need not be predetermined, but is decided by the agent on an individual basis or as part of its reasoning process.

In the experiments, since cars come and go (and may not reappear), it is assumed the cost of the communication is high and should be avoided whenever possible. In addition, the implementation assumes interaction happens only between the query initiator and the requested party such that the requested party does not issue further queries. In the experiment, a star communication channel is assumed, i.e. any car can communicate with any other car (within distance) directly, and communication between any two cars has the same cost. No prior knowledge has been assumed either. In spite of this, we do see other possibilities and welcome further investigations involving various alternative strategies.

7.6 Remarks on Alternative Approaches

If the agents’ knowledge is assumed to be complete, the problem of distributed path planning is solvable purely deductively. This can be achieved by combining logic programming with a distributed query answering mechanism, such as using
KQML messages (Finin, Fritzson, McKay, & McEntire, 1994) together with a matchmaker.

Using a KQML-style matchmaker, each car can advertise its ability to solve the reachability problem. In such case, if car A can not solve the query $reachable(d, l)$, it would send a recruit query to the matchmaker who would route it to B and B would in turn, when it could not solve a sub-query $reachable(g, l)$, use the matchmaker to have its sub-query sent to C.

This type of approach works on the assumption that the global theory is complete or $\mathcal{H} = \emptyset$. That is, there indeed exists a path from $a$ to $l$, based on the cars’ total traversed history. However, this type of approach would not (gracefully) extend to the situation where $\mathcal{H} \neq \emptyset$. In distributed systems with missing information, hidden information or failure, we can expect that induction (or abduction) would often be required. The DIR approach, on the other hand, allows assumptions to be made regarding reachability and thus works for $\mathcal{H} \neq \emptyset$ too.

In addition, the KQML-based approach does not necessarily have any knowledge of which queries need answering, so under this approach, each agent might have to generate all possible query forms and advertise them to the matchmaker arbitrarily. DIR, on the other hand, provides the basis for a more principled mechanism for demand-driven distributed query answering, from an epistemic (knowledge-driven) standpoint.

The KQML-based approach also needs to know about the specific form of query a priori, which could possibly lead to computationally intractability, without explicit mechanisms for defining this. The consequence would be less general forms of solutions, or more work for the programmer. In comparison, DIR provides an elegant, integrated way or programming demand-driven (lazy) solution through the ability to induce (or abduce) queries adaptively, as an alternative to communication.

With respect to automated theorem proving techniques, since the KQML-based approach would necessarily require implementation using conceptually dif-
ferent program modules, such as query answering and matchmaking modules, this has the possibility of ruling out the easy utilization of certain theorem proving optimizations. In DIR, one can consider the (potentially bi-directional) theorem proving occurring over the same, joint, proof tree. Consequently this facilitates the incorporation of theorem proving techniques that address tractability, such as the implementation of fragments of logic known to be decidable — such as datalog.

7.7 Experimental Results

Based on the implementation and assumptions described above, experiments were carried out that compared the DIR approach to distributed path planning against a centralized approach—in terms of the total communication cost involved (measured by the number of logical terms transferred). In the centralized approach, collaboration takes the form of transferring information (one clause at a time) from participating agents to the query initiator, before executing a deductive path searching algorithm.

In the experiments, the number of agents, $A$, was varied from 2 to 6 and the size of the graphs utilized, $G$, was varied from 60 to 120. The graphs were generated in such a way that there existed at least one path from a given starting location to a given finish location. The experiments then involved distributing the total graph, $G$, (containing all of the links) randomly over the agents. 100 trials were run for every different value of $A$ and $G$ for both approaches (results are plotted in Figure 7.3).

The Figure on the Top Left shows a comparison of communication costs between two agents using the centralized approach (dotted line), DIR approach (dashed line) and DIR approach with scoring (solid line). Communication cost is plotted as a function of information distribution, measured in terms of entropy. Entropy is measured, $\sum_{i=1}^{n} \frac{|K_i|}{|K_T|} \ln \frac{|K_i|}{|K_T|}$, based on the number of terms that each
Chapter 7. Application: Distributed Path Planning

Figure 7.3: **Top Left:** Communication versus knowledge distribution is shown for the Centralized solution (dotted line), DIR (dashed line) and DIR+Scoring (solid line). **Top Right:** Communication costs (amount of information transferred) using the DIR approach with different numbers of agents (from 2 to 6) for different graph sizes (from 60 to 120). **Bottom Left:** Comparison of communication costs using the DIR approach (solid line) with a centralized approach (dashed line) for different numbers of agents with a graph of 120 links. **Bottom Right:** Communication savings as the information (links) known to each agent increases, with three agents and graphs of different sizes.
agent knows, denoted by $|K_i|$, normalized by the total knowledge $|K_T|$. In general, results reveal that communication costs are higher when knowledge is more evenly distributed, which indicates more knowledge need to be communicated in order to arrive at the solution. The DIR approaches (with or without scoring) almost always outperform the centralized approach where sharing complete knowledge is involved—except in extreme cases where one agent has dominant amount of information, in which case it is actually cheaper to simply transfer all knowledge across.

On the Top Right of Figure 7.3 the chart shows an increase of communication costs associated with the DIR approach as the total number of agents in the system increases, as can be observed from lines gradually moving up as A increases. This is somewhat expected since when more agents are involved, more communication is required for a fixed problem in general. The reason these plots are not monotonically increasing as the G increases is because the graph with 60 links is a significantly harder graph to find a path. As a result, more interaction is required to find the solution. However, in general, communication costs increase as the graph size increases.

The chart at the Bottom Left of the Figure compares communication costs using the DIR approach against the centralized approach, with a graph of 120 links and varying number of agents. It is noticed that communication savings with the DIR approach are significant when the total number of agents is small. As the number of agents increases (along with the entropy of the system) the benefit decreases and eventually the cost using the DIR approach exceeds that of the centralized approach. Similar situations occur when using graphs of other sizes. This can be explained in terms of the participation of agents during the solution of the DIR problem. As the number of agents increases, the link information is more sparsely distributed over the agents such that each agent has less knowledge, which makes it harder to come up with useful hypotheses. Consequently, this leads to much more communication, not less. We thus conclude that the DIR
approach is effective in saving communication cost only when each agent has a large amount of information which can be used to solve a significant part of the problem (but not necessarily the entire part) on its own.

The chart on the Bottom Right explores this aspect, plotted as a relationship between communication costs and partial knowledge represented as ‘links per agent’, $L$. It is found that the DIR approach consistently outperforms the centralized approach uniformly with different numbers of agents and graph sizes when each agent has, roughly speaking, 30 links or more.
Chapter 8

Application: Network Fault Diagnosis

8.1 Overview

In this chapter, we describe yet another application in which the DIR framework can be applied on top of conventional routing methods to enhance fault tolerance and improve communication efficiency. Relevant domains include sensor and wireless networks, where communication costs are frequently a limiting factor, as well as improving throughput of general purpose internetworking applications.

Conventional routing approaches, such as distance vector (DV) and link state (LS) routing, involve nodes periodically exchange routing information with neighboring nodes (Hu, Johnson, & Perrig, 2003; Murthy & Garcia-Luna-Aceves, 1996). Those strategies try to keep at every node a complete picture of the entire network at all times, such that each node knows the best routes to all other nodes. Those strategies that are based on periodic updates have high communication overhead and often assume power consumption is not a concern. In wireless and sensor network applications, this is generally not desirable. Some diagnostic strategies thus
have been proposed to reroute around faults should it become necessary. However, such strategies often adopt a global approach which involves reinvoking the route discovery process for the entire network when a broken link is detected or when some part of the network is not responding (Staddon, Balfanz, & Durfee, 2002).

In contrast to those, the DIR approach has been applied as an enhancement to the existing DV routing method which allows neighboring nodes to collaborate during diagnosis of faults and recovery from them. The diagnostic process based on DIR has the following advantages: (i) diagnostic and rerouting decisions are made locally among a small number of neighboring nodes; (ii) the system reacts and converges faster to topology changes than a global approach; (iii) nodes no longer rely on a periodic updating mechanism to keep themselves informed, thus reducing communication tremendously.

8.2 Deductive-Inductive Diagnostic Procedure

In summary, the DIR diagnostic procedure proceeds as follows. When a new node connects to the network, it queries its neighborhood to know who they are and requests their routing tables. In this way, it figures out the best paths to all other nodes and keeps them in its own routing table. When a fault is detected, nodes collaborate to infer where the source of the fault is, how the fault impacts on their own routing and, possibly, adapt to the new network condition by establishing new routes around the fault. Since node failures can be boiled down to link failures (i.e. the failure of all links connected to that node), this work focuses on link failures as the only type of faults.

To see how the DIR diagnosis approach works in practice, consider the following network shown in Figure 8.1(a), as a weighted graph. According to distance vector (DV) routing, each node keeps a table of how to get to other nodes with the least cost. For example, node $a$ has the table abbreviated in 8.1(b). Encoded
8.2. DEDUCTIVE-INDUCTIVE DIAGNOSTIC PROCEDURE

Figure 8.1: Network Routing Example: The scenario involves a network topology (a); (b) a routing table for node a; (c) the routing table encoded in logic terms; and (d) the routing table viewed as a graph.

using first order logic terms, this table becomes 8.1(c). It can also be viewed as the graph in 8.1(d), where solid lines represent physical links and dotted line represent reachability.

Assume the link between b and c is broken (marked as × in Figure 8.1(a)) and node a is first affected by this when it tries to reach node c and fails, i.e. node a noticed ¬reachable(a, c) \(^1\). Node a reasons (inductively) that, given the local knowledge it uses for routing (based on its routing table), this could only have been caused by either ¬link(a, b) or ¬reachable(b, c). The first hypothesis can be falsified, assuming every node is aware of the nodes it is directly connected to and can examine those links for failure. This leaves ¬reachable(b, c) being the only possible cause. The next thing for a to do is to communicate this latest event in a form that will benefit the decision making of its neighbors, i.e. nodes b and e. Initially, a infers that b will definitely be interested in knowing ¬reachable(b, c). a also understands that it is possible that e is connected to c via itself and therefore be interested in knowing ¬reachable(a, c). Notice that the outgoing messages to b and e are different as a result of the network topology and the reasoning process. The local routing information for each node, from the outset, is shown in Figure 8.2:TOP.

\(^1\)The term unreachable is used to encode negation explicitly in the implementation.
CHAPTER 8. APPLICATION: NETWORK FAULT DIAGNOSIS

When node b receives the message from node a telling it ¬\textit{reachable}(b, c), it infers that ¬\textit{link}(b, c) must be true. b subsequently communicates the new information ¬\textit{reachable}(b, c) to d in case d needs to use this itself to get to c. There is no need, however, to send this information back to a, since a is the node who sends out this very information in the first place.

At the same time, node e receives the message ¬\textit{reachable}(a, c) from a. However, e is not affected by this piece of information as it does not require a to get to c, as it has a direct connection with c. The story does not end here. Instead, e infers that a may be interested to know that \textit{reachable}(e, c) and may use this information to reestablish a connection with c via e. Hence, e communicates \textit{reachable}(e, c) to a. At this time, the news regarding the broken link has propagated and the routing information at node a, b and d have been consequently modified.

When a receives the message \textit{reachable}(e, c) from e, it deduces that it can in fact reach c via e. It reorganizes its routing table (see Figure 8.2:BOTTOM) and passes this latest information, \textit{reachable}(a, c), to b. Node b effectively performs the same reasoning and concludes \textit{reachable}(b, c). It passes this information on to its neighboring node, i.e. d. Node b and d both update their routing tables to reflect the latest information. Eventually, no more new information can be inferred and no more propagation is required. At the end of the scenario, the final routing information at each node is shown at the bottom of Figure 8.2. The reasoning and interaction steps between the agents are summarized in Table 8.1.

During the diagnosis and rerouting processes, nodes a, b and d all have their routing table successfully adapted to the latest network condition. Nodes c and e have not been affected. Node e’s routing table is not modified because there is no need to. Node c’s routing table is not modified because it has not noticed any changes. Due to the broken link, no information has reached it. This leaves node c’s routing table in an outdated status. This seeming flaw actually has advantages in terms of controlling the propagation of messages in the network and further
limiting communication. A message gets sent out only when the sender deems it potentially useful to the receiver. Node $c$ will, however, discover the change at a later stage, as soon as it tries to use its outdated routing table to reach any of $a$, $b$ or $d$. Node $e$ will provide it with an alternative route to $b$ when they start to communicate.
<table>
<thead>
<tr>
<th>STEP</th>
<th>ID</th>
<th>ACTION</th>
<th>RESULT</th>
<th>REASONING</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>Detects $\varphi = \neg \text{reachable}(a,c)$</td>
<td>$H = \neg \text{reachable}(b,c)$ $T(a) \land H \models \varphi$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>$\Pi(T(a))$</td>
<td>$H = \neg \text{reachable}(b,c)$ $T(a) \land H \models \varphi$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>Sends $b$</td>
<td>$\neg \text{reachable}(b,c)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>Sends $e$</td>
<td>$\neg \text{reachable}(a,c)$</td>
<td></td>
</tr>
<tr>
<td>2.1</td>
<td>$b$</td>
<td>Receives $\varphi = \neg \text{reachable}(b,c)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$\Pi(T(b))$</td>
<td>$H = \neg \text{link}(b,c)$ $T(b) \land H \models \varphi$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>Sends $d$</td>
<td>$\neg \text{reachable}(b,c)$</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>$e$</td>
<td>Receives $\varphi = \neg \text{reachable}(a,c)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$e$</td>
<td>$\Sigma(T(e))$</td>
<td>$\psi = \text{reachable}(e,c)$ $T(e) \models \psi$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$e$</td>
<td>Sends $a$</td>
<td>$\text{reachable}(e,c)$</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>$d$</td>
<td>Receives $\varphi = \neg \text{reachable}(b,c)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$d$</td>
<td>$\Sigma(T(d))$</td>
<td>$\psi = \neg \text{reachable}(d,c)$ $\varphi \land T(d) \models \psi$</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td>$a$</td>
<td>Receives $\varphi = \text{reachable}(e,c)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$\Sigma(T(a))$</td>
<td>$\psi = \text{reachable}(a,c)$ $\varphi \land T(a) \models \psi$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>Sends $b$</td>
<td>$\text{reachable}(a,c)$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$b$</td>
<td>Receives $\varphi = \text{reachable}(a,c)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$\Sigma(T(b))$</td>
<td>$\psi = \text{reachable}(b,c)$ $\varphi \land T(b) \models \psi$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>Sends $d$</td>
<td>$\text{reachable}(b,c)$</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>$d$</td>
<td>Receives $\varphi = \text{reachable}(b,c)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$d$</td>
<td>$\Sigma(T(d))$</td>
<td>$\psi = \text{reachable}(d,c)$ $\varphi \land T(d) \models \psi$</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.1: Deductive-inductive reasoning and interaction between agents is shown for the collaborative fault diagnosis scenario, including messages passed and hypotheses induced.
8.3 Knowledge-based Diagnostic Algorithm

Algorithmically, failures are handled as follows:

1. Noticing failure: When a link failure first occurs, it remains undiscovered until traffic going through that link gets interrupted. We assume the existence of some kind of acknowledgement mechanism, such that the sender becomes aware of the interruption. This causes the sender to start the reasoning and diagnosis process by sending messages to its neighbors.

2. Reasoning about failure:
   
   (a) Finding the cause (inductively): when new reachability (or unreachability) information is received or inferred, a node hypothesizes a possible explanation for it and adjusts its routing table by replacing contradicting facts.

   (b) Finding the consequence (deductively): when new reachability (or unreachability) information is received or inferred, a node checks if any other known information in the routing table gets changed as a result of the new information and replaces it accordingly.

   (c) Rerouting (deductively): under certain situations, new route may be inferred during the process to circumvent the failed link.

3. Communicating failure: When new information becomes available, either received as a message or as an outcome of the reasoning process, a node passes the information on to those neighboring nodes which it believes may benefit from the information.

   The agent’s reasoning steps are governed by the algorithm shown in Algorithm 5. When receiving a message \( \varphi \), an agent first checks whether \( \varphi \) is a deductive consequence of its knowledge (line 1). In other words, it checks whether \( \varphi \) is already known. If \( \varphi \) is already entailed by its knowledge, \( \varphi \) does not add any value
to its knowledge and is thus ignored. If not, the agent checks if \( \varphi \) contradicts any existing knowledge (line 3). Thus, it checks whether \( \neg \varphi \) is entailed by the existing knowledge. If that is the case, it is an indication that something unexpected has happened. The agent further checks if \( \varphi \) is an inductive consequence of its existing knowledge (line 5), that is, whether \( \varphi \) can be explained by its existing knowledge and some inductive hypothesis. If so, a fault has been identified and the agent’s knowledge needs to be modified to reflect this.

The algorithm also deals with theory revision. To keep the existing theory (routing tables) in a consistent form, both the deductive and inductive processes may be required to modify existing information in the routing table (line 5 and 8). When generating hypotheses to explain queries in the form of \( \neg \text{reachable}(A, B) \), the inductive process simply removes entries from the routing table until \( \text{reachable}(A, B) \) is no longer entailed. The hypothesis is obtained from the negation of the entry removed. Any other contradicting entries in the routing table are also removed. This revision also happens when a new piece of knowledge is deduced.

The remaining portion of the code deals with sending information to neighboring agents (line 7). When deciding whether a neighboring node needs to be informed of a piece of information (line 9), the agent’s reasoning is governed by the following two rules: (i) my neighbors will be interested in knowing all other nodes to which I become reachable/unreachable; (ii) if my neighbor is unable to reach a node but I am able to, it will be interested in knowing it.

### 8.4 Remarks on Alternative Approaches

If periodic sensing of neighboring nodes is assumed to be feasible—not the case in many wireless network applications—approaches based on truth maintenance (Huhns & Bridgeland, 1991) can be applied to maintain the consistency of topological information among agents and potentially can be used to discover and reroute around faults.
Algorithm 5 Deductive-inductive steps adopted by agents for collaborative fault diagnosis.

1: if $T \models \varphi$ then
2:     ignore $\varphi$.
3: else if $T \models \neg \varphi$ then
4:     if $\exists H \ T \land H \models \varphi$ then
5:         $T \leftarrow T \cup H$
6:     end if
7:     for all $\psi, T \models \psi$ do
8:         $T \leftarrow T \cup \psi$
9:     if $\psi$ is useful to any neighbor $i$ then
10:        send $\psi$ to $i$.
11:    end if
12: end for
13: end if
CHAPTER 8. APPLICATION: NETWORK FAULT DIAGNOSIS

Under maintenance based approaches, nodes may detect failed links based on observation and communicate changes to the neighboring nodes that are still connected to, who would then bring their knowledge bases consistent with each other to reflect the latest changes to the topology of the network.

Truth maintenance based approaches, without major extensions, typically assume a homogeneous set of agents. Thus these type of approaches does not extend gracefully to situation where agents have their personal perspective of the world encoded in their own representations and are not aware about the internal logic of other agents. The drawback with this type of approaches, compared with the DIR diagnostic approach, is it requires adjacent nodes not only to know about each others knowledge of the world and their representations, but also to keep them in consistent states at all time.

In contrast, in the DIR approach, an agent figure out the new status of the network not simply based on direct awareness and constant updating of other agents’ belief, but based on its own (deductive or inductive) reasoning on the status of its prior knowledge with the help of information from other agents.

This allows the agents to be heterogeneous, who infer changes to their knowledge bases and information to be shared on a case-by-case basis through deductive-inductive reasoning.

8.5 Experimental Results

It has been studied empirically the effects on reducing communication costs and transmission errors when extending the well-known distance vector (DV) routing algorithm (Hu et al., 2003), using a diagnostic routine based on DIR. The diagnostic approach is compared with the basic DV algorithm, or \( DV(Basic) \), and two improved versions which are referred to as \( DV(Active) \) and \( DV(On~Demand) \).

The basic DV algorithm requires the nodes to regularly exchange routing tables with their neighbors at some predetermined time interval. The DV(Active)
approach extends DV by having each node proactively notifying their neighbors soon as it detects any topology change. The DV(On Demand) approach improves DV by having each node regularly sense their neighbors but only exchange routing tables with them when topology change is detected.

The experiments involve randomly generated topologies, with random number of nodes (from 10 to 50) at random locations (on a two-dimensional rectangular surface). Faults were introduced by breaking random links (one at a time) at random time instances. The nodes were designed to transfer data with each other at regular intervals, at a higher frequency than the rate faults were introduced. Each different approach was tested over 500 trials for every topology setting and the results shown were averaged.

In summary, results demonstrate that the diagnostic enhancement based on DIR is effective in keeping information loss at a minimal level (and subsequently misses to a minimum), with only a negligible amount of extra communication required and a moderate amount of computational overhead. Results are plotted in Figure 8.3. An exponential decrease in communication is observed as the time interval (for exchanging routing tables) is increased for all approaches (Figure 8.3:TOP). As the time interval is increased, however, information loss also increases—except for the DV(DIR) approach—as a result of outdated routing information (Figure 8.3:MIDDLE). This is because, for the three approaches with greatest information loss—DV(basic), DV(active) and DV(on-demand)—if any network faults happen during the time interval, nodes are unable to recover until the next time they exchange routing tables. As a result, all data transferred in the interim would be lost.

In contrast, information loss remains constant for the DV(DIR) approach regardless of how often nodes exchange routing tables with each other, since as soon as the first transmission miss occurs the affected node starts diagnosis. As a result, the number of transmission misses converges quickly after a few affected nodes

\[^2\text{Complete result is presented in Table A.2.}\]
have been involved and new routes have been established. Eventually, all nodes that are affected have their routing tables updated and transmission resumes as normal. The fact that DV(DIR) is constant in terms of information loss also suggests that the diagnostic approach is able to endure a much larger—possibly even infinite—time interval for table exchanging, without significantly downgrading its performance. Whereas the increase of time interval will cause the other approaches to malfunction significantly.

The fact that DV(DIR) does not intrinsically rely on a periodic exchange of routing tables enables the diagnostic approach to reduce communication to the asymptotic level and is thus actually more communication efficient than any of the periodic approaches.
8.5. EXPERIMENTAL RESULTS

Figure 8.3: **TOP**: Communication measured as number of message decreases exponentially as the time interval lengthens for all four approaches (note that DV(DIR) is coincident with DV(basic) and DV(active) for this plot). **MIDDLE**: Information loss, measured as the number of misses, increases as the time interval increases for DV(basic), DV(active) and DV(on demand). Information loss remains constant for DV(DIR). **BOTTOM**: Computational effort decreases exponentially as Time Interval lengthens. DV(DIR) only requires around 50% extra computation at typical update intervals. Refer to Table A.2 for the actual data.
Chapter 9

Conclusion

9.1 Summaries and Discussions

This thesis has identified and formalized a new class of multi-agent programming problems involving collaboration and learning, the CollabLP problem, and proposed the deductive-inductive resolution (DIR) framework which integrates existing works on deductive and inductive logic programming.

The DIR approach has shown promise even for problems not requiring learning, in which induction may simply be an enhancement over deductive reasoning or a preferable alternative to communication. This is especially relevant when communication is expensive due to energy requirements and delay, such as in sensor network applications, where communication is typically several orders of magnitude more expensive than computation.

By incorporating induction into deductive theorem proving, we inevitably move away from a logic system that has a truth-preserving property. However, this does not simply mean that integrating deductive and inductive reasoning is hopeless. On the contrary, it indicates that this level of uncertainty is something fundamental and has to be dealt with since, after all, logic is the science
CHAPTER 9. CONCLUSION

of reasoning—not necessarily the science of correct reasoning. Moreover, as has been demonstrated through examples that in restricted domains, this integration can be used for solving many practical collaborative problems, where induction yields promising results. It also opens the possibility of discovering new forms of collaborative problem solving algorithms in multi-agent settings.

Applications of the DIR framework include, but not limited to (i) logic based inductive learning problems in distributed settings; (ii) collaborative execution of logic programs based on distributed knowledge bases; (iii) high-level multi-agent planning tasks in the same vein as (Missiaen et al., 1995; do Lago Pereira & de Barros, 2004b).

9.2 Contributions

Specifically, the thesis has contributed the following technical results:

- A formal definition of the collaborative logic programming (CollabLP) problem, which captures not only problems involving learning in multi-agent environments, but also deductive problems requiring collaboration in general;

- A new, bidirectional deductive-inductive resolution (DIR) approach for solving instances of the CollabLP problem;

- A modal treatment of the DIR approach, based on which DIR is proven to be sound and complete (under the separably inducible assumption) with respect to solving the CollabLP problem, and

- Experimental evaluations of two applications that both illustrate solutions to instances of the CollabLP problem and empirically demonstrate the advantages of the DIR approach, such as for avoiding centralization, reducing inter-agent communication and enhancing routing accuracy.
9.3 Directions for Further Research

A number of promising further research directions are foreseeable. Among them, the computational complexity implications of the recursive application of deductive-inductive inference from an omniscient point-of-view is worthy of further investigation. For instance, so far it has been proven, in the completeness theorem, that the solution to a CollabLP problem can always be found through the collaborative endeavor based on DIR as long as the global hypothesis is separably inducible. But it remains unanswered how hard it is to find such a solution in general. After all, logic-based approaches are often criticized for its intractability. On the other hand, the amelioration of complexity due to the collaborative aspects of the DIR approach, from the perspective of each agent through problem decomposition, is also worth investigating.

The DIR framework, in its current form, allows further extensions in various possible ways. For example, it may be practical in some applications to impose a hierarchical structure to make the approach more efficient, in the same vein as (Nieuwenborgh et al., 2007). Epistemic-level knowledge management techniques may also prove beneficial, relative to a particular domain, to avoid unguided application of deductive-inductive inferences.

Currently, the definition of the CollabLP problem assumes agents’ theories are consistent. Although this is highly ideal in complex domains, existing works in belief change (Kern-Isberner, 2001; Delgrande, Lang, Rott, & Tallon, 2005; Bonanno, Delgrande, Lang, & Rott, 2007) or non-monotonic reasoning (Brogi, Lamma, Mancarella, & Mello, 1997; Sakama, 2001) can be incorporated for reasoning based on inconsistent theories.

So far, systems have been developed based on DIR for specific applications of CollabLP problems. It would be interesting to see a general purpose theorem prover constructed based on the five elementary inferential relations imposed by DIR. The major obstacle that forbids us from achieving this goal is the lack of a
general purpose ILP system. Unlike Prolog, ILP systems in their current status are not powerful enough to take on this task for being an inductive engine in general. In spite of this, with some fine-tuning, ILP systems such as Aleph (Srinivasan, 2001) or even a manual implementation of the inverse resolution algorithm are sufficient for simple abductive reasoning tasks in various specific applications.
Appendix A

Supplementary Experimental Details

A.1 Overview

Experimental data and output, which have been left out in the main body of the thesis, is provided as a supplement in this chapter. Interested readers are encouraged to contact the author to obtain a copy of the source code used in collecting and producing the results shown here.

A.2 Output of the Sorting Example

Presented in Table A.1 is the system output after executing the sample query \( deduce(i, \min([3, 2, 4, 1, 5], M)) \) described in the sorting example in Section 4.6. The multi-agent collaborative logic programming system is implemented by integrating SWI-Prolog and the ILP system Aleph (Srinivasan, 2004), in which distributed programs (even with some missing fragments) can be executed without being congregated.

In the sample query in this example, we have specified that agent \( i \) is asked to
deduce $\min([3, 2, 4, 1, 5], M)$. Agent $i$ keeps replacing goals until the point where it fails to find a definition for $sort$. It first attempts to induce the definition for it based on its own background knowledge, but fails. It then turns to agent $j$ for the definition, who successfully induces it and gets back to agent $i$. Agent $i$ then proceeds with the rest of the goals. Note here that since agent $i$ has acquired the knowledge that agent $j$ knows the definition for $sort$, it directs the current goal to agent $j$ to deduce. Surely enough, agent $j$ has the definition and returns success for the goal $sort([3, 2, 4, 1, 5], G62)$. After having the result back, agent $i$ finally finishes replacing all goals in the list and successfully find out that $M = 1$.

A.3 Data on Network Fault Diagnosis

The experiment data presented in Table A.2 are collected on a DELL OPTIPLEX GX270 PC with 2 Intel Core CPU @2.4GHz each and 2GB of RAM, in Windows XP SP3 environment.

The data shown in this table are used for plotting Figure 8.3, which compares the diagnostic approach based on DIR against three other competitive approaches based on distance vector routing for a selected network fault diagnosis scenario. Each data entry in the table represents an average of 500 trials using the same diagnostic scenario, comparing the communication costs, number of misses as well as the computational costs (measured in milliseconds realtime).

The system for experimenting collaborative network fault diagnosis is implemented in Java and has a front-end GUI as shown in Figure A.1. The program may be run either in interactive mode, in which nodes can be inserted and moved manually to form a network, or in batch mode, in which predefined network topologies can be loaded from external files. The back-end implements four different routing and diagnostic approaches, based on variations of the distance vector routing algorithm as described in Section 8.5.
A.3. DATA ON NETWORK FAULT DIAGNOSIS

?- deduce(i, min([3, 2, 4, 1, 5], M)).
---------------- Deduction Attempt ----------------
Agent: i
Goal: min([3, 2, 4, 1, 5], _G298)
---------------------------------------------------
Current Goals: [sort([3, 2, 4, 1, 5], _G975), first(_G975, _G298)]
---------------- Induction Attempt ----------------
Agent: i
Predicate: sort
Background: [last(_G1327, _G1328), first(_G1321, _G1322), min(_G1315, _G1316)]
---------------------------------------------------
---------------- Induction Attempt ----------------
Agent: j
Predicate: sort
Background: [ordered(_G1646), permutation(_G1640, _G1641)]
---------------------------------------------------
---------------- Induction Success ----------------
Predicate: sort
---------------------------------------------------
Current Goals: [sort([3, 2, 4, 1, 5], _G62), first(_G62, _G45)]
---------------- Deduction Attempt ----------------
Agent: j
Goal: sort([3, 2, 4, 1, 5], _G62)
---------------------------------------------------
Current Goals: [permutation([3, 2, 4, 1, 5], [_G1549|_G1550]), ordered([_G1549|_G1550])]
Current Goals: [select(_G1549, [3, 2, 4, 1, 5], _G1596), permutation(_G1596, _G1550), ordered([_G1549|_G1550])] Current Goals: [permutation([2, 4, 1, 5], _G1550), ordered([3|_G1550])]
Current Goals: [select(_G1633, [2, 4, 1, 5], _G1638), permutation(_G1638, _G1634), ordered([3, _G1633|_G1634])] Current Goals: [permutation([4, 1, 5], _G1634), ordered([3, _G1633|_G1634])] ...
Current Goals: [select(_G1855, [5], _G1860), permutation(_G1860, _G1856), ordered([1, 2, 3, 4, _G1855|_G1856])]
Current Goals: [permutation([], _G1856), ordered([1, 2, 3, 4, _G1855|_G1856])] Current Goals: [ordered([1, 2, 3, 4, _G1855|_G1856])] Current Goals: [1=<2, ordered([2, 3, 4, 5])] Current Goals: [ordered([2, 3, 4, 5])] Current Goals: [2=<3, ordered([3, 4, 5])] Current Goals: [ordered([3, 4, 5])] Current Goals: [3=<4, ordered([4, 5])] Current Goals: [ordered([4, 5])] Current Goals: [4=<5, ordered([5])] Current Goals: [ordered([5])] Current Goals: []
---------------- Deduction Success ----------------
Current Goals: [first([1, 2, 3, 4, 5], _G45)]
Current Goals: []
---------------- Deduction Success ----------------
X = 1

Table A.1: Sample output of the sorting example in Section 4.6 when given the query min([3,2,4,1,5],M) to agent i. In the simulated multi-agent programming environment, implemented based on Prolog and Aleph, when given the query to any agent in the team, the agents collaboratively execute the program and work out the missing fragment when necessary to answer the query. The output has been chopped to fix in a page.
<table>
<thead>
<tr>
<th>Interval</th>
<th>DV(basic)</th>
<th></th>
<th></th>
<th>DV(active)</th>
<th></th>
<th></th>
<th>DV(on demand)</th>
<th></th>
<th></th>
<th>DV(DIR)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21393.0</td>
<td>73.0</td>
<td>21.84</td>
<td>21557.0</td>
<td>52.0</td>
<td>18.79</td>
<td>3428.0</td>
<td>22.0</td>
<td>8.88</td>
<td>21513.7</td>
<td>16.6</td>
</tr>
<tr>
<td>2</td>
<td>10593.0</td>
<td>146.0</td>
<td>11.21</td>
<td>10757.0</td>
<td>104.0</td>
<td>9.65</td>
<td>2078.0</td>
<td>44.0</td>
<td>4.93</td>
<td>10717.9</td>
<td>16.9</td>
</tr>
<tr>
<td>3</td>
<td>6993.0</td>
<td>219.0</td>
<td>7.61</td>
<td>7157.0</td>
<td>156.0</td>
<td>6.62</td>
<td>1628.0</td>
<td>66.0</td>
<td>3.55</td>
<td>7114.0</td>
<td>16.6</td>
</tr>
<tr>
<td>4</td>
<td>5193.0</td>
<td>292.0</td>
<td>5.62</td>
<td>5357.0</td>
<td>208.0</td>
<td>4.87</td>
<td>1403.0</td>
<td>88.0</td>
<td>2.89</td>
<td>5314.6</td>
<td>16.8</td>
</tr>
<tr>
<td>5</td>
<td>4113.0</td>
<td>365.0</td>
<td>4.49</td>
<td>4277.0</td>
<td>260.0</td>
<td>3.80</td>
<td>1268.0</td>
<td>110.0</td>
<td>2.54</td>
<td>4235.4</td>
<td>16.8</td>
</tr>
<tr>
<td>6</td>
<td>3393.0</td>
<td>438.0</td>
<td>3.61</td>
<td>3557.0</td>
<td>312.0</td>
<td>3.34</td>
<td>1178.0</td>
<td>132.0</td>
<td>2.13</td>
<td>3514.9</td>
<td>16.7</td>
</tr>
<tr>
<td>7</td>
<td>2889.0</td>
<td>511.0</td>
<td>3.28</td>
<td>3053.0</td>
<td>364.0</td>
<td>2.84</td>
<td>1115.0</td>
<td>154.0</td>
<td>2.21</td>
<td>3010.4</td>
<td>16.7</td>
</tr>
<tr>
<td>8</td>
<td>2529.0</td>
<td>584.0</td>
<td>2.82</td>
<td>2693.0</td>
<td>416.0</td>
<td>2.74</td>
<td>1070.0</td>
<td>176.0</td>
<td>1.95</td>
<td>2651.1</td>
<td>16.8</td>
</tr>
<tr>
<td>9</td>
<td>2241.0</td>
<td>657.0</td>
<td>2.54</td>
<td>2405.0</td>
<td>468.0</td>
<td>2.19</td>
<td>1034.0</td>
<td>198.0</td>
<td>1.72</td>
<td>2361.3</td>
<td>16.6</td>
</tr>
<tr>
<td>10</td>
<td>1953.0</td>
<td>730.0</td>
<td>2.15</td>
<td>2117.0</td>
<td>520.0</td>
<td>1.94</td>
<td>998.0</td>
<td>220.0</td>
<td>1.77</td>
<td>2073.4</td>
<td>16.6</td>
</tr>
<tr>
<td>11</td>
<td>1809.0</td>
<td>803.0</td>
<td>2.10</td>
<td>1973.0</td>
<td>572.0</td>
<td>1.84</td>
<td>980.0</td>
<td>242.0</td>
<td>1.67</td>
<td>1933.5</td>
<td>16.9</td>
</tr>
<tr>
<td>12</td>
<td>1593.0</td>
<td>876.0</td>
<td>1.82</td>
<td>1757.0</td>
<td>624.0</td>
<td>1.65</td>
<td>953.0</td>
<td>264.0</td>
<td>1.54</td>
<td>1714.4</td>
<td>16.7</td>
</tr>
<tr>
<td>13</td>
<td>1521.0</td>
<td>936.0</td>
<td>1.72</td>
<td>1685.0</td>
<td>676.0</td>
<td>1.58</td>
<td>944.0</td>
<td>286.0</td>
<td>1.55</td>
<td>1637.9</td>
<td>16.3</td>
</tr>
<tr>
<td>14</td>
<td>1377.0</td>
<td>1008.0</td>
<td>1.56</td>
<td>1541.0</td>
<td>728.0</td>
<td>1.46</td>
<td>926.0</td>
<td>308.0</td>
<td>1.54</td>
<td>1496.2</td>
<td>16.5</td>
</tr>
<tr>
<td>15</td>
<td>1233.0</td>
<td>1080.0</td>
<td>1.44</td>
<td>1397.0</td>
<td>780.0</td>
<td>1.33</td>
<td>908.0</td>
<td>330.0</td>
<td>1.61</td>
<td>1353.1</td>
<td>16.5</td>
</tr>
</tbody>
</table>

Table A.2: Experimental data collected for collaborative fault diagnosis, showing the comparison of the diagnostic approach based on DIR against three other competitive approaches in terms of the communication costs, number of misses as well as the computational costs. Each data entry represents an average of 500 trials. These data are plotted in Figure 8.3.
Figure A.1: Software interface of the program created for experimenting collaborative fault diagnosis. The program interface is created using Java® (Swing package). The program may be run in either interactive or batch mode.
Bibliography


Bibliography


Bibliography


Bibliography


Bibliography


Index

abductive logic programming, 3, 44–46
answer set programming, 2, 40–42
CollabLP, see collaborative logic programming
collaborative logic programming, 3–4, 42, 46, 49–52, 54–55, 58, 78–81, 83, 90–95, 99, 115, 129, 131
hypothesis, 50, 54, 55
inducibility, 54, 81
message, 51, 55
query, 50, 51, 55
separably inducible, 80, 94, 95
theory, 50, 51, 55
DCSP, see distributed constraint satisfaction problem
deductive reasoning, 4, 14–16, 34, 36, 130
deductive-inductive resolution, 3–6, 37, 42, 46, 61–64, 71, 78–81, 83, 90, 91, 100–103, 116–119, 129, 131
collaborative deductive resolvent, 76
collaborative deductive-inductive inference, 76–77
collaborative inductive resolvent, 77
collaborative inductive inference, 65, 66
deductive inference, 65, 66
deductive resolvent, 71–73
deductive-inductive inference, 5, 64–66, 68, 71–73, 86
inductive inference, 65, 67
inductive resolvent, 71
inferential relation, 5, 64–66, 73, 74, 101, 132
soundness, 6, 80, 83, 90–91
DIR, see deductive-inductive resolution
distributed constraint satisfaction problem, 27, 39
distributed path planning, 7, 57–59, 99
DV, see network fault diagnosis, distance vector routing
epistemic logic, 23–27, 131
axiomatization, 24, 85
common knowledge, 26
distributed knowledge, 26
group knowledge, 26
knowledge axiom, 25
logical omniscience problem, 26
negative introspection, 25
positive introspection, 25

ILP, see inductive logic programming

ILP implementation
Aleph, 20, 132, 133
Cigol, 20
FOIL, 20
Golem, 20
Progol, 20

ILP technique
generalization, 19
inverse entailment, 20
inverse resolution, 20–22, 75, 132
least general generalization, 20, 22
specialization, 19
v-operator, 21–22, 62
w-operator, 21–22

inductive logic programming, 3, 18–22, 36, 44–46, 61, 70, 132
ture
LGG, see ILP technique, least general generalization

logic programming, 16–17, 36, 40, 61
definite clause, 50
goal clause, 50
Horn clause, 16, 50
Prolog, 16–17, 70, 133
resolution, 16, 20, 61
SLD-resolution, 16–17, 92
completeness, 16, 17
soundness, 16, 17
undecidable, 16
unification, 17

modal logic, 23, see epistemic logic
multi-agent learning, 1–2, 28–30, 43–44
division learning, 29
interactive learning, 29, 43–44
isolated learning, 1, 28, 43
multiplication learning, 29
multi-agent system, 27–28
awareness classification model, 28
collaborative problem solving, 2, 37–42, 58
distributed computing, 38
distributed problem solving, 38

network fault diagnosis, 7, 115, 134
distance vector routing, 116, 134
link state routing, 116

Kripke structure, see possible-world struct-
possible-world structure, 6, 23–24, 83
  reflexive, 26
  symmetrical, 26
  transitive, 26
RichProlog, 2, 36, 69–71
  133
universe structure, 84–87, 93, 94
  compression, 87–89, 91