DIVIDING AND CONQUERING LOGIC

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Eyal Amir
July 2002

© Copyright by Eyal Amir 2002
All Rights Reserved

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

John McCarthy
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Michael R. Genesereth

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Sheila A. McIlraith
(Knowledge Systems Lab, Stanford University)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Leora Morgenstern
(IBM T.J.Watson)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Nils J. Nilsson

Approved for the University Committee on Graduate Studies:

Abstract

This thesis provides principles for knowledge representation and reasoning with logical theories that are composed from fragments. It provides algorithms for reasoning with logical theories that are made of multiple partitions that may overlap in contents. Algorithms for automatic decomposition of given theories to partitions are proposed. Some of these algorithms are shown to approximate the optimum by a small constant factor. An architecture for an intelligent agent that uses similar algorithms and follows Brooks' subsumption architecture using logic is examined. Experiments applying this architecture for the task of mobile-robot control are reported. Finally, following the object-oriented design methodology, tools and methodology for the design of decomposed theories are provided. The tools and methodology are applied for the design of logical theories describing the domain of robot motion and interaction.

Acknowledgements

In writing this thesis I was helped and influenced by many colleagues and friends. I would like to take the space here and thank them all for their support and stimulation, for the good times we spent together and for being who they are. I tried to remember everyone, but inevitably there will be some whom I did not list. Please accept my apologies. My love and appreciation is yours nonetheless. Even those that I have listed here could not get proper attention because of lack of space and the public nature of this manuscript. Be sure that my gratitude goes way beyond the words I have assembled here in your honor.

First and foremost, I wish to thank my advisor, John McCarthy, for giving some of his inspiration and excitement, time, kindness and friendliness, jokes, stories, patience and experience, for sharing his thoughts and dreams and for bringing me to Stanford. A Ph.D. under his advice is the experience of a lifetime, so there is no chance I'll forget any of it. I also am indebted to him for putting his trust in someone he barely knew. Keep on dreaming, John.

I also wish to thank Pedrito Maynard-Reid II for being my first coauthor and one of my best friends. Our research and life-focused interactions were some of the most fruitful and joyful that I've had in my studies. It is funny how two people from such different backgrounds and environments can grow up to be so similar. Some of the work that we have done together appears in this volume (especially Chapter 6). Good luck with the new addition to the family, Pedrito!

Sheila McIlraith has been an important collaborator, supporter and friend. She has spent much effort and time on our joint research, always being positive and cheerful and having an open ear. I will always be grateful for our discussions and the experiences we had working on our research together. Some of the work that we have done together appears in this volume (especially Chapter 4). Thank you, Sheila.

I was fortunate to have Nils Nilsson as a faculty member and friend in the department. He is one of the most supportive professors at Stanford, always patient and good-spirited. He always gives you only constructive criticism, is always excited about your work, and is ready to listen, give

advice, push you to achieve higher standards and help you in time of need. Thanks, Nils, for being smart and perceptive, supportive and patient. Don't retire (though you already have)!

I am also grateful to Mike Genesereth for being a faculty friend in the department and playing an important role in my studies. He has been my sounding board and supporter for some years already. We had long discussions on what directions I should take and what merit various advances would have. I thank you, Mike, for always encouraging me to strive for more, always giving constructive criticism, and simply for being there.

Last, but not least, from my reading committee is Leora Morgenstern. I valued our discussions about research and life and her excitement about research and AI. Her ability to provide me with perspective and give constructive criticism has been invaluable for me. Besides being a good friend, she has always been a consultant, supporter and confidante. Good luck in the rest of the way, Leora!

The experience of writing this thesis and going through the Ph.D. program would have been much less joyous without my student friends in the *AI reading group* (sometimes called *Quail*, *student support group* and *TSG*). Lise Getoor, Patrick Doyle, Urszula Chajewska, Karl Pflegger, Avi Pfeffer, Pedrito Maynard-Reid II and Sunil Vemuri are my friends, support group, reviewing board and everything else that you may imagine. My years at Stanford would have been so diminished if they had not included you!

In earlier stages of my studies I met Berthe Choueiry and Josefina Sierra, then researchers in the department and now professors at the University of Nebraska, Lincoln and Universidad Autonoma de Madrid, respectively. I will never forget their help and encouragement (each separately and in separate times) through the most difficult times I had while doing research on *Elaboration Tolerance*.

I also wish to thank the members of my research group, the *Formal Reasoning Group*: Tom Costello, Sasa Buvac and Aarati Parmar. They have commented on my ideas and helped me improve and advance many of them. In particular, I thank Aarati Parmar for help in discussing my work on object-oriented situation calculus, reported in this thesis.

Another research group that served as a second home for me in various stages of my studies was The Nobots group (Daphne Koller's, Yoav Shoham's and Nils Nilsson's research groups). The people in this group have given me tons of good advice and corrections, especially during the difficult first few years. I am very grateful to them for welcoming me in and for all of the attention and help.

During my studies I have made use of some of the resources at the robotics lab (computers, robots, etc.). In this context, I wish to thank Yoav Shoham, who funded my presence on the robotics

lab computers, and Jean-Claude Latombe and Nils Nilsson who allowed Pedrito and me to use their Nomad 200 robots. I also want to thank Héctor González-Baños for a lot of help and advice on using (and occasionally fixing) the robots.

I have had many discussions and received support from many other people and groups throughout my studies. In particular, I wish to thank Rada Chirkova, for many fun discussions and supporting words; Jan Wiebe and the DC (Doctoral Consortium) of 1999, for being such a great group of supporters; Sarit Kraus, for giving the right word at the right time; and all those who lent their support over the years: Raymond Reiter, Erik Sandewall, Alon Levy, Fausto Giunchiglia, Shlomit Pinter, Mary-Anne Williams, Jeremy Frank and Vladimir Lifschitz.

Special thanks go to Mark Stickel for allowing me to use his PTTP sources (both for PROLOG and LISP) and for providing helpful answers to my inquiries regarding its use. He has also discussed different developments in my thesis as it drew close to completion.

I cannot express enough gratitude to Erika Henik, my girlfriend, for sticking with me through all the good and bad times, for teaching me how to write better and for constantly editing my works. The love that we shared and her confidence were priceless for me. I discovered and was enormously enriched by our life together. My love is yours, Erika.

I also thank my parents, Avraham and Adi Amir, my sister, Tal (Liat), and my brothers, Shay and Guy, whom I've missed so so much during my studies here. To my brother, Shay, I will never forget the half-year you spent with me (just wait until I come and spend half a year in your bedroom...).

Noa, Avital, Michal, Miriam and Luis Shein and Hadasa and Ralph Kramer have been like a second family to me here. Your love and welcoming contributed to my sanity in this speed-of-light place.

I also want to mention all the friends that I miss in Israel who did not let me go: Camila Edry, Orna Nomberg, Amit (Iris) Yalon, Olga Melayev, Michal Fulman, Esti and Nir Goldberg, Robert Krauthgamer, Amit Reisman and Inbar Levin, Yair Bartal and Gal Barkan. They were all great supportive friends during the difficult moments. Before coming here I thought I might lose most of you guys, but the distance did not deter you.

To the friends I made here, Aviram Gutman and Limor Zer-Gutman, Adela Ben-Yakar, Mor Armony, Michal and Doron Gal, Eran and Tamar Liron, Assaf Zeevi and Sharon Peled, Yochai Konig and Orna Tarshish, Ehud Isacoff, Natasha Noy, and many, many others, it was good to have spent such a good time with you.

Finally, I wish to thank Stanford University for such a great experience and being a great place.

I am sad to leave. Also, throughout my Ph.D. studies I was supported by grants given to my advisor and quite a few travel fellowships. I wish to acknowledge the support of ARPA (ONR) grant N00014-94-1-0775 and Air Force grant F49620-97-1-0207 and many AAAI and IJCAI travel scholarships (1996,1997,1998,1999,2000,2001) (the travel fellowship of 1999 was provided by a joint award from AAAI and ACM-SIGART for the Doctoral Consortium).

Contents

Abstract **v**

Acknowledgements **vii**

1 Introduction **1**

1.1 Representation, Reasoning and Control with Logic 1

1.1.1 Overview 1

1.1.2 What Is the Problem? 2

1.1.3 What Are the Contributions of This Thesis? 3

1.1.4 Why Are These Results Important for Computer Science? 4

1.2 The Technical Results of This Thesis 6

1.2.1 Object-Oriented Design of Logic-Based Commonsense Theories 6

1.2.2 Automated Reasoning With Loosely-Related Theories 8

1.2.3 Automated Decomposition of Logical Theories 9

1.2.4 An Agent Control Architecture Based on Automated Theorem Provers . . . 10

1.3 Logic Needed to Read This Thesis 11

1.4 Plan of This Thesis 13

2 Object-Oriented Design in Logic **15**

2.1 Introduction 15

2.2 The Problem of Building and Reusing Logical Theories 16

2.3 An Object-Oriented Approach to Logic 18

2.4 Object-Oriented First-Order Logic 20

2.4.1 Language 21

2.4.2 Example Theory: Model-Based Reasoning 21

2.4.3 First-Order Semantics . . . 23
2.4.4 Inheritance . . . 24
2.4.5 Reasoning Algorithms for OOFOL . . . 24
2.5 Directional Influence Between Objects . . . 26
2.5.1 Example Applications: Hierarchical Theories and Action Theories . . . 26
2.5.2 Directional Semantics . . . 28
2.5.3 Reasoning Algorithms for Directional OOFOL . . . 30
2.6 Related Work: Object-Oriented KR Systems . . . 31
2.7 Summary and Discussion . . . 34

3 Object-Oriented Design of Commonsense Theories 35
3.1 Introduction . . . 35
3.2 Objects for Situations and Actions . . . 36
3.2.1 Situation Calculus . . . 36
3.2.2 Object-Oriented FOL . . . 37
3.2.3 Situation Calculus Classes . . . 38
3.2.4 Modeling With One Object per Class . . . 39
3.2.5 Reasoning With Objects . . . 40
3.3 The Frame Problem . . . 41
3.4 Combining Action Theories . . . 43
3.5 Combining Different Agents . . . 45
3.6 Conditional Independence . . . 48
3.7 Related Work: Building Commonsense Theories . . . 48
3.8 Conclusions . . . 50

4 Reasoning with Partitioned Logical Theories 51
4.1 Introduction . . . 51
4.2 Partition-Based Theorem Proving . . . 53
4.2.1 Forward Message Passing . . . 54
4.2.2 Backward Message Passing . . . 61
4.2.3 Queries Drawn From Multiple Partitions . . . 62
4.3 Resolution and Resolution Strategies . . . 63
4.3.1 Resolution Message-Passing . . . 64
4.3.2 Linear Resolution . . . 68

4.3.3 Set-of-Support and Semantic Resolution . . . 69
4.3.4 Directional Resolution, A-ordering and Lock Resolution . . . 71
4.3.5 Analysis and Comparison of Resolution-Based Inference . . . 75
4.4 Partition-Based Propositional Satisfiability . . . 77
4.4.1 A Partition-Based SAT Procedure . . . 77
4.4.2 Analyzing Satisfiability in LINEAR-PART-SAT . . . 78
4.4.3 Order for Dynamic Backtracking . . . 79
4.5 Experimental Results . . . 80
4.6 Related Work: Structure in Reasoning . . . 81
4.6.1 Use of Decompositions in Propositional SAT . . . 81
4.6.2 Use of Decompositions in FOL Theorem Provers . . . 82
4.7 Conclusions . . . 83

5 Automatic Decomposition of Logical Theories 85
5.1 Introduction . . . 85
5.2 What is a Good Partitioning? . . . 87
5.3 Principles of Partitioning Logical Theories . . . 88
5.4 Split: Greedy Vertex Min-Cut in the Graph of Symbols . . . 89
5.4.1 Minimum Vertex Separators . . . 89
5.4.2 Procedure *Split* . . . 91
5.4.3 Fine-Tuning *Split* . . . 93
5.5 Treewidth . . . 94
5.6 Using 2-Way Vertex Separators . . . 95
5.6.1 Factor-4 Approximation Algorithm . . . 96
5.6.2 Factor-$4\frac{1}{2}$ Approximation Algorithm . . . 98
5.7 Using 3-Way Vertex Separators . . . 100
5.7.1 Multiway Vertex Cut . . . 100
5.7.2 Factor-$3\frac{2}{3}$ Approximation Algorithm . . . 101
5.7.3 Factor-$O(lgk)$ Approximation . . . 102
5.8 Experimental Results . . . 103
5.9 Related Work: Automated Decomposition . . . 104
5.9.1 Automated Decomposition for Reasoning . . . 104
5.9.2 Optimal Decomposition . . . 105

5.9.3 Other Decomposition Approaches for Logic . . . 106
5.10 Summary and Discussion . . . 107

6 A Logic-Based Subsumption Architecture 109
6.1 Introduction . . . 109
6.2 Principles of Subsumption Architectures . . . 110
6.2.1 Brooks' Subsumption Architecture . . . 110
6.2.2 Behavioral Decomposition . . . 112
6.2.3 Subsumption Principles . . . 113
6.3 Logic-Based Subsumption . . . 113
6.3.1 Basic Machinery . . . 114
6.3.2 Circumscription-Based Subsumption . . . 115
6.3.3 Putting It All Together . . . 116
6.4 Static Semantics for LSA . . . 118
6.5 Logical Layers for a Mobile Robot . . . 120
6.5.1 LAYER -1: Halt or Go . . . 120
6.5.2 LAYER 0: Obstacle-avoidance . . . 121
6.5.3 LAYER 1: Destination-seeking . . . 123
6.5.4 LAYER 2: Local Action Planning . . . 124
6.5.5 LAYER 3: Wide-Range Motion Planning . . . 125
6.5.6 The Default Assumptions of Robots' Layers . . . 126
6.5.7 Layers Do Not Always Have a Model of Time . . . 128
6.6 Implementation on a Mobile Robot . . . 128
6.6.1 Choosing a Theorem Prover . . . 129
6.6.2 Offline Experiments with PTTP and Simulated Sensors . . . 130
6.6.3 The Software and Dynamics of the System . . . 132
6.6.4 LiSA's behavior . . . 133
6.6.5 Tuning the System . . . 134
6.7 Related Work: Cognitive Robotics and Subsumption Architectures . . . 135
6.8 Conclusion . . . 137

7 Conclusions 139
7.1 Conclusions From This Work . . . 139
7.2 Summary of Results . . . 140

7.3 Future Directions and Open Problems . . . 140
7.3.1 Using Commonsense Reasoning in AI Applications . . . 140
7.3.2 Inference and Structure . . . 142
7.3.3 Machine Learning Using Partitions . . . 143

A Proofs **145**
A.1 Proofs for Chapter 2: OOFOL . . . 145
A.1.1 Theorem 2.5.2: Equivalent Semantics for Directional OOFOL . . . 145
A.1.2 Theorem 2.5.3: Completeness Proof for Directional OOFOL . . . 146
A.2 Proofs for Chapter 3: Object-Oriented Situation Calculus . . . 151
A.2.1 Proposition 3.4.1: Reiter's Frame Solution Keeps Links Thin . . . 151
A.3 Proofs for Chapter 4: Reasoning . . . 153
A.3.1 Theorem 4.2.4: FORWARD-M-P (MP) is Sound and Complete . . . 153
A.3.2 Corollary 4.2.8: MP and Intersection Graphs that are Trees . . . 154
A.3.3 Corollary 4.2.9: MP with BREAK-CYCLES is Sound and Complete . . . 154
A.3.4 Theorem 4.2.10: BACKWARD-M-P (BMP) is Sound and Complete . . . 155
A.3.5 Theorem 4.3.2: RESOLUTION-M-P (RES-MP) is Sound and Complete . . . 155
A.3.6 Lemma 4.3.6: Resolving on P is $\mathcal{L}_P$-Generation Complete . . . 156
A.3.7 Theorem 4.3.7: MP Can Simulate Variable Ordering (Most Times) . . . 158
A.3.8 Theorem 4.3.8: Variable Ordering Can Simulate MP (Propositional, Sometimes) . . . 160
A.3.9 Theorem 4.4.1: LINEAR-PART-SAT is Sound and Complete . . . 162
A.3.10 Lemma 4.4.2: Time Complexity of LINEAR-PART-SAT . . . 163
A.4 Proofs for Chapter 5: Decomposition . . . 164
A.4.1 Proposition 5.4.1: Time Complexity for SPLIT . . . 164
A.4.2 Lemma 5.6.2: $\frac{2}{3}$-vtx-sep finds a $\frac{2}{3}$-separator. . . . 164
A.4.3 Theorem 5.6.4: 2way-2/3-triang Finds a Factor-4 Approximation. . . . 165
A.4.4 Lemma 5.6.5: Existence of Two-Way $\frac{1}{2}$-Vertex-Separators of Size $k + \frac{1}{6}|W|$ 166
A.4.5 Lemma 5.6.6: Correctness of 2way-half-triang . . . 166
A.4.6 Lemma 5.6.7: Time Complexity and Correctness of $\frac{1}{2}$-vtx-sep . . . 167
A.4.7 Lemma 5.7.1: Correctness and Time for α-sum-sep . . . 167
A.4.8 Lemma 5.7.3: Correctness of lgk-triang . . . 168
A.5 Proofs for Chapter 6: LSA . . . 169

A.5.1 Theorem 6.4.1 and Corollary 6.4.2: Interpolation for Circumscription . . . 169
A.5.2 Theorem 6.4.4: LSA is Complete for *Sensors to Actions* . . . 172
A.5.3 Theorem 6.4.4: LSA is Sound for *Sensors to Actions* . . . 172

B More on Logical Systems Appearing in This Thesis 175
B.1 First-Order Logic and Propositional Logic . . . 175
B.1.1 Propositional Logic . . . 175
B.1.2 First Order Logic: Syntax and Semantics . . . 176
B.1.3 Proof Theory of First-Order Logic . . . 178
B.1.4 Model Theory of First-Order Logic . . . 181
B.1.5 Resolution Theorem Proving . . . 181
B.2 Nonmonotonic Reasoning Systems in Sections 2.5,6.3.2,6.4 . . . 186
B.2.1 Negation As Failure and Closed World Assumptions . . . 186
B.2.2 Circumscription . . . 191
B.2.3 Progression of Nonmonotonic Systems . . . 196

C More on Reactive Systems and Robot Control 199
C.1 AI Architectures . . . 199
C.1.1 SOAR . . . 199
C.1.2 CYC . . . 200
C.1.3 Blackboard Systems . . . 201
C.2 Reactive Systems . . . 202
C.2.1 Boddy & Dean – Time-Dependent Planning . . . 203
C.2.2 Brooks' Layered Control System . . . 203
C.3 Robot Motion Planning . . . 204
C.3.1 What Is a Robot? . . . 205
C.3.2 Basic Planning Methods . . . 205
C.3.3 Computational Complexity . . . 207
C.3.4 Reducing the Complexity . . . 208

List of Tables

5.1 Graphs, their processing time and the resulting width of the decomposition. 103

6.1 Proof time and inference steps measurements for the LSA during experiments in three different scenarios: (1) single-floor planning, (2) lower-level elevator planning, and (3) multi-floor planning. (*SD* is *standard deviation*.) 130

List of Figures

2.1 A Cup object. . . . 18
2.2 Connecting Cup and Liquid objects. . . . 19
2.3 A subclass of the **Cup** class. . . . 20
2.4 Some of the classes in the CAR theory and a theory fragment using these classes. . . . 22
2.5 Diagrammatic structure of the CAR theory fragment. . . . 23
2.6 An algorithm for proving Q from a (Directed) OOFOL theory T by message-passing. 25
2.7 A hierarchical decomposition. . . . 26
2.8 PMON in Directed OOFOL. . . . 27
2.9 Multiple Agents. . . . 28
2.10 Influence flows from $\mathcal{O}_1$ to $\mathcal{O}_2$. . . . 29

3.1 Blocks-world in the situation calculus: A_{bw}. . . . 37
3.2 A situation object connected to an action object. . . . 37
3.3 Blocks-world situation calculus classes. . . . 39
3.4 The OOFOL theory $\mathcal{A}_{ob}$ 40
3.5 Diagrammatic view of $\mathcal{A}_{ob}$ 40
3.6 Frame-problem solution using objects. . . . 42
3.7 Adding $Poss$ and explanation closure to **BWAct**. . . . 43
3.8 The domain of buying and selling blocks. . . . 43
3.9 Connecting two domain theories. . . . 44
3.10 High-level diagrammatic view of a theory combining multiple domains. . . . 45
3.11 Diagrammatic view of Daddy and Junior in object-oriented situation calculus. . . . 46
3.12 Merging perspectives on situations. . . . 47

4.1 A partitioning of $\mathcal{A}$ and its intersection graph. . . . 53
4.2 Axiomatization of a simplified espresso machine. . . . 54

4.3 A forward message-passing algorithm. 55
4.4 A proof of hot_drink from $\mathcal{A}$ in Figure 4.1 after asserting ok_pump (12) in $\mathcal{A}_1$ and ok_boiler (13), on_boiler (14) in $\mathcal{A}_2$. 56
4.5 An intersection graph before (left) and after (right) applying BREAK-CYCLES. . . 59
4.6 An algorithm to transform an intersection graph G into a tree. 60
4.7 A backward message-passing algorithm. 62
4.8 A procedure that transforms a graph G to a chain G'. 63
4.9 A resolution forward message-passing algorithm. 65
4.10 Subroutine RES-SEND using unskolemization. 66
4.11 Subroutine RES-SEND without unskolemization. 67
4.12 Linear resolution with forward message passing. 68
4.13 Semantic resolution with forward message passing. 71
4.14 An algorithm for SAT of a partitioned propositional theory. 78
4.15 Iteratively projecting and joining tables to check satisfiability. 79
4.16 An algorithm for SAT of a propositional theory, using decomposition and order over propositions. 80

5.1 A partitioning of $\mathcal{A}$ and its intersection graph. 89
5.2 Decomposing $\mathcal{A}$'s symbols graph. 90
5.3 An algorithm for finding a minimum separator between a and b in G. 91
5.4 An algorithm for generating partitions of axioms. 92
5.5 Recursive use of *Split* by aggregating minimal separators into single nodes. Only the larger side of the leftover graph is shown after each split. 93
5.6 Recursively using a *balanced separator* of previous separators. The algorithms' progression is depicted in (A) through (D), culminating with the tree at the bottom. 96
5.7 A factor-4 approximate triangulation algorithm. 97
5.8 Find a $\frac{2}{3}$-vertex-separator of W in G. 98
5.9 A factor-$4\frac{1}{2}$ approximate triangulation algorithm. 99
5.10 Find a two-way $\frac{1}{2}$-vertex-separator of W in G. 99
5.11 A factor-$3\frac{2}{3}$ approximate triangulation algorithm. 101
5.12 Find an α-sum-separator in G of size at most k. 101

6.1 Layers 0 and 1 of Brooks' subsumption architecture robot control system. 111
6.2 An abstract diagram of the LSA. 114

6.3 The LSA algorithm. 117
6.4 A detailed look at two layers. 117
6.5 Nomad 200 120
6.6 Diagrammatic view of an LSA system controlling a robot. 121
6.7 An example of simulated sonar input asserted in the theory. 131
6.8 An example proof by the theory. 132
6.9 Average time (in seconds) per cycle for each of the layers[1]. The bars for each layer correspond to total cpu-time per cycle, cpu-time for proofs in a cycle, total clock time per cycle and clock time for proofs in a cycle, in this order from left to right. . 133

A.1 Connecting two domain theories. 151

B.1 Progress through the different variants of Negation as Failure 191
B.2 Circumscription Systems 197
B.3 Nonmonotonic Reasoning Systems. 198

Chapter 1

Introduction

This thesis introduces a new approach to using logic for artificial intelligence. It presents the methodology of "divide-and-conquer" to help guide the creation of artificial intelligence applications that use logic efficiently. Particularly, it presents a methodology for building large commonsense knowledge bases that are made of parts, algorithms for reasoning with such knowledge bases, algorithms for the automatic partitioning of existing knowledge bases, and an applied robot-control architecture that is made of logic-based components.

Most of the ideas and results reported in this thesis are applicable to logic in general, not just to AI. Also, some of the results of Chapter 5 are advances in algorithmic graph theory. Finally, the theoretical results for first-order logic allow us to design new algorithms for automated reasoning in other representations, such as Bayesian networks, constraint satisfaction problems and Markov decision problems.

1.1 Representation, Reasoning and Control with Logic

1.1.1 Overview

This thesis applies decomposition to different methods of using logic in artificial intelligence (AI). My innovation is to use sets of logical theories, rather than a single theory, to represent knowledge about the world. I propose an AI architecture that uses such sets of theories, and I show how to obtain suitable theory sets, use them to drive robots and answer questions using the combined knowledge in the theories.

Logical theories (also known as knowledge bases) are sets of assertions about our world. For

example, a robot may have a theory that describes the way the robot itself works, the way the world behaves, the locations of objects and people, some laws of physics and the actions that the robot may perform. The robot can decide what action to perform (e.g., ”go forward with speed X”) by trying to prove that this action would be a step towards achieving its goal. My work rests on the tenet that intelligent computers (or robots) can be built if we represent knowledge in logical theories and let computers decide what to do by reasoning with them.

The idea of dividing theories into smaller sets has appeared in the literature before, but only for limited cases and with little methodology on how to reason efficiently with them or how to build them. Also, there was no architecture that allowed multiple theories to be embodied in the control of intelligent agents or robots, nor was there a practical architecture that could use an automated first-order logic (FOL) theorem prover for the reactive/online control of intelligent agents or robots.

1.1.2 What Is the Problem?

Logic is promising for AI because it has the ability to represent virtually every domain and problem that humans and other intelligent beings are interested in. Logic may not be the preferred way to represent certain types of knowledge, such as pictorial knowledge or algebraic knowledge, but logical representation of these knowledge types is feasible and allows us to discuss an overall architecture. AI researchers who study logical approaches (e.g., (McCarthy, 1958b)) advocate representing the world and the computer’s view of it in logical theories and then letting the computer use these logical theories to reason and act in the world. This approach was reinvigorated by several successful implementations (e.g., (Levesque et al., 1997; Shanahan, 1996)) that use logical theories for semantics of robot-control systems and for specifying high-level control. Also, efforts have been made to create large common-sense knowledge bases (e.g., (Lenat and Guha, 1990; Cohen et al., 1998)) for reasoning about natural language, supplying common sense to expert systems, intelligent agents and robots, and building programs that exhibit human-level intelligence.

Nonetheless, the use of general-purpose theorem provers for the control of intelligent agents or for reasoning about knowledge bases has so far been avoided, principally because their computational inefficiency precludes it (Levesque and Brachman, 1985). This fact, together with the difficulty of building or changing large general-purpose logical knowledge bases (McCarthy, 1998), has led research on logical approaches for artificial intelligence to focus on limited cases: restricted formal languages (e.g., description logics, Horn databases or propositional languages), restricted applications (e.g., frame systems, planning), limited uses of logic (e.g., formal treatments of already-built systems) and small domains (e.g., the blocks-world examples of (Sussman, 1990)).

1.1.3 What Are the Contributions of This Thesis?

In this thesis I report on new algorithms and methodologies that can be aggregated into four main results. The first is a general algorithm for efficient reasoning with sets of logical theories[1]. This algorithm is given a set of theories and a query that may require the combined knowledge in these theories. The algorithm uses traditional automated reasoning techniques for each of the theories separately. While these separate reasoning procedures are run, "messages" in the form of assertions are sent between the logical theories. These messages are added to those theories that received them, and the reasoning procedures that use those theories include these assertions in their continued reasoning. The algorithm uses a network structure that associates the theories to one another to decide the messages sent and their destinations.

This algorithm and several extensions and modifications were shown to answer correctly a comprehensive class of queries that they may be asked. They are faster than current state-of-the-art algorithms when the theories in the set are small and the structure of the theories-network is a tree that allows only a limited number of messages to be sent between connected theories.

The second result is a set of algorithms for the automatic decomposition of logical theories into smaller theories. These algorithms take a logical theory and use its syntax (the symbols that appear in the assertions in this theory) to suggest a division of the assertions into sets of smaller theories. They create a network of nodes associated with the symbols that appear in the theory and iteratively divide the set of symbols in two or three. At each iteration, a part that resulted from the previous iteration is further divided. The resulting set of partitions of symbols is used to create the division of the assertions. I have proved that these algorithms have guaranteed performance (speed of algorithm and optimality of results) that is better than that of previous algorithms. It was also experimentally verified that logical KBs (logical theories) that contain common-sense knowledge of everyday life (for example, two people cannot stand in exactly the same place) can be decomposed into the kind of partitions most suitable for fast reasoning.

Third, the thesis provides a methodology for building large, multi-domain sets of theories. The methodology uses an adaptation of object-oriented design to logical theories. The object-oriented design technique is one of the leading design methodologies of computer software (Gamma et al., 1994; Stroustrup, 1997; Darwin, 2001). In rough terms, it suggests that one should build knowledge in small fragments, each fragment focused on an object. To model a car, for example, we should collect knowledge about the steering wheel, the brakes, the battery, etc., and then connect those

[1] In the rest of this thesis we use the convention that every set of axioms is a *theory* (and vice versa). Also, unless stated otherwise, theories, axioms and knowledge bases (KBs) are in FOL.

collections into a large theory describing the car. I added the object-oriented tools of inheritance (e.g., every Toyota is a car, so it "inherits" the properties of a generic car) and interfaces to logic, allowing easy design and reuse of theories that relate to different domains. I have demonstrated the methodology on a few domains where traditional design methods have failed to scale up to large theories. In those domains the method has worked well. It remains to be examined if the methods scale up to large KBs (on the order of more than 1000 symbols and assertions).

Finally, the thesis provides an architecture for using sets of theories to control autonomous intelligent agents, such as robots. The main difference between the reasoning procedures described above and the architecture for control of autonomous agents is that the latter is reactive. The architecture is made reactive by adding one important feature: Each of the small theories has a set of default assumptions. For example, the robot concludes that the only objects that exist are those whose existence it can prove from its sensors and the statements in its theory. Each of the theories is given such (different) default assumptions. This allows each theory and local reasoning procedure to behave as if it is the only one available in the system, until messages from other theories arrive. The earlier conclusions are then retracted in favor of those conclusions that take this new information into account. Pedrito Maynard-Reid II and I have implemented this architecture for a mobile robot and showed the feasibility of this approach with several experiments.

1.1.4 Why Are These Results Important for Computer Science?

The immediate use of these algorithms and techniques is in allowing us to build and use large KBs about different topics, such as biology, human behavior and nature. Several such knowledge bases already exist (e.g., (Lenat, 1995; Cohen et al., 1998)) and there is interest in building more, larger KBs. Reasoning with these KBs can be done more efficiently using the algorithms presented in this thesis. Our working hypothesis is that, if the proper knowledge is collected, then such knowledge bases can be used in everyday life to answer queries from the Internet, help perform more intelligent searches, help in finding answers to problems that we humans struggle with, or make robots smarter.

The longer term use of this thesis and the most important idea coming from it, is that we can use logic from beginning to end to build a control system for a software agent or a physical agent. Logic may not always be the preferred basis of a system's design, but one can combine and present a complete control system made of parts using simple logical tools.

One way to build a control system is to collect the modules and functions that we believe an agent would need (e.g., visual identification of objects, identification of relationships between objects and over time, localization of the agent, higher-level reasoning and planning, self-diagnosis

capabilities, manipulation of the environment and of other agents and others), represent them in logic, connect them using logic, verify that the combined system performs the proper behavior (theoretically and experimentally) and then transform the logical parts into more efficient code, if desired.

The ability to apply the system in the real world in real-time without special-purpose code for the particular modules is essential because such systems are more readily debugged (i.e., axioms are more easily changed) and because a system built from general-purpose modules is more adaptive to its environment and more easily extendible. For example, if one module is influenced by knowledge sent to it from other modules, and if it is hard to predict the exact knowledge that would be sent from those modules, then our module should be ready to receive advice that we cannot predict before the system is deployed.

Another important insight that comes out of this thesis is that control using logic does not necessarily take the form of a single theory and a theorem prover. It may take many modules, each running its own mode of reasoning, possibly on different computers. With the aid of logical formalisms we can examine the meaning of the entire system and compare it to other running systems. This way we have a better understanding of larger control systems composed from many modules. The uniformity and relative simplicity of logic allows us to create large and reasonably transparent compound systems.

Logic can also represent many other representation languages, such as computer software languages (e.g., C++, Java, Lisp), probabilistic representation models, frame systems, databases, neural networks, algebraic systems, physical systems and hardware specifications. The results obtained here generalize many results obtained in some of those other systems (particularly probabilistic representation systems) and can be used to implement similar results in the rest.

My primary motivation now is human-level AI. One possible approach is a vastly distributed system along the lines I have suggested in this thesis. Human-level intelligent software can be built on the ready accessibility and cheap distribution of computers nowadays. This parallelism and distribution of computing power allows us to let every computer in a network reason with only a small fragment of the knowledge. The network does not need a central control node. The conclusions derived by each of the nodes are sent to adjacent nodes in the network, eventually joining the small pieces of work into a response to a user sitting in front of one computer in this network.

It is not very expensive to build a farm of 100,000 loosely coupled computers. SETI@home (W. T. Sullivan et al., 1997; Anderson and Werthimer, 2001) is one example of such a farm that was

built simply by asking users to contribute some of their CPU time. The shortage of AI applications that use this many computers is the main reason why we do not see many such endeavors. The methods proposed in this thesis suggest the use of such large distributed systems for fast automated reasoning and control of intelligent agents. Parallelism of this scale had no clear use for logic before this thesis. At the present I am pursuing this vision.

Some of these ideas and motivations are not new. The idea of decomposition and construction from parts is a fundamental concept in computer science and software engineering. However, this thesis suggests the use of new modules in the construction of complex systems that act in the world, namely, modules that perform automated reasoning in real time and respond in a principled manner to the other components of the system. It also suggests the reconstruction and reorganization of the system in an automatic manner to achieve new tasks or to perform tasks more efficiently or with more precision. Finally, the organization of components suggested by this thesis is close in spirit to connectionist approaches to artificial intelligence (Bishop, 1995), neuroscience (Churchland and Sejnowski, 1992) and cognitive psychology (Minsky, 1985). These can guide us in following up on this thesis and the creation of human-level intelligent agents.

1.2 The Technical Results of This Thesis

1.2.1 Object-Oriented Design of Logic-Based Commonsense Theories

FOL provides a richness of representation, but does not necessarily make change and reuse easy. The difficulty of building and reusing first-order theories is visible in the obstacles encountered in such areas as Commonsense Reasoning, Knowledge Reuse and Formal Verification (e.g., see (Lenat, 1995; Cohen et al., 1998; Cohen et al., 1999)). In Chapter 2 I take the object-oriented design paradigm and specialize it to the design of logical theories that describe various situations in the world. Chapter 3 shows how this methodology solves several problems related to the design of large axiomatizations describing the effects of actions on the world.

The object-oriented design paradigm has proved to be successful at facilitating software engineering, database engineering and knowledge engineering. The intuition behind our approach to object-oriented logic is that a theory can be presented as a collection of theories communicating with one another via *interface vocabularies*. From this point of view, *a theory is a graph of smaller theories* (the graph is possibly directed). If the interface vocabularies are small, the design of each object is mostly independent of the design of the rest of the theory. Modifications in the internals

of these theories have only local impact, an impact that is circumscribed by the interface vocabularies. Also, there is a significant performance improvement for reasoning with such theories over reasoning with ordinary FOL theories because of the algorithms that can be applied (the subject of Chapter 4 and Chapter 5). Finally, the directionality of such interfaces allows us to restrict local inconsistencies from influencing other parts of the theory.

Chapter 2 proposes adding object-oriented tools to FOL. In it I show that these tools add structure to FOL while not increasing the complexity of knowledge representation. This simplicity of representation ensures that excessive machinery is not needed to get the basic benefits offered by object-oriented design. This chapter gives the extension a simple semantics using FOL. A second extension that also allows for directing influence between modules is also discussed and shown to be semantically characterizable using the nonmonotonic logic of circumscription. Both of these logics use the object-oriented constructs of classes, objects and inheritance. In each of the logics, classes are theories equipped with interfaces, objects are copies of their classes with links to other objects, and inheritance is used to define classes that specialize their parents. An object-oriented theory is a collection of objects linked to one another.

I show the existence of efficient inference algorithms for both extensions to FOL, that take advantage of the structure they add. I also show that the extensions I provide are suitable and sufficient for several application areas. I do so by building example theories for physical systems (following the model-based reasoning approach), hierarchical domain models, theories of action and modeling of agents.

A specific case study is presented in Chapter 3 which deals with Situation calculus (McCarthy and Hayes, 1969). *Situation calculus* (McCarthy and Hayes, 1969) is a knowledge representation formalism for representing temporal information and the effects of actions done by agents. Situation calculus theories have a restricted form and a clear ontology that together help modeling. However, using situation calculus in large, multi-domain settings is not easy. It is hard to think of all the parts of a large theory in advance, reuse is not simple, and modifications can have unintended consequences. Giving situation calculus theories a partitioned representation allows us to use reasoning algorithms that make use of such structures. It also opens the way for merging situation calculus theories into reactive frameworks (see Chapters 4, 6 for details on such algorithms and systems).

I present one object-oriented reformulation of situation calculus, following some traditional decompositions of theories of action (Shanahan, 1997). It has a class for situations and a class for actions. *Subclasses* of class **Situation** (these are supersets of the class of situations, roughly speaking) contain domain constraints. Subclasses of class **Action** contain axioms describing the

effects of actions. I describe one theory constructed using objects of these and other classes. The combined theory has semantics that is equivalent to situation calculus, after translation to FOL. We apply Reiter's monotonic solution to the frame problem (Reiter, 1991) to this system of objects and classes.

Object-oriented situation calculus theories can be combined easily. We demonstrate this by combining domain theories. The approach also allows us to give a new treatment of agents in the situation calculus. Domain theories are connected by linking the corresponding objects with only a small vocabulary, even when the frame solution is used. This result extends to a representation of loosely interacting agents that we propose. Our treatment of agents allows each agent's theory to be written with no regard to the existence of other agents. Only new subtheories in which the agents interact need to consider the explicit existence of agents. We do this without giving up the *result* formalism (which is planning-friendly), as in (McCarthy and Costello, 1998), or the independence of the agents (which allows efficient computation), as in (Reiter, 1996), (Pinto, 1998). We point out the applications and influence of the algorithms of Chapter 4 on these theories. We conclude by proposing some guidelines for the knowledge engineering of situation calculus theories.

The combination of object-oriented ideas and logical knowledge representation is not new. Frame systems (e.g., (Brachman et al., 1983; Fikes et al., 1997)), Context systems (e.g., (Lenat and Guha, 1990; McCarthy, 1993; Giunchiglia and Traverso, 1995; Giunchiglia and Ghidini, 1998)), Object-Oriented Prolog dialects (e.g., (Shapiro and Takeuchi, 1983; Moss, 1994)) and logics describing object-oriented programs and databases (e.g., (Kifer et al., 1995; Lausen and Vossen, 1998; Maier, 1986)) are examples. Still, frame systems do not have the theoretical framework or tools to support object-oriented design in FOL. Most important, frame systems either have no semantic difference between a FOL axiom associated with one frame's ABox or another frame's ABox (e.g., Ontolingua (Gruber, 1993; Fikes et al., 1997)), or there is such a difference, but the semantics is complicated (context-supported CycL (Guha and Lenat, 1990)). Our work is applicable to these systems.

1.2.2 Automated Reasoning With Loosely-Related Theories

Some logical systems that are made of parts exist today. Some of the known ones are Cycorp's Cyc (Lenat and Guha, 1990), the DARPA-funded High Performance Knowledge Base (HPKB) systems developed by Stanford's Knowledge Systems Lab (KSL) (e.g., (Fikes and Farquhar, 1999)) and by SRI (e.g., (Cohen et al., 1998)), and the DARPA-funded RKF (Rapid Knowledge Formation) project. Chapter 4 proposes *partition-based* logical reasoning algorithms that take advantage of the

decomposed knowledge bases in a combined system, a decomposition proposed by the structure of object-oriented logical theories, or a decomposition proposed by an automated decomposition procedure. These algorithms reason with logical theories that are decomposed into related partitions of axioms.

We develop forward and backward message-passing algorithms, some variants that use different resolution strategies and an algorithm for propositional satisfiability. Our message-passing algorithms are designed so that, without loss of generality, reasoning within a partition can be realized by an arbitrary consequence-finding engine (Lee, 1967), which characterizes a class of reasoning procedures that ensures completeness and soundness of our algorithms. Chapter 4 investigates the impact of these algorithms on resolution-based inference and analyzes the computational complexity for our partition-based SAT.

1.2.3 Automated Decomposition of Logical Theories

A critical aspect of partition-based logical reasoning is the selection of a *good* partitioning of the theory. The computational analysis of our partition-based reasoning algorithms suggests parameters of partitioning that influence the computation of our algorithms: the *bandwidth* of communication between partitions, the size of each partition and the topology of the partitions graph. These parameters guide us in Chapter 5 when we propose algorithms for decomposing logical theories into partitions.

First, we look at a simple greedy method for decomposition that attempts to optimize the result according to the guidelines that we identify. This algorithm is fast, but may produce results that are far from optimal. Then, we show how the problem of automated decomposition can be solved using algorithms that find approximations to triangulations of minimum treewidth. The algorithms that we propose are guaranteed to be close to the optimal, but are slower than the first algorithm.

Given an undirected graph, G, and an integer k, the treewidth of G is at most k (Robertson and Seymour, 1986) if there is a *triangulation* of G with a clique number that is at most $k+1$ (the clique number of a graph is the size of the largest clique in this graph; a graph is triangulated if every cycle of length greater than 3 has a chord (an edge cutting the cycle)). This is also equivalent to finding a *tree decomposition* or a *junction tree* of G with width at most k. The solution of many graph-related NP-hard problems is possible in polynomial time if the graph has low treewidth and a triangulation of minimum treewidth is given (e.g., (Arnborg et al., 1991)). A good survey paper on finding the treewidth of graphs is (Bodlaender, 1997). A good book on the subject is (Kloks, 1994).

Following the introduction of our simple greedy algorithm, we present four approximation algorithms for finding triangulations of minimum treewidth. The first algorithm improves an algorithm of (Robertson and Seymour, 1995) and produces factor-4 approximations in time $O(2^{4.38k}n^2k)$, where n, k are the number of nodes and the treewidth of the given graph, G, respectively. The second algorithm improves an algorithm of (Becker and Geiger, 1996) and produces factor-$3\frac{2}{3}$ approximations in time $O(2^{3.6982k}n^3k^3lg^4n)$. The third algorithm produces factor-$4\frac{1}{2}$ triangulations in time $O(2^{3k}n^2k^{\frac{3}{2}})$. The last algorithm produces a factor-$O(lgk)$ approximation in time $O(n^3lg^4nk^5\,lgk)$.

The time bounds achieved by the first and second algorithms are faster by factors of $O(2^{0.4k})$ and $O(2^k poly(n))$, respectively, than previously available algorithms for these approximation factors. The third algorithm has faster combined n, k time than any known algorithm that produces a constant-factor approximation. The last algorithm is the first polynomial-time algorithm that approximates the optimal by a factor that does not depend on n. We have implemented the 4-approximation and $4\frac{1}{2}$-approximation algorithms and used them to find tree decompositions of graphs used in the a subset of the HPKB project (Cohen et al., 1999), a subset of the CYC knowledge base (Lenat, 1995), and several CPCS Bayesian networks (Pradhan et al., 1994). These graphs have between 100 and 600 nodes and between 400 and 4000 edges. Our results compare favorably with the algorithms of (Becker and Geiger, 1996; Shoikhet and Geiger, 1997).

1.2.4 An Agent Control Architecture Based on Automated Theorem Provers

In (Brooks, 1986), Rodney Brooks provided a decomposition of the problem of robot control into layers corresponding to levels of behavior, rather than a sequential, functional form. Within this setting, he introduced the idea of subsumption, that is, that more complex layers not only depend on lower, more reactive layers, but could also influence their behavior. The resulting architecture was one that could service multiple, potentially conflicting goals simultaneously in a reactive fashion, giving precedence to high-priority goals.

The problem of extending similar architectures to more complex tasks and goals and to agents that are not necessarily physical has already been raised and discussed in general terms by (Minsky, 1985) and (Stein, 1997), but to our knowledge, no practical AI architecture that includes high-level reasoning has been developed along these lines.

In Chapter 6 we describe an architecture that is designed in the spirit of Brooks' subsumption architecture but relies on a logical framework and has wider applicability and extendibility. Our *Logic-Based Subsumption Architecture* (LSA) includes a set of FOL theories, each corresponding to a layer in the sense of Brooks' architecture. Each layer is supplied with a separate theorem

prover, allowing the system of layers to operate concurrently. After proving a goal, each layer sends the result to lower layers. We use nonmonotonic reasoning to allow layers to make default assumptions. These assumptions may be subsumed when information arrives from higher layers or from the agent's sensors. This allows each layer's performance to be independent of the performance of other layers, supporting reactivity. We also use nonmonotonic reasoning to provide semantics for the combined (static) system.

The architecture has been implemented and tested on a Nomad 200 mobile robot. It exhibits real-time performance and performs navigation and control tasks. The layers can receive and incorporate new axioms from the user at run-time, allowing the user to give advice to the robot and to correct erroneous behaviors. The architecture also allows for incorporating layers that perform diagnosis and can be extended to layers that remember experiences for other layers. Our experiments show that real-time commonsense control of AI agents can be achieved with an architecture based on logical theories and general-purpose theorem provers with the use of subsumption. These theorem provers can be replaced with special-purpose reasoners for improved efficiency, but we show that in simple cases this is not needed with current theorem-proving technology.

This work improves on results presented using GOLOG (e.g. (Levesque et al., 1997)) and the work of (Shanahan, 1996) in that our system is fully declarative and has the full expressiveness of FOL. We provide a more detailed comparison to these and other related work at the end of Chapter 6.

1.3 Logic Needed to Read This Thesis

The literature on the large field of logic cannot faithfully be presented in a single chapter, even if we limit ourselves to the literature that is most relevant to AI. A series of handbooks (Gabbay et al., 1993a; Gabbay et al., 1993b; Gabbay et al., 1993c; Gabbay et al., 1993d; Gabbay et al., 1993e) and textbooks (Genesereth and Nilsson, 1987; Fitting, 1990; Brewka et al., 1997) that are dedicated to logic in AI offers a good treatment and introduction. Thus, this section includes only background material that is related to AI and is used in this thesis. The reader may find it useful to have read at least one textbook on mathematical logic (e.g., (Shoenfield, 1967; Enderton, 1972)) or have taken a course on the subject. We give a brief account of basic logical notions in Appendix B.1 as a reference in case some of the notations or vocabulary are ambiguous.

This thesis focuses on FOL and propositional logic. The following describes the vocabulary and notations that are used in the rest of this thesis for these logics. If you read this section and find the

terms familiar, then you should be able to read and understand almost all of the thesis without any extra reading. Specific details of the vocabulary and more advanced concepts are included in the thesis where needed.

Propositional logic theories are sets of axioms that are built from propositional symbols together with the connectives $\neg$ (negation), $\vee$ (disjunction) and $\wedge$ (conjunction). Propositional *literals* are propositional symbols or their negation (e.g,. p or $\neg p$ are literals, but $\neg\neg p$ is not). Clauses are disjunctions of literals. Often, clauses are represented as sets of literals (e.g., $p \vee \neg q \vee r$ is represented by $\{p, \neg q, r\}$). The semantics of a propositional theory is given by the set of truth assignments that satisfy the theory.

FOL theories are sets of axioms that are built from constant, function, predicate (relation) and variable symbols together with the connectives $\neg$ (negation), $\vee$ (disjunction) and $\wedge$ (conjunction) and the quantifiers $\forall$ (for all) and $\exists$ (exists). For example, the logical sentence $\forall x \exists y\ likes(x, y) \vee likes(y, x)$ reads "for every x there is a y such that the x likes y or y likes x".

In this thesis we consider every set of axioms as a theory. In the context of FOL, we use the symbols $A, B, C, ...$ (upper-case English letters from the beginning of the alphabet) to refer to constant symbols; $x, y, z, ...$ (lower-case English letters from the end of the alphabet) to refer to variables, $f, g, h, ...$ to refer to function symbols; and $P, Q, R, ...$ to refer to predicate symbols.

FOL *literals* are atomic formulae or their negation (e.g., $R(f(g(x), A), h(B))$ is a literal, but $P(A) \vee Q(f(x))$ is not). *Clauses* in FOL are axioms that include only disjunctions of literals (e.g., $P(x) \vee \neg Q(y, x) \vee R(f(A), f(y))$ is a clause while $P(x) \wedge Q(A)$ and $\forall x \exists y Q(x, y)$ are not). In FOL as well as in propositional logic, clauses are represented often by sets of literals (e.g., $P(A) \vee Q(f(x))$ is a clause that can be represented by $\{P(A), Q(f(x))\}$).

In this thesis we use L to denote propositional or FOL signatures (the set of symbols used in the language, their type (predicate, function, constant, propositional) and arity (when relevant)). We use $\mathcal{L}$ to denote the language (a set of formulae) relevant to our context. Theories are denoted with the symbols $\mathcal{A}, \mathcal{B}, T$, their signature is denoted by $L(T)$ and their language (the set of FOL or propositional logic formulae built using the symbols in $L(T)$) is denoted by $\mathcal{L}(T)$.

The semantics of a FOL theory is given by the set of interpretations that satisfy the theory (these are called *models* of the theory). We use $\mathcal{M}, \mathcal{N}, ...$ to denote models of a theory and A, B to denote interpretations (structures) of a language. $\mathcal{M} \models T$ says that $\mathcal{M}$ is a model of T. For two formulae, $\varphi, \psi \in \mathcal{L}$, $\varphi \vdash \psi$ means that there is a proof of ψ from φ, and $\varphi \models \psi$ means that every model of φ is a model of ψ. $Cn(\mathcal{A})$ is the set of logical consequences of $\mathcal{A}$ (i.e., those formulae that are valid consequences of $\mathcal{A}$ in FOL). For a first-order structure A in L, we write $U(A)$ for the universe

of elements of A. Finally, for every symbol s in L, we write s^A for the interpretation of s in the structure A.

1.4 Plan of This Thesis

The thesis stands at the intersection of several sub-areas of artificial intelligence and computer science, namely, mathematical logic and automated logical reasoning, commonsense reasoning and knowledge representation, AI architectures, robot control, object-oriented design and algorithmic graph theory. Nonetheless, the reader is not assumed to have a thorough background in any of these areas, and I have made some effort to make the thesis readable to the general audience in artificial intelligence. Most of the thesis should be comprehensible to a general computer scientist as well.

Chapters 2-6 present the main contributions in this thesis. The chapters are self-contained and can be read separately from the rest. Chapter 2 describes an object-oriented approach to modeling in FOL. Chapter 3 uses this methodology for the case of commonsense theories that talk about actions and situation in situation calculus. Chapter 4 presents reasoning algorithms for reasoning with partitioned theories. Chapter 5 provides algorithms for automatic decomposition of logical theories. Chapter 6 describes an AI architecture that drives a robot and is based on logical theorem provers and Brooks's subsumption architecture.

Those readers most interested in the design of large knowledge bases are advised to read Chapters 2, 3 and then chapters 4, 5. Those interested in automated reasoning should start with Chapters 4, 5 and continue to Chapters 2, 3. Those interested in AI architectures and robot control are advised to read chapters 6, 3, 4 in this order, possibly adding Chapters 2, 5.

Chapter 7 summarizes the results of this thesis, draws a few conclusions and presents a set of open questions and directions for further investigation.

Some of the material in this thesis has appeared before in (**?**; Amir and Maynard-Reid II, 1999; Amir and McIlraith, 2000; Amir, 2000a; Amir and McIlraith, 2001a; Amir, 2001; McIlraith and Amir, 2001).

Chapter 2

Object-Oriented Design in Logic

2.1 Introduction

There is growing interest in building large knowledge bases (KBs) of everyday knowledge about the world, teamed with theorem provers to perform inference. Three such systems are Cycorp's Cyc (Lenat and Guha, 1990) and the High Performance Knowledge Base (HPKB) systems developed by Stanford's Knowledge Systems Lab (KSL) (e.g., (Fikes and Farquhar, 1999)) and by SRI (e.g., (Cohen et al., 1998)). These KBs comprise tens/hundreds of thousands of logical axioms. One approach that we advocate for dealing with the complexity of building such KBs is to use object-oriented design techniques and tools. In this chapter we examine the problem of modeling large knowledge bases in First-Order Logic (FOL) using object-oriented design. We add object-oriented design tools to FOL in an effort to help with building such KBs.

FOL provides a richness of representation, but does not necessarily make change and reuse easy. The difficulty of building and reusing first-order theories is visible in the obstacles encountered in such areas as Commonsense Reasoning, Knowledge Reuse and Formal Verification (e.g., see (Lenat, 1995; Cohen et al., 1998; Cohen et al., 1999)). On the other hand, the Object-Oriented paradigm has proved to be successful at facilitating software engineering database engineering and knowledge engineering. (e.g., (Lausen and Vossen, 1998; Meyer, 1997; Brachman et al., 1983; Lenat and Guha, 1990; Koller and Pfeffer, 1997)).

In this chapter, we propose adding object-oriented tools to FOL. I show that this extension does not increase the expressive power of FOL by giving it a simple semantics using FOL. A second extension that also allows for directionality of influence is shown to be semantically characterizable using Circumscription. In these logics, classes are theories equipped with interfaces, objects are

copies of their classes with links to other objects, and inheritance is used to define classes that specialize their parents.

The object-oriented tools that we provide are suitable and sufficient for several application areas. We show this by building example theories for physical systems (following the model-based reasoning approach), hierarchical domain models, theories of action and modeling agents. We also show that there are reasoning algorithms that can take advantages of the object-oriented structure of KBs built from this object-oriented FOL, providing further motivation for building KBs using this methodology. These algorithms receive the focus of attention in Chapter 4.

The combination of object-oriented ideas and logical knowledge representation is not new. Frame systems (e.g., (Brachman et al., 1983; Fikes et al., 1997)), Context systems (e.g., (Lenat and Guha, 1990; McCarthy, 1993; Giunchiglia and Traverso, 1995; Giunchiglia and Ghidini, 1998)), Object-Oriented Prolog dialects (e.g., (Shapiro and Takeuchi, 1983; Moss, 1994)) and logics describing object-oriented programs and databases (e.g., (Kifer et al., 1995; Lausen and Vossen, 1998; Maier, 1986)) are examples. Still, frame systems do not have the theoretical framework or tools to support object-oriented design in FOL. Most remarkably, frame systems either have no semantic difference between a FOL axiom associated with one frame's ABox or another frame's ABox (e.g., Ontolingua (Gruber, 1993; Fikes et al., 1997)), or there is such a difference, but the semantics is complicated (context-supported CycL (Guha and Lenat, 1990)). Our work is applicable to these systems.

We expand on this comparison at end of this chapter. Before that, Section 2.2 discusses the problem of building and reusing theories, Section 2.3 describes the motivation and structure of our approach, Section 2.4 provides the syntax, semantics, reasoning algorithms and a few examples for using Object-Oriented FOL (OOFOL), and Section 2.5 provides the semantics, reasoning algorithms and example uses of a version of OOFOL that allows directions associated with the links between objects.

Throughout this chapter, we treat logic as an engineering tool and view it through the eyes of an engineer. This approach influences our design decisions, as will be evident. The results in this chapter appeared previously in (**?**).

2.2 The Problem of Building and Reusing Logical Theories

When creating a knowledge base, a knowledge engineer faces design and modeling choices. The decisions taken may later affect the evolution of the knowledge base. Indeed, when new knowledge

needs to be integrated, the knowledge base may have to undergo some rewriting and redesign in order to incorporate the new information. Creating the knowledge base such that future changes are made easier is a major concern of the knowledge engineer.

One simple example is the following. Assume that we are given the logical theory

$$Rain \implies Cold \tag{2.1}$$
$$Wet \implies Cold \tag{2.2}$$

Now, say we want to add the fact that in the tropics it is not cold but it may rain and it is wet. One way to do that is to rewrite the theory from scratch:

$$Rain \implies (Cold \vee Tropics)$$
$$Wet \implies (Cold \vee Tropics)$$

But, consider the following theory, which is similar to the original theory above.

$$Rain \implies Preconditions \tag{2.3}$$
$$Wet \implies Preconditions \tag{2.4}$$
$$Preconditions \implies Cold \tag{2.5}$$

In this theory we added the observation that being wet and having rain are two properties that share preconditions. In this case, instead of rewriting the entire theory from scratch, we can simply replace the third sentence (2.5) with

$$Preconditions \implies (Cold \vee Tropics)$$

We would like to prefer compact changes, like the last one, over rewriting extended sections of our knowledge base, as seen in the first case.

Coming up with a a set of design rules that can accomodate every change seems like an open-ended task (in fact, it was shown to be practically impossible in (Amir, 2000b)). These may include design decisions like Object-Oriented design (e.g., using frames (see (Minsky, 1975) and (Brachman and Levesque, 1985))), but may also include simple aggregations such as the one demonstrated in the example above.

The remainder of this thesis focuses on one proposed approach to tackling this problem using object-oriented design. Other investigations under the banner of *elaboration tolerance* of commonsense knowledge bases include (McCarthy, 1998), who gives some examples and discusses intuitions of elaboration tolerance, (Lifschitz, 2000; Gustafsson and Kvarnstrom, 2001), who show how their systems can treat some of these example elaborations, (Amir, 2000b), who provided a preliminary formal investigation of the problem, and (Costello, 1997; Amir, 1997), who showed how different theories of action relate with respect to certain elaborations. Other relevant material is (Giunchiglia and Walsh, 1992) and the work on abstraction and reformulation.

2.3 An Object-Oriented Approach to Logic

The intuition behind our approach to Object-Oriented logic is that a theory can be presented as a collection of theories communicating with one another via *interface vocabularies*. From this point of view, *a theory is a graph of smaller theories* (the graph is possibly directed). If the interface vocabularies are small, the design of each object is mostly independent of the design of the rest of the theory. Modifications in the internals of these theories have only local impact, an impact that is circumscribed by the interface vocabularies. Also, there is a significant performance improvement for reasoning with such theories over reasoning with ordinary FOL theories because of the algorithms that can be applied (this is the subject of Chapters 4,5). Finally, in some cases we may want influence to flow between objects only in one direction. Extensions of our object-oriented system to allow such designs allow us to restrict inconsistencies from influencing other parts of the theory.

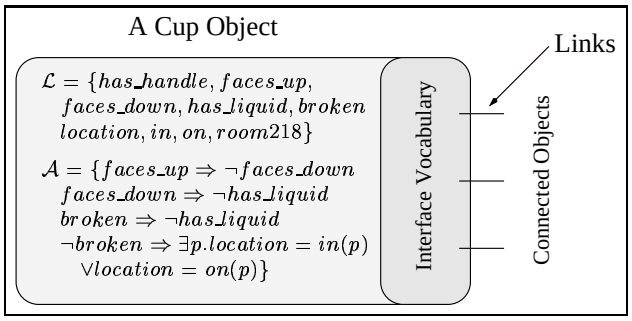

Figure 2.1: A Cup object.

We describe the structure of objects through a simple example. The $\mathcal{C}up$ object, depicted in Figure 2.1, describes some of the characteristics of a typical cup. As with other objects, it has a

vocabulary, a set of axioms, an interface vocabulary and a set of links to other objects. In its simple form (non-directional), a link specifies an equality between two interface-vocabulary symbols in two respective objects.

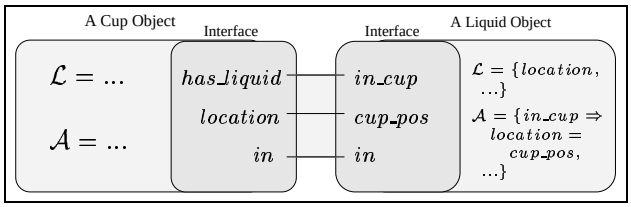

Figure 2.2: Connecting Cup and Liquid objects.

Two linked objects are presented in Figure 2.2. Some of the interface symbols of the $\mathcal{C}up$ object are linked to the interface symbols of the $\mathcal{L}iquid$ object. The link specifies that has_liquid in the object $\mathcal{C}up$ has the same semantics as in_cup in the object $\mathcal{L}iquid$. The same goes with the pairs $location, cup_pos$ and in, in. From a FOL perspective, it is as if we added the equality/equivalence axioms $Cup.has_liquid \equiv Liquid.in_cup$, $Cup.location = Liquid.cup_pos$ and $\forall x(Cup.in(x) = Liquid.in(x))$. Readers familiar with applications of context (e.g., (Ghidini and Serafini, 1998)) can view the following as a special case having the semantics of FOL, object-oriented tools and no explicit contexts. More on that in Section 2.6.

From an automated reasoning perspective, describing a theory as a set of objects allows the following style of reasoning: All objects generate consequences in parallel, sending results to those other objects interested in them. These results are then incorporated into the theorem-proving process of the receiving objects. Axioms sent from object to object are restricted to the communication vocabulary, but the language is full FOL. For example, look again at Figure 2.2. If an object $\mathcal{L}iquid$ has a link to the *Cup1* object that includes $has_liquid, location, in$ (on the side of *Cup1*), then a sentence like $has_liquid \Rightarrow \exists p\ location = in(p)$ will be translated and sent to $\mathcal{L}iquid$ in the form $in_cup \Rightarrow \exists x\ cup_pos = in(x)$.

The use of fixed interfaces allows the encapsulation of the internals of objects. This way, we can add or replace internal axioms with higher-grain axioms, describing the type of liquid, the heat, the degrees of tilt of the cup or the material of which the cup is made. Since we can expect the interface to remain the same, we can verify the change by examining changes to results in the interface, paying less attention to other internal results (possibly applying the techniques of (Inoue, 1992)).

To allow reuse of theories and following object-oriented design tradition, we define a *class* to be a "template" of objects. A class can be thought of as an object without specified links. Alternatively,

an object is simply a copy of its class with links to other objects. We allow different uses of two objects of the same class in the same theory (linked to possibly different objects).

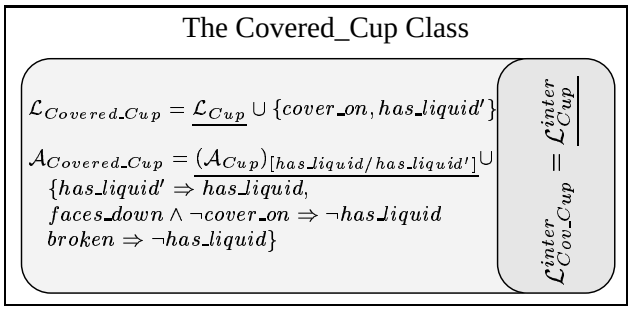

Figure 2.3: A subclass of the **Cup** class.

Inheritance is yet another object-oriented tool used in many KR systems. It is sometimes reasoned about using nonmonotonic reasoning systems (e.g., (McCarthy, 1986a; Morgenstern, 1996; Thomason et al., 1986)). In contrast, we implement inheritance of a class from its parent class outside the logic, as we are interested in it as an engineering tool.

Figure 2.3 describes **Covered_Cup**, a subclass of **Cup**; the inherited part is underlined. A subclass inherits *all the axioms* and vocabulary of its superclass, but may add symbols to the vocabulary and the interface vocabulary and may also add axioms. To enable the subsumption of axioms, we allow symbols used in the superclass to be replaced by new symbols (i.e., the subsumption of axioms is done by subsuming symbols[1]). In our example, the symbol has_liquid is substituted by has_liquid' in all of $\mathcal{A}_{Cup}$ to allow for a new definition for has_liquid.

This kind of inheritance is similar to inheritance in Object-Oriented programming languages, where the interface of the subclass must include the interface of the superclass, but may implement some of the internals differently. This definition allows the specialization of a class by adding more detail in its subclasses.

2.4 Object-Oriented First-Order Logic

In what follows, we use some notations that are common in object-oriented programming languages, namely, $\mathcal{O}bj.c$ (or $\mathcal{N}.c$), to refer to the symbol c in the object $\mathcal{O}bj$ (with the name $\mathcal{N}$). For simplicity of exposition, we shall not define classes and objects recursively (i.e., classes whose theories are

[1]This enables the reuse of superclass results in the subclass.

themselves OOFOL theories). A recursive definition may sometimes be useful, but it requires only a simple technical step and makes our notations significantly more dense. Thus, this definition is omitted, but we note that all the results that follow hold for this more general case.

2.4.1 Language

First, we define our language: classes and objects, interfaces, links and theories. Unless otherwise mentioned, all languages mentioned are full first-order.

A *Class*, **C**, is a tuple $\langle \mathcal{L}_C, \mathcal{A}_C, \mathcal{I}_C \rangle$ corresponding to a FOL vocabulary ($\mathcal{L}_C$), a set of FOL axioms ($\mathcal{A}_C$) in this language, and $\mathcal{I}_C \subset \mathcal{L}_C$, the interface vocabulary of C.

An *OOFOL Theory* T is a set of statements

$$Object(\mathbf{C}, \mathcal{N}, Link),$$

(defining an object), where **C** is a class reference (a class description as above, or a reference to such a description), $\mathcal{N}$ is a unique name for the object (typically a string of symbols) and $Link$ is a relation $Link = \{(P_i, \mathcal{N}_i, Q_i)\}_{i \leq n}$ that specifies the links between this object and other objects in the theory. Here, P_i is a symbol in the interface vocabulary of **C**, $\mathcal{N}_i$ is a name of another object in T, and Q_i is an interface symbol of this other object. This way, an object is simply "declared" to be in the theory. We assume that no statement in T refers to a name $\mathcal{N}_i$ (in the linking relation) for which no corresponding object exists in T. In typed systems, we would also assume that P_i, Q_i have the same type.

2.4.2 Example Theory: Model-Based Reasoning

The next section provides the basic tools suggested by the object-oriented paradigm: classes, objects, interface and inheritance. This section gives a simple example application of these tools in the context of model-based reasoning (a more elaborate example in the context of theories about effects of actions is given in Chapter 3).

Below, we give a fragment of a theory describing the functionality of a car. We use qualitative proportionalities (see (Forbus, 1996)) to represent the relationships among different variables of the car (in our case, these are monotonically increasing and monotonically decreasing functions). Figure 2.4 on the next page describes some of the classes for this domain and the theory fragment.

The notation *class* **C1** *:* **C2** indicates that class **C1** is a subclass of class **C2**. For example, we have some knowledge about ordinary brakes systems, but we also have some specialized knowledge

class Qualitative_Proportionalities {
axioms: $\forall xy(x > y \Rightarrow rel_pos(x) > rel_pos(y))$
$\forall xy(x > y \Rightarrow rel_neg(x) < rel_neg(y))$ }

class Brakes : Qualitative_Proportionalities {
interface: $apply_force(x)$
axioms: $\forall xy(pedal_pressure(x) \wedge oil_pressure(y) \Rightarrow$
$apply_force(rel_pos(x) * rel_pos(y)))$
$pedal_pressure(0) \Rightarrow apply_force(0), oil_pressure(70)$ }

class ABS : Brakes {
interface: $apply_force(x), abs(x)$
axioms: $\forall x(old_apply_force(x) \wedge x \geq ABS_step \Rightarrow apply_force(abs(x)))$
inherit: $[apply_force/old_apply_force]$ }

class Tires : Qualitative_Proportionalities {
interface: $bad_traction(wheel), apply_force(x), engine_force(x), brake_force(x)$
axioms: $\forall xy(brake_force(x) \wedge engine_force(y) \Rightarrow$
$apply_force(rel_pos(y) - rel_pos(x)))$
$\forall wheel(\neg balanced(wheel) \vee old(wheel) \Rightarrow bad_traction(wheel))$
$\forall wheel(balanced(wheel) \wedge new(wheel))$
$\forall x(new(x) \iff \neg old(x))$ }

class Car : Qualitative_Proportionalities {
interface: $tire_force(x), bad_traction(wheel)$
axioms: $\exists w.bad_traction(w) \wedge tire_traction(t) \Rightarrow t < Good,$
$\forall t(\forall w.\neg bad_traction(w)) \wedge tire_traction(t) \Rightarrow t = Good$
$\forall fts(tire_force(f) \wedge tire_traction(t) \wedge speed(s) \Rightarrow$
$stop_time(rel_neg(s) * rel_pos(f) * rel_neg(t)))$ }

$$T\left\{\begin{array}{l} Object(\textbf{ABS}, \mathcal{B}rakes1, \{\}). \\ Object(\textbf{Tires}, \mathcal{T}ires1, \{(brakes_force, \mathcal{B}rakes1, apply_force)\}). \\ Object(\textbf{Car}, \mathcal{C}ar1, \{(tire_force, \mathcal{T}ires1, apply_force), \\ \quad (bad_traction, \mathcal{T}ires1, bad_traction)\}). \end{array}\right.$$

Figure 2.4: Some of the classes in the CAR theory and a theory fragment using these classes.

about anti-lock brakes systems (ABS). The class **ABS** is a subclass of **Brakes** that inherits all the axioms of **Brakes**, replacing the symbol $apply_force$ with old_apply_force (in those inherited axioms). In this subclass, we also add one more axiom and add an interface symbol abs. The object named $\mathcal{B}rakes1$ is declared of class **ABS**, but is used as an instance of class **Brakes**. The theory including the three objects $\mathcal{B}rakes1$, $\mathcal{T}ires1$ and $\mathcal{C}ar1$ is semantically/diagrammatically depicted in Figure 2.5.

Using this theory, we can prove that increasing the pressure on the brakes will make the car stop faster, if we keep the rest constant (e.g., not increase the pressure on the accelerator). Notice that we chose undirected semantics (FOL semantics) because influence may flow in both directions between the objects: A tire explosion may influence the force applied on the brakes.

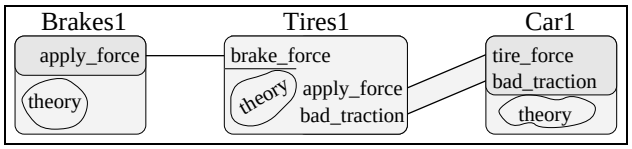

Figure 2.5: Diagrammatic structure of the CAR theory fragment.

2.4.3 First-Order Semantics

Unlike object-oriented approaches to logic that use context (see Section 2.6), we take advantage of the limited expressivity of our language (we cannot quantify over objects) and the limited interaction among our objects to give a simple FOL semantics to OOFOL theories.

We define the semantics by making objects have mutually exclusive vocabularies and viewing the links as equality/equivalence assertions. For an OOFOL theory T, define

$$L(T) = \{\mathcal{N}.P \mid Object(\mathbf{C}, \mathcal{N}, Link) \in T,\ P \in L_C\}.$$

Also, define $\mathcal{L}(T) = \mathcal{L}(L(T))$, the language built from the non-logical symbols in $L(T)$. This set of symbols guarantees a unique vocabulary for each object in T. We apply a simple translation into this vocabulary (replacing P by $\mathcal{N}.P$ in the axioms of the object $\mathcal{N}$ for each symbol P in its vocabulary) and add equality/equivalence assertions:

$$\begin{aligned}\widetilde{T} = &\{(\mathcal{A}_C)_{[P/\mathcal{N}.P \mid P \in L_C]} \mid Object(\mathbf{C}, \mathcal{N}, Link) \in T\} \bigcup \\ &\{\forall \overrightarrow{x}(\mathcal{N}.P(\overrightarrow{x}) \equiv \mathcal{N}'.P'(\overrightarrow{x})) \mid Object(\mathbf{C}, \mathcal{N}, Link) \in T,\ Link(P, \mathcal{N}', P')\}\end{aligned} \tag{2.6}$$

where "$\equiv$" means "$=$" if P, P' are function symbols, and $\overrightarrow{x}$ may be of arity 0 (if P, P' are propositional or constant symbols).

An *OOFOL structure* in the vocabulary $L(T)$ is a FOL structure of the signature $L(T)$.

Definition 2.4.1 (FOL Semantics for OOFOL) *Let T be an OOFOL theory and let $\mathcal{M}$ be an OOFOL structure of $L(T)$. A formula $\varphi \in \mathcal{L}(T)$ is satisfied in $\mathcal{M}$, written $\mathcal{M} \models \varphi$, iff $\mathcal{M}$ satisfies φ in the FOL sense.*

For OOFOL theory T, $\mathcal{M}$ is a model of *T, written $\mathcal{M} \models T$, iff $\mathcal{M} \models \widetilde{T}$ (in the FOL sense).*

In other words, the semantics of an OOFOL theory as an undirected graph of connected theories has the links state equality/equivalence among symbols and otherwise has the theory behave like a FOL theory.

EXAMPLE If T is the OOFOL theory containing the two connected objects from Figure 2.2, then

$$T = \{Object(\textbf{Cup}, \mathcal{C}up1, \{(has_liquid, \mathcal{L}iquid1, in_cup),$$
$$(location, \mathcal{L}iquid1, cup_pos), (in, \mathcal{L}iquid1, in)\}),$$
$$Object(\textbf{Liquid}, \mathcal{L}iquid1, \{\})\}.$$

The equality/equivalence sentences that we add in defining $\widetilde{T}$ and T's semantics are:

$$\mathcal{C}up1.has_liquid \equiv \mathcal{L}iquid1.in_cup$$
$$\mathcal{C}up1.location = \mathcal{L}iquid1.cup_pos$$
$$\forall x(\mathcal{C}up1.in(x) = \mathcal{L}iquid1.in(x)).$$

Notice that the definition of $\mathcal{L}iquid1$ above did not include any links, because the links were already specified for $\mathcal{C}up1$. ■

2.4.4 Inheritance

Unlike nonmonotonic inheritance (as in (Morgenstern, 1998; Kifer et al., 1995)), inheritance in our system is treated outside the logic, having the engineer specify its exact use. We wish to allow three differences between a class and its parent: (1) the child's vocabulary and interface vocabulary are expansions of the parent's; (2) the child class inherits all the axioms of the parent, subject to the replacement of non-logical symbols (in all of the axioms in the same way) with some of the new symbols in the child's language; and (3) the child class may have additional axioms.

Formally, for a set of axioms A and non-logical symbols c, c', $A_{[c/c']}$ is the set of axioms A with every occurrence of the symbol c replaced by an occurrence of the symbol c'. $A_{[c_1/c'_1,\ldots,c_n/c'_n]}$ is similarly defined. A class $\mathbf{C} = \langle \mathcal{L}, \mathcal{A}, \mathcal{I} \rangle$ is a *subclass* of $\mathbf{C'} = \langle \mathcal{L}', \mathcal{A}', \mathcal{I}' \rangle$ if $\mathcal{L}' \subseteq \mathcal{L}, \mathcal{A}'_{[c_1/c'_1,\ldots,c_n/c'_n]} \subseteq \mathcal{A}$ ($n \geq 0$) and $\mathcal{I}' \subseteq \mathcal{I}$. A detailed example is provided in Section 2.4.2 on page 21.

2.4.5 Reasoning Algorithms for OOFOL

Algorithms for inference in OOFOL may reduce the theories to the appropriate semantics and perform inference on the result. We are more interested in algorithms that can take advantage of the special structure of our theories. We present such an algorithm for the special case of totally ordered theories. The general case can be reduced to this one using the methods of Chapter 4 (see also (**?**)).

We say that an OOFOL theory T is *totally ordered* if the objects in T can be sorted into a chain

$\langle Obj_1, ..., Obj_n \rangle$ such that every object Obj_i in the chain is directionally linked (or simply *linked* in the FOL-semantics case) to its predecessor Obj_{i-1} and there are no other links (i.e., no cycles in the graph of objects).

Applying our original intuition from Section 2.3, the following algorithm performs inference in one object, passes results to the appropriate objects and repeats the process in subsequent objects. Various inference methods can be used to prove a given query in a given object, but for simplicity we assume forward reasoning and a *totally ordered* theory T.

For two objects $Obj = Object(C, N, Link), Obj' = Object(C', N', L')$, let $\mathcal{L}_{Obj,Obj'}$ be the language that includes the set of *linking symbols* of Obj, Obj', $\{P \mid L'(Q, N, P)$ for some $Q\}$ with equality. For every sentence φ in $\mathcal{L}_{Obj,Obj'}$, we say that the *translation of φ from Obj to Obj'* is the sentence resulting from following L' in replacing every symbol $N.P$ in φ by $N'.Q$ such that $L'(Q, N, P)$. $\mathcal{L}_{=}$ is the first-order language with the relation "=" and no other relation, function or constant symbols. Figure 2.6 describes our message-passing algorithm. Theorem 2.4.2 validates our approach for the FOL semantics.

PROCEDURE MESSAGE-PASSING
Given a totally ordered OOFOL theory $T = \{Obj_i\}_{i \leq m}$ and a query Q whose symbols are from $L(Obj_n)$ (for some $n \leq m$),

1. Concurrently perform consequence finding for each of the objects $Obj_i \in T$ with $i \leq m$.
2. For each $i \leq n$, when we prove a formula $\varphi \in \mathcal{L}_{Obj_i,Obj_{i+1}}$, add the translation of φ from Obj_i to Obj_{i+1} to the set of axioms of Obj_{i+1}.
3. For each $n < i \leq m$, when we prove $Obj_i \models \varphi$, for $\varphi \in \mathcal{L}_{=}$, add φ to the set of axioms of Obj_n.
4. If we proved Q in Obj_n, return YES.

Figure 2.6: An algorithm for proving Q from a (Directed) OOFOL theory T by message-passing.

Theorem 2.4.2 (Completeness and Soundness for OOFOL) *Let T be a totally ordered OOFOL theory, $\{Obj_i\}_{i \leq n}$. Let $\varphi \in \mathcal{L}(Obj_n)$ be a sentence. $T \models \varphi$ if and only if the MESSAGE-PASSING algorithm outputs YES on φ given T.*

This theorem and its proof are special cases of Corollary 4.2.8 in Chapter 4. It is brought here for completeness of this chapter.

2.5 Directional Influence Between Objects

The previous sections gave semantics and reasoning procedures that are sufficient to represent an object-oriented system that has no direction associated with the links in the objects graph. From a procedural perspective, this approach corresponds to allowing influence to flow in both directions of a link. In Figure 2.2, if we proved $\mathcal{C}up1.has_liquid$, then we can infer $\mathcal{L}iquid1.in_cup$ and vice versa.

Sometimes, we would like to have the influence go in only one direction. For example, we may want to allow different accuracies of reasoning in two connected objects $\mathcal{O}_1, \mathcal{O}_2$. *Accuracy* here means that although something of interest to $\mathcal{O}_1$ may come up in $\mathcal{O}_2$, we prefer to have $\mathcal{O}_1$ ignore it. Abstractions and approximations are examples of such differences in accuracy. This ability to ignore influence is of particular importance in time-dependent tasks.

In this section we extend the methods provided above for the simple FOL case (providing bi-directional connections) to a system that can account for directionality of influence between the objects. We provide the semantics, example uses and inference algorithms for this extended object-oriented system.

2.5.1 Example Applications: Hierarchical Theories and Action Theories

Hierarchical Reasoning

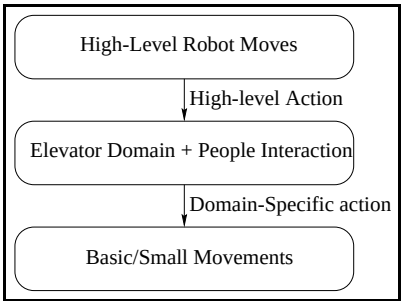

Figure 2.7: A hierarchical decomposition.

Suppose that a robot is given the assignment of navigating from one office to another. Since the robot's moves are miniscule and are supposed to be run and monitored in real-time, reasoning about the entire plan can be extremely expensive. Adding the requirement that the robot should reason about the behavior of other objects/agents (e.g., the elevator, doors, etc) makes brute-force

reasoning intractable and knowledge modeling difficult.

With Directed OOFOL one can represent and reason about several layers of abstraction concurrently (similar to Brooks' Subsumption architecture (Brooks, 1986)), leading to the desired behavior at the low-level sensors/actuators. To reason about high-level actions, the robot does not need to reason about the different ways these actions may be realized by lower layers. Figure 2.7 describes the theory. A more detailed description of a use of a similar theory is given in (Amir and Maynard-Reid II, 1999).

Action Theories: PMON and Multiple Agents

The temporal logic PMON (Sandewall, 1994; Gustafsson and Doherty, 1996) uses the *features and fluents* language $\mathcal{L}(\mathcal{FL})$ and has a theory divided into several subtheories: Γ_{OBS} (observations), Γ_{SCD} (schedule of events), Γ_{UNA} (unique names axioms), Γ_T (axiomatization of time) and Γ_{NCG} (no-change axiom).

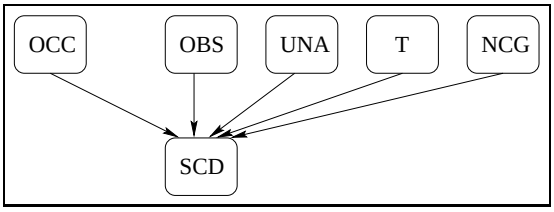

Figure 2.8: PMON in Directed OOFOL.

Different axioms are allowed in each category. We refer the reader to (Gustafsson and Doherty, 1996) for a relatively recent work on the subject. (Gustafsson and Doherty, 1996) showed that the semantics of PMON can be defined by

$$\Gamma_{NCG} \wedge \Gamma_{OBS} \wedge \Gamma_{SCD} \wedge \Gamma_{UNA} \wedge \Gamma_T \wedge Circ[\Gamma_{SCD}(Occlude); Occlude;].$$

Thus, we can represent PMON using the Directed OOFOL theory displayed schematically in Figure 2.8. In this figure, *OCC* is an object including the axiom $\forall x.Occlude(x)$. The rest of the objects represent their respective theories.

When we want to reason about several agents, decomposing the domain according to the different agents and their interactions can yield a significant gain in modeling effort and reasoning efficiency.

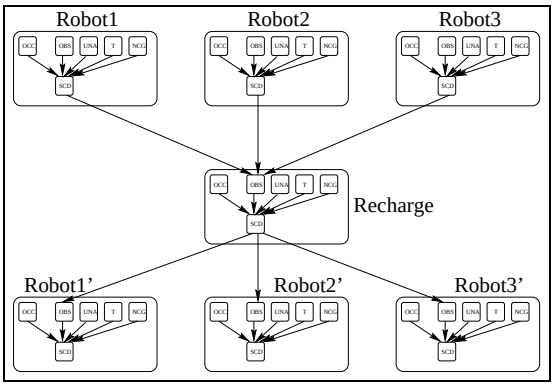

Figure 2.9: Multiple Agents.

For example, assume that there are three robots that need to perform independent tasks but should interact every hour to recharge their batteries. We can create a class **Robot** that includes the action theory for that robot and a class **Recharge** that includes the action theory for the multiple robots involved. Each object $\mathcal{R}ob1, \mathcal{R}ob2, \mathcal{R}ob3, \mathcal{R}ob1', \mathcal{R}ob2', \mathcal{R}ob3'$ is of class **Robot**. An action schedule can be given to each of the robots, and the results of executing these schedules are fed into the $\mathcal{R}echarge$ object (of class **Recharge**). The results of the recharging process are then fed into the separate schedules, and the robots may continue in their separate tasks. The complete picture is depicted in Figure 2.9. A more detailed examination of the use of object-oriented techniques for action theories that are written using situation calculus is given in Chapter 3.

2.5.2 Directional Semantics

To give directional semantics to OOFOL theories, we convert the equivalence/equality axioms that we added in Section 2.4.3 into defaults. This is done simply by adding *abnormality predicates* in front of each link equality/equivalence. We later *Circumscribe* these abnormalities. The circumscription will be done for each pair of linked objects separately. Thus, we write the corresponding formula to (2.6) for each pair separately:

$$\begin{aligned}\widetilde{\mathcal{A}}(\mathcal{O}_1, \mathcal{O}_2) = \{(\mathcal{A}_{C_i})_{[P/N_i.P|P\in L_{C_i}]} \mid i = 1, 2\} \bigcup \\ \{\forall \overrightarrow{x}\,(Ab_{N_1.P_1,N_2.P_2}(\overrightarrow{x}) \Rightarrow \mathcal{N}_1.P_1(\overrightarrow{x}) \equiv \mathcal{N}_2.P_2(\overrightarrow{x}))\ |Link_2(P_2, \mathcal{N}_1, P_1)\}\end{aligned} \tag{2.7}$$

where $\mathcal{O}_i = Object(\mathbf{C}_i, \mathcal{N}_i, Link_i)$.

We use Circumscription (McCarthy, 1986a) for our semantics. Recall that Circumscription is a

method for nonmonotonic reasoning in which, given a set of axioms T, the extensions of some predicate symbols $\overrightarrow{P}$ are minimized while some other symbols $\overrightarrow{Q}$ are allowed to vary their semantics. This is achieved by replacing T with the second-order formula (or FOL schema) $Circ[T; \overrightarrow{P}; \overrightarrow{Q}]$:

$$T(\overrightarrow{P}, \overrightarrow{Q}) \wedge \forall \overrightarrow{p}, \overrightarrow{q}\,[T(\overrightarrow{p}, \overrightarrow{q}) \wedge \overrightarrow{p} \leq \overrightarrow{P} \Rightarrow \overrightarrow{p} = \overrightarrow{P}].$$

In the following semantics, to have influence flow in only one direction, we circumscribe the abnormality predicates Ab_{O_1,O_2} (the set of all abnormality predicates in $\widetilde{\mathcal{A}}(O_1, O_2)$) in the axioms of the object-pair, $\widetilde{\mathcal{A}}(O_1, O_2)$. We allow only the symbols of O_2 to vary, thus allowing influence to flow from O_1 to O_2, but not vice versa.

Definition 2.5.1 (Directional Semantics) $\mathcal{M}$ is a Directed OOFOL model of an OOFOL theory T, *written* $M \models_D T$, *iff*

$$M \models \bigwedge_{\mathcal{O}_1, \mathcal{O}_2 \in T} Circ[\widetilde{\mathcal{A}}(\mathcal{O}_1, \mathcal{O}_2); Ab_{\mathcal{O}_1,\mathcal{O}_2}; L(\mathcal{O}_2)]$$

We define $M \models_D \varphi$ and $T \models_D \varphi$, for a formula $\varphi \in \mathcal{L}(T)$, as usual.

Roughly speaking, in this definition, the links of $\mathcal{O}_2$ serve as *inputs* and the links of $\mathcal{O}_1$ serve as *outputs*. This distinction is made precise by allowing the circumscription to vary only $L(\mathcal{O}_2)$, holding $L(\mathcal{O}_1)$.

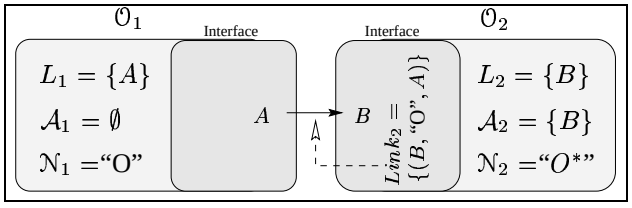

Figure 2.10: Influence flows from $\mathcal{O}_1$ to $\mathcal{O}_2$.

EXAMPLE Look at Figure 2.10. We are going to show that influence between the objects $\mathcal{O}_1$ and $\mathcal{O}_2$ flows only in the arrow's direction. As before, the link $Link_2(B, "O", A)$ roughly says "link my B to O's A". Proving $\mathcal{O}_2.B$ does not allow us to prove $\mathcal{O}_1.A$. Formally,

$$Circ[B \wedge (\neg ab \Rightarrow (B \iff A)); ab; B] \not\models A.$$

This is because the only models (in the propositional language $\{A, B\}$) that satisfy $B \wedge (\neg ab \Rightarrow (B \iff A))$ are $M_1 = \{\neg A, B, ab\}, M_2 = \{A, B, \neg ab\}$, $M_3 = \{A, B, ab\}$. M_1, M_2 are not

comparable by the preference relation of our circumscription because they differ on A (which is held constant in our circumscription). M_2 is preferred to M_3, and thus we have two minimal models of our circumscription, one of which does not satisfy A.

2.5.3 Reasoning Algorithms for Directional OOFOL

The message-passing algorithm described in Figure 2.6 on page 25 is the same algorithm that we can use for the directional semantics. We use it in a modified way to account for this semantics. Instead of sending messages from all the objects in the direction of the partition of the query, we send messages only in those directions instructed by the directed graph associated with the directional OOFOL theory. In the totally ordered case, this implies that a query in object $\mathcal{O}_k$ would receive messages only from $\mathcal{O}_{k-1}$ and would not be influenced by $\mathcal{O}_{k+1}$.

To show that this is a sound and complete algorithm for reasoning with Directed OOFOL theories, we first give an equivalent semantics to our directional semantics using *prioritized circumscription* for totally ordered theories. Define $flat(T) \stackrel{def}{=} \bigcup_{O_i,O_j \in T}(\widetilde{\mathcal{A}}(O_i, O_j))$.

Theorem 2.5.2 (Equivalent Directional Semantics) *Let $T = \{Obj_i\}_{i \leq n}$ be an OOFOL theory that is totally ordered. $\mathcal{M}$* is a model of *T iff it is a model of*

$$\bigwedge_{i \leq n} Circ[flat(T); Ab_i; Ab_{i+1}, ..., Ab_n, L(Obj_{i+1}), ..., L(Obj_n)]$$

where $Ab_i = Ab_{O_i,O_{i+1}}$.

PROOF See Appendix A.1.1.

Note that the formula above can be written as a single Prioritized Circumscription formula after the symbols in $L(Obj_{i+1}), ..., L(Obj_n)$ are eliminated from the varied part (Cadoli et al., 1992).

Theorems 2.5.3 and 2.5.5 validate our approach to the directional semantics.

Theorem 2.5.3 (Completeness for Directional Semantics) *Let $T = \{Obj_i\}_{i \leq m}$ be a totally ordered OOFOL theory, with $Obj_i = \langle C_i, N_i, Link_i \rangle$. Let $n \leq m$, $\varphi \in \mathcal{L}(Obj_n)$ be a sentence and $E = \{\psi \in \mathcal{L}_= \mid \exists i > n\ Obj_i \models \psi\}$. If $T \models_D \varphi$ and the circumscriptions involved in the semantics of T are* smooth[2]*, then there is $a \leq n$ and a sequence of sentences $\varphi_a, \ldots, \varphi_n$ such that $E_n \cup \{\varphi_n\} \models \varphi$, $Obj_a \models \varphi_a$ and for every i such that $a \leq i < n$, $\varphi_i \in \mathcal{L}_{Obj_i,Obj_{i+1}}$ and $Obj_{i+1} \models \varphi_i' \Rightarrow \varphi_{i+1}$, where φ_i' is the translation of φ_i from Obj_i to Obj_{i+1}.*

[2]The circumscription is *smooth* if every model either is minimal or has a minimal model that is preferred over it. See (Kraus et al., 1990) for a more formal definition of *smoothness* of a preferential order.

PROOF See Appendix A.1.2.

Corollary 2.5.4 (Completeness for Directional Semantics) *Let T be a totally ordered OOFOL theory, $\{Obj_i\}_{i\leq m}$, and $\varphi \in \mathcal{L}(Obj_n)$ be a sentence ($n \leq m$). If $T \models_D \varphi$, then MESSAGE-PASSING will output YES.*

For soundness, we need to assume that influence does not "try" to flow back (i.e., if $O_{i+1} \models \varphi'$ for some $\varphi \in \mathcal{L}_{O_i,O_{i+1}}$, then $T \setminus \{O_j \mid j \leq i\} \models \varphi$).

Theorem 2.5.5 (Soundness for Directional Semantics) *Assume that there are $\varphi_k, ..., \varphi_l$ such that $k \leq l$ and $O_k \models \varphi_k$ and $k \leq \forall i < l$, $\varphi_i \in \mathcal{L}_{O_i,O_{i+1}}$ and $O_i \models \varphi_i' \Rightarrow \varphi_{i+1}$, where φ_i' is the translation of φ_i from O_i to O_{i+1}. Then, $T \models \varphi_l$.*

PROOF Since there is no attempt for influence back, we know by lemma A.1.4 that $T \models \forall x.\neg ab(x)$ for all the abnormality predicates ab in our system. We get that the soundness follows from that of entailment in FOL. ■

The ways the two semantics (FOL and directional) use MESSAGE-PASSING differ in two ways: (1) for FOL Semantics we can ask queries only from the last object in the sequence, Obj_m; and (2) MESSAGE-PASSING is not sound for Directional Semantics if influence "tries" to flow back, in the sense described above. This is because when there is such influence, the semantics sanctions that it may block influence flow forward (i.e., some of the ab's become TRUE), while there is no such block in our algorithm (we always transfer axioms forward).

It is illuminating to notice the close relationship between our assumption above and the soundness of inference. If we weaken the assumption to merely state that T is consistent with $\forall x.\neg ab(x)$, we immediately find a counter-example to soundness. Take $O_1 \models P$ and $O_2 \models \neg P' \vee \neg Q'$, where P, P' are linked by ab_P and Q, Q' are linked by ab_Q. Our algorithm will pass P from O_1 to O_2, and we will conclude $\neg Q'$ in O_2. On the other hand, according to our directional semantics, $\neg Q'$ does not follow from $T = \{O_1, O_2\}$, since there is a minimal model $\mathcal{M}$ of T in which $\mathcal{M} \models P \wedge Q \wedge \neg P' \wedge Q'$.

2.6 Related Work: Object-Oriented KR Systems

Many object-oriented knowledge representation systems are relevant to this work. However, systems available up until OOFOL (**?**) lack expressivity, lack object-oriented tools, extend the expressivity beyond the simple FOL semantics or are not concerned with knowledge engineering at all.

We include here only the most relevant, and others (e.g., communication protocols such as KQML and OKBC (Chaudhri et al., 1998)) are avoided in this comparison.

The seemingly closest approach to ours is the work on object-oriented Prolog (O-O Prolog) (e.g., (Shapiro and Takeuchi, 1983; Moss, 1994)). Obviously, full FOL expressivity is different from Horn clauses with the Closed World Assumption. More interestingly, Prolog and FOL raise different challenges for object-oriented approaches. O-O Prolog structures are built using a programming perspective, whereas OOFOL structures are built emphasizing a representation perspective. Also, the program flow in O-O Prolog is identical to the program flow in ordinary Prolog. In contrast, the object-oriented structures in OOFOL present guidelines for *theorem proving flow* that do not exist in FOL. The object structure gives heuristic information to the theorem prover on how to aggregate computation and what information should flow from object to object. Such information is not used in O-O Prologs.

An approach to the formulation and construction of object-oriented *temporal action logic* (TAL) theories (Doherty et al., 1998) was proposed in (Gustafsson and Kvarnstrom, 2001), subsequent to our work. It has more expressivity than our approach in that it considers classes and objects as first-order elements in the universe. This allows much more expressivity from a design point of view, but it is not clear how this expressivity can be tamed in a general FOL theory. For the case of TAL theories they show that some object-oriented theories can be reduced to TAL and therefore can be used with relative computational efficiency. Also, this approach does not use interfaces for objects, and encapsulation of theories is lost. It is somewhat reminiscent of some formalizations of context (see mention below).

Another close approach is Frame Systems. Currently, there is no adequate object-oriented approach for full FOL in Frame systems. KRYPTON / KL-ONE (Brachman et al., 1983), one of the early Frame systems, had an A-Box part that allowed arbitrary FOL sentences to be stated. This structure was completely flat and had nothing of the object structure that we gave to our OOFOL. Ontolingua (Gruber, 1993; Fikes et al., 1997), which is one of the most expressive Frame systems to date, allows different sentences to be associated with each frame. Unlike in OOFOL, there is no difference between a sentence put in one frame and the same sentence put in a different frame in Ontolingua (Fikes, 1997). In particular, there is no “scope” in which the axiom is active/inactive. The situation is similar in other Frame systems such as (Clark and Porter, 1997) (for a recent survey see (Fridman-Noy and Hafner, 1997)). Frame systems also traditionally use only a restricted form of the axiom inheritance used in this chapter (for the A-Box part of these systems). *Overriding* is achieved (if at all) using nonmonotonic techniques.

The field of *description logics* (e.g., (Borgida, 1996; Donini et al., 1991)) grew out of the interest in generalizing Frame systems and giving them clear semantics and tractable algorithms. Subsets of FOL that are computationally tractable are discussed in the context of describing objects. In contrast, our approach to describing objects is not restricted to relations that include this object and those related to it. We follow the object-oriented approach devised by programming languages in which objects serve as landmarks around which one builds the application/theory. This results in a different approach to object-oriented design from that applied in description logics, together with unlimited FOL expressivity.

Object-Oriented Databases are conceptually different from our approach. (Dietrich et al., 1992) is an example of such an approach. It takes a class to be a schema for a relation in a relational database. This approach has naturally limited expressivity. Logics for Object-Oriented Databases (e.g., (Kifer et al., 1995), (Lausen and Vossen, 1998)) try to give specific semantics to object-oriented databases. They take the approach that objects and classes are elements of specific sorts in our universe and create a specific language that allows us to reason about object membership in a class, class inheritance and properties of objects. In contrast, we do not try to reason about objects, but rather use the object-oriented technology to its purpose, much in the sense of programming languages, allowing modularity and distributed inference.

Compared to the approaches for formalizing context (e.g., McCarthy's (McCarthy, 1993), CycL (Lenat and Guha, 1990; Guha and Lenat, 1990; Guha, 1991; Lenat, 1995) and QLC (Buvač, 1996)), our logic allows only prescribed connections between the different objects (contexts usually have no restrictions), and instead of appealing to modalities for representing the different theories (or other special-purpose semantics, as in (Giunchiglia and Ghidini, 1998; Ghidini and Serafini, 1998; Giunchiglia and Serafini, 1994) (using local-models semantics for the undirected case and proof-theoretic semantics for the directed case)), we use a simple FOL semantics for the undirected case and Circumscriptive/Preferential semantics for the directional case. Our semantics also has one universe of elements, whereas contexts are usually considered to have distinct domains of elements. For OOFOL we devised much more efficient general-purpose algorithms than those available for context systems (see (Lenat and Guha, 1990; Amir and McIlraith, 2000)).

Finally, Bayes Nets (Pearl, 1988) and Object-Oriented Bayes Nets (Koller and Pfeffer, 1997) are closely related in spirit to our development here. Some of the ideas presented here arose from our observations of the work done in these fields. The similarity in spirit suggests future combination of such theories. Such a combination seems especially interesting because it would allow the merging of the two representational paradigms of logic and probabilities.

2.7 Summary and Discussion

Object-Oriented Design is a proven method in software engineering. Its benefits range from ease of change and composition, to the aggregation of related information into the same place, information encapsulation and hiding. These benefits make software construction a much less tedious process, allowing software to be larger and more robust.

This chapter presented an object-oriented approach to FOL. We showed that this approach can be captured by a simple FOL semantics in the undirected case and a simple circumscriptive semantics in the directional case. These semantics are especially simple compared to previous approaches to decomposing logical theories into small fragments. This chapter demonstrated several applications of our approach and showed how object-oriented design can be facilitated by the tools we provided. Finally, we described efficient inference algorithms for both logics. No other approach to date describes the application of object-oriented design for general FOL theories. Other approaches to logic emerged after this work was published to supply the tools needed for object-oriented design without abandoning the original FOL semantics.

Compared to unstructured FOL, the object-oriented constructs encourage the knowledge engineer to aggregate knowledge for a particular use in one place, hiding most of the details of the implementation from the outside world by the provision of minimal interfaces. These interfaces limit the impact of internal-object changes on the rest of the theory. To allow for knowledge reuse, information integration and better knowledge engineering, FOL needs to be used with a structured methodology. In this work we demonstrated one such methodology and provided the tools needed for its application.

The expressivity of OOFOL is equivalent to that of FOL, but the expressivity of its object language (the language governing the construction of objects from classes and the relationships between objects) is limited. This is a drawback that the system of (Gustafsson and Kvarnstrom, 2001) improves upon. Such expressive power may sometimes be needed if we want to express complex relationships between objects. However, it is not clear how such an expressive system would behave computationally. This is a topic of ongoing research.

Knowledge-representation languages that have contributed some intuitions to this work and are related to object-oriented designs are (Kifer et al., 1995; Koller and Pfeffer, 1997; Lausen and Vossen, 1998; Maier, 1986). An application of this work to situation calculus (McCarthy and Hayes, 1969) appears in the next chapter (Chapter 3).

Chapter 3

Object-Oriented Design of Commonsense Theories

"Common sense is not so common."
Voltaire

3.1 Introduction

This chapter has two related motivations: making large situation calculus theories easier to design and maintain, and giving a partitioned representation to situation calculus theories. Situation calculus theories have a restricted form and a clear ontology that together help modeling. However, using situation calculus in large, multi-domain settings is not easy. It is hard to think of all the parts of a large theory in advance, reuse is not simple, and modifications can have unintended consequences. Giving situation calculus theories a partitioned representation allows us to use the algorithms of Chapter 4 and opens the way for merging situation calculus theories into reactive frameworks such as presented in Chapter 6. This chapter focuses on the first motivation and addresses the second as a by-product.

We present one object-oriented reformulation of situation calculus, following some traditional decompositions of theories of action (Shanahan, 1997). We create a class for situations and a class for actions (recall, *classes* are theory fragments, roughly speaking). *Subclasses* of class **Situation** (recall, these are supersets of the class of situations, roughly speaking) contain domain constraints. Subclasses of class **Action** contain effect axioms. We describe one theory constructed using objects

of these and other classes. The combined theory has semantics that is equivalent to situation calculus, after translation to FOL. We apply Reiter's monotonic solution to the frame problem (Reiter, 1991) to this system of objects and classes.

Object-oriented situation calculus theories can be combined easily and are elaboration-tolerant in that sense. We demonstrate this by combining domain theories and by giving a new treatment of agents in the situation calculus. Domain theories are connected by linking the corresponding objects with only a small vocabulary, even when the frame solution is used. This result extends to our proposed representation of agents. Our treatment of agents allows each agent's theory to be written as a simple theory, with no regard to the existence of other agents. Only new subtheories in which the agents interact need to consider the explicit existence of agents. We do this without giving up the *result* formalism (which is planning-friendly), as in (McCarthy and Costello, 1998), or the independence of the agents (which allows efficient computation), as in (Reiter, 1996), (Pinto, 1998). We point out the applications and influence of the algorithms of Chapter 4 on these theories. We conclude by proposing some guidelines for the knowledge engineering of situation calculus theories.

Our running example is a simplification of the *Daddy and Junior* scenario, taken from (McCarthy and Costello, 1998). The results in this chapter appeared previously in (Amir, 2000a).

3.2 Objects for Situations and Actions

3.2.1 Situation Calculus

Situation calculus (McCarthy and Hayes, 1969) is a knowledge representation formalism for representing temporal information. The language consists of four sorts: *situations*, for situations in the world; *actions*, for events and actions; *fluents*, for situation-dependent properties; and *objects*, for simple other FOL objects.

s, a, f (or subscripted versions thereof) are variables for situations, actions and fluents, respectively. All other variables are of sort "object." The predicate $Holds(f, s)$ asserts that a fluent f holds in the situation s (note that $f(s)$ is used as a shorthand for $Holds(f, s)$ when such a shorthand causes no confusion). The function $result(a, s)$ returns the situation that results from performing a in s (we use $res(a, s)$ as a shorthand). $S0$ is a situation constant. Figure 3.1 displays A_{bw}, a sample situation calculus theory for the blocks-world domain. Capitalized symbols are constants. Free variables in axioms are universally quantified with maximum scope.

$$
\begin{array}{l}
on(A, B, S0) \wedge on(B, Table, S0) \wedge on(C, Table, S0) \\
on(x, y, s) \Rightarrow above(x, y, s) \\
above(x, y, s) \wedge above(y, z, s) \Rightarrow above(x, z, s) \\
clear(b, s) \iff \forall b' \neg on(b', b, s) \\
handEmpty(s) \iff \forall b \neg inHand(b, s) \\
clear(b, s) \wedge handEmpty(s) \Rightarrow inHand(b, res(pickUp(b), s))) \\
(y = Table \vee clear(y, s)) \wedge inHand(x, s) \Rightarrow \\
\qquad\qquad on(x, y, res(putOn(x, y), s)) \\
on(x, y, s) \Rightarrow \neg inHand(x, s) \wedge \neg inHand(y, s) \\
inHand(x, s) \Rightarrow \neg on(x, y, s) \wedge \neg on(y, x, s) \\
on(x, y, s) \wedge y \neq z \Rightarrow \neg on(x, z, s)
\end{array}
$$

Figure 3.1: Blocks-world in the situation calculus: A_{bw}.

3.2.2 Object-Oriented FOL

We wish to make the design of situation calculus theories more structured. To this end, we adopt some object-oriented design principles and tools. In the rest of this section, we build an object-oriented situation calculus theory for the blocks-world.

Recall, that an *object-oriented first-order logic (OOFOL) theory* is divided into subtheories associated with *objects*. A subtheory associated with an object has its own first-order language and axioms. We distinguish a subset of the vocabulary of each object and call it the object's *interface*. There are interface *links* between objects. Each interface link specifies equality/equivalence between symbols in two objects. Only symbols from these interfaces may participate in the link.

A situation calculus theory can be broken into an object (subtheory) associated with situations and an object associated with actions (additional other objects are possible). In the first, we put domain constraints (sentences that mention no action term). In the second, we put effect axioms (axioms of the form $Holds(\Phi, s) \Rightarrow Holds(\Psi, res(a, s))$). This decomposition for our blocks-world theory A_{bw} is diagrammatically presented in Figure 3.2. Here, the symbols $on, inHand, Holds$ are on the link between the objects and thus have the same semantics in both objects.

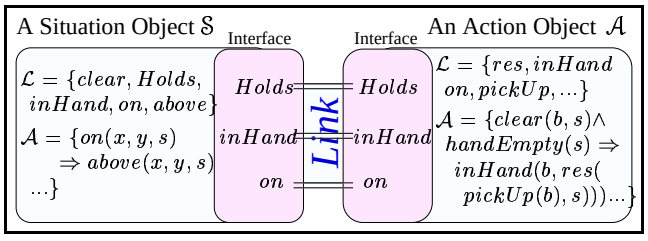

Figure 3.2: A situation object connected to an action object.

Recall, that an OOFOL theory T is a set of object declarations (see below). The semantics is given by a translation to FOL: We replace every symbol in an axiom with the same symbol appended to the name of the object in which this axiom appears (making symbols in different objects distinct). For example, the symbol on used in a situation object $\mathcal{S}$ is translated to $\mathcal{S}.on$ in all the axioms of $\mathcal{S}$. Then, we add equality (or equivalence) axioms of the form $\forall \overrightarrow{x}(\mathcal{S}.P(\overrightarrow{x}) \equiv \mathcal{A}.P'(\overrightarrow{x}))$ for all linked symbols P, P' between objects $\mathcal{S}, \mathcal{A}$. The result of this translations is denoted $\widetilde{T}$.

3.2.3 Situation Calculus Classes

We can now represent a theory as a set of linked objects. Presenting situation calculus theories in this form gives a designer a high-level, abstract view of the theory and allows the combination of several theories by *linking* them. We also wish to reuse theories. For this, the notions of *classes* and *inheritance*[1] become handy. A class is a template for its objects, specified with no interface links, but otherwise identical to them. This allows the application of identical axiom sets in different positions in the theory by specifying the class to which the object belongs.

Inheritance extends this reuse by allowing some extensions to a copy of an existing class. Recall, that inheritance in our system is treated outside the logic (unlike nonmonotonic inheritance, such as in (Morgenstern, 1998)), for engineering purposes. In principle, a *subclass* (child) inherits all the axioms, vocabulary and interface vocabulary of its parent class. We can make the subclass different from its parent by: (1) adding to the inherited vocabulary and interface vocabulary; (2) adding axioms; and (3) explicitly replacing some non-logical symbols in all of the inherited axioms with some new symbols in the child's language.

We use standard object-oriented methodology and define a hierarchy of classes for the blocks-world situation calculus. Figure 3.3 displays our classes. There are two root classes: **Situation** and **Action**. Both include interface declarations (to be inherited by subclasses) and no axioms. The notation *class1:class2* is used to declare that class1 is a subclass of class2. Classes **BWSit**,**BWAct** are subclasses of **Situation**,**Action**, respectively, specialized for the blocks-world. They inherit the respective interfaces and add it to their explicitly declared interface. Since **BWS0** is a subclass of **BWSit**, it includes all the interface and axioms of its superclass, adding some observations about a particular $S0$.

[1] *Objects*, *classes* and *inheritance* are meta-logical notions.

class Situation {
  interface: Holds }

class Action {
  interface: res, Holds }

class BWSit : Situation {
  interface: $on, inHand$
  axioms: $clear(b,s) \iff \forall b' \neg on(b',b,s)$
  $on(x,y,s) \Rightarrow above(x,y,s)$
  $above(x,y,s) \wedge above(y,z,s) \Rightarrow above(x,z,s)$
  $on(x,y,s) \Rightarrow \neg inHand(x,s) \wedge \neg inHand(y,s)$
  $inHand(x,s) \Rightarrow \neg on(x,y,s) \wedge \neg on(y,x,s)$
  $on(x,y,s) \wedge y \neq z \Rightarrow \neg on(x,z,s)$ }

class BWAct : Action {
  interface: $on, inHand, pickUp, putOn, Table$
  axioms: $clear(b,s) \wedge handEmpty(s) \Rightarrow$
  $inHand(b, res(pickUp(b), s)))$
  $(y = Table \vee clear(y,s)) \wedge inHand(x,s) \Rightarrow$
  $on(x,y,res(putOn(x,y),s))$
  $clear(b,s) \iff \forall b' \neg on(b',b,s)$
  $handEmpty(s) \iff \forall b \neg inHand(b,s)$ }

class BWS0 : BWSit {
  interface: $on, S0, A, B, C, Table$
  axioms: $on(A,B,S0) \wedge on(B,Table,S0) \wedge on(C,Table,S0)$ }

Figure 3.3: Blocks-world situation calculus classes.

3.2.4 Modeling With One Object per Class

Using these classes, we create the theory $\mathcal{A}_{ob}$, shown in Figure 3.4. In this theory, $\mathcal{S}0_{obj}$ is an object (instantiation) of class **BWS0**, $\mathcal{S}$ is an object of class **BWSit** and $\mathcal{A}$ is an object of class **BWAct**. The statement $Object(\mathbf{Cl}, \mathcal{O}, \{(P, \mathcal{O}', Q), ...\})$ declares an object named $\mathcal{O}$ of class **Cl** and specifies interface links with other objects: P in $\mathcal{O}$ is equal/equivalent to Q in $\mathcal{O}'$ (e.g., $\mathcal{O}.P \equiv \mathcal{O}'.Q$). $\mathcal{S}0_{obj}, \mathcal{S}$ are linked on $on, Holds$; $\mathcal{S}, \mathcal{A}$ are linked on $on, inHand, Holds$; and $\mathcal{A}, \mathcal{S}0_{obj}$ are linked on $Table$.

Our modeling can be summarized by the following two rules: (1) link all those symbols that should have the same intended meaning in two objects; and (2) omit links of symbols that follow from transitivity of equality (e.g., we omit $Holds, on$ from the link among $\mathcal{A}$ and $\mathcal{S}0_{obj}$).

Figure 3.5 is a diagrammatic view of the structure of $\mathcal{A}_{ob}$ showing the three objects and the links between them. Each link is labeled with the symbols that are pronounced equal between the two objects (e.g., for the fluent $on(x,y)$, $\forall xy\mathcal{S}.on(x,y) = \mathcal{A}.on(x,y)$). Almost all the examples we will see have identical symbols on both ends of a link (e.g., on in $\mathcal{S}$ and on in $\mathcal{A}$), so this convention

$Object(\textbf{BWS0}, \mathbb{S}0_{obj}, \{(on, \mathbb{S}, on), (Holds, \mathbb{S}, Holds)\}).$
$Object(\textbf{BWSit}, \mathbb{S}, \{(inHand, \mathcal{A}, inHand),$
$\quad (on, \mathcal{A}, on), (Holds, \mathcal{A}, Holds)\}).$
$Object(\textbf{BWAct}, \mathcal{A}, \{(Table, \mathbb{S}0_{obj}, Table)\}).$

Figure 3.4: The OOFOL theory $\mathcal{A}_{ob}$

is clear.

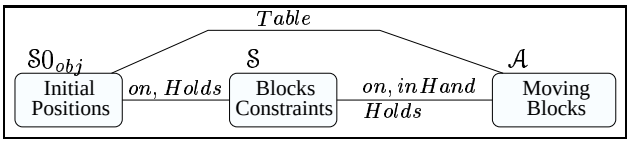

Figure 3.5: Diagrammatic view of $\mathcal{A}_{ob}$

The OOFOL semantics (Definition 2.4.1) given to $\mathcal{A}_{ob}$ is equivalent to the semantics given to A_{bw} from Figure 3.1. The translation of $\mathcal{A}_{ob}$ into $\widetilde{\mathcal{A}}_{ob}$ is syntactic, adding equality/equivalence axioms between linked symbols.

Proposition 3.2.1 (Semantics) *Let $\varphi \in \mathcal{L}(A_{bw})$ and let φ' be its translation to $\mathcal{L}(\mathcal{A}_{ob})$. $A_{bw} \models \varphi$ iff $\mathcal{A}_{ob} \models \varphi'$.*

The translation of φ to $\mathcal{L}(\mathcal{A}_{ob})$ is straightforward. Symbols of φ that appear in more than one object have the same semantics in all of them, so we can arbitrarily pick one. For example, $A_{bw} \models \neg inHand(A, S0)$ iff $\mathcal{A}_{ob} \models \neg\mathbb{S}.inHand(\mathbb{S}0_{obj}.A, \mathbb{S}0_{obj}.S0)$.

3.2.5 Reasoning With Objects

Chapter 4 provides message-passing algorithms in the style of (Pearl, 1988) for reasoning with theories that are decomposed in this way. Roughly speaking, if object $\mathbb{S}$ is connected to object $\mathcal{A}$, sentence φ_1 from $\mathcal{L}(\mathbb{S})$ can be logically combined (after translation) with a sentence φ_2 from $\mathcal{L}(\mathcal{A})$ only if $\mathcal{L}(\varphi_1)$ is included in the link.

More generally, given a graph of links and objects (as in Figure 3.5) and a query, we first decide on an object that can express the query in its vocabulary (if there is no such object, we can create one and link it to the others). Then, we transform the graph into a tree by running an algorithm called BREAK-CYCLES (we iteratively select a minimal cycle in the graph, remove a link and add its language to that of the other links in the cycle) and then perform MESSAGE-PASSING

to prove the query, which roughly performs the following: Continuously perform consequence-generation in each object. Send a proved formula (message) to another object only if the formula is in the vocabulary of the link to that object, and this other object is on the graph path to the object containing the query.

Chapter 4 shows that this kind of algorithm is complete and sound for the semantics of Definition 2.4.1 and that it improves running time compared to standard deduction methods. For the propositional case, if T is a theory with n objects, each connected to d other objects in a tree structure, with each link having l symbols, and $\mathcal{L}(T)$ has m symbols, then the running time for an analogous algorithm for SAT is $O(n * 2^{d*l} * f_{SAT}(m/n))$ (f_{SAT} is the time to compute SAT). This is an exponential time-improvement compared to standard reasoning techniques if the links' vocabularies are small.

We can now explain our modeling decisions from a computational perspective. To increase efficiency, we try to minimize the vocabularies of the links without giving up the correctness of queries. This is why we omit the symbols $A, B, C, S0, res$ and several fluents from the links. Each symbol shows only in one object ($clear$ is defined by the fluent on in both $\mathcal{S}, \mathcal{A}$). This is also the reason why we use the transitivity of equality to omit symbols from the links.

For a given domain theory (e.g., our blocks-world theory), $Holds$ and fluents that appear in the action object typically appear on the link between the situation and action objects. This seems to limit the computational improvement due to such algorithms, and it seems to be an inherent property in theory structures of the form presented in Figure 3.5 (one object per class, connecting on $Holds$ and fluents). There are other structures that take advantage of an assumed tree of situations to give better performance (e.g., multiple objects, connected in a tree structure). We do not pursue these further here. Instead, our computational effort focuses on larger theories made of multiple domains, as described below.

3.3 The Frame Problem

The *frame problem* concerns the conclusion of non-effects of actions from the known effects in a concise, correct, expressive and elaboration-tolerant manner (see (Sandewall and Shoham, 1994; Shanahan, 1997) and the more recent (Levesque et al., 1998)).

One of the simplest solutions generates *explanation closure* axioms (e.g., (Haas, 1987; Pednault, 1989; Schubert, 1990)) from the effect axioms and the domain constraints. When it is possible, we mechanically (outside the logic) add axioms saying that if something has changed, then one of the

enumerated actions occurred. For example, one axiom that is generated for explanation closure for $inHand$ in A_{bw} is $\neg inHand(x,s) \wedge inHand(x, res(a,s)) \Rightarrow (clear(x,s) \wedge handEmpty(s) \wedge a = pickUp(x))$.

(Reiter, 1991) summarized the effort and showed how to generate such axioms automatically if there are no state constraints, (Lin and Reiter, 1994) extended this process for the presence of state constraints, using deduction, and (McIlraith, 2000) gave a closed-form solution in the presence of some restricted state constraints. These solutions also add other axiom sets, including unique names axioms (UNA) for sort "actions," preconditions for executing actions (summarized by the predicate $Poss(a,s)$) and foundational axioms for situations (here we call them *Peano axioms for situations*, although the time structure represented by situations is a tree rather than a line).

The diagram in Figure 3.6 presents the theory resulting from appending this solution to $\mathcal{A}_{ob}$, together with additional axioms for domain closure (DCA) and UNA for objects (these are not needed for the solution, but are sometimes assumed in domain theories). In this figure, $\mathcal{U}na_a$, $\mathcal{U}na_o\&Dca_o$, $\mathcal{P}oss$, $\mathcal{P}eano$ contain UNA for actions, UNA and DCA for objects, preconditions for execution and the Peano axioms, respectively. Our Peano axioms for situations include the first-order induction axiom $(\forall f)(Holds(f,S0) \wedge (\forall as)[Holds(f,s) \Rightarrow Holds(f, res(a,s))] \Rightarrow (\forall s) Holds(f,s))$ instead of the second-order axiom of (Lin and Reiter, 1994).

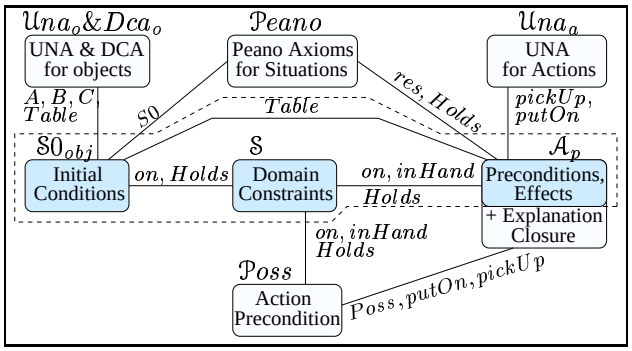

Figure 3.6: Frame-problem solution using objects.

The object $\mathcal{A}_p$ contains effect and explanation closure axioms with the following provision. To allow the introduction of the predicate $Poss(a,s)$, we need to make the effect axioms of **BWAct** dependent on $Poss(a,s)$. To do that, we create a subclass **BWPossAct** (shown in Figure 3.7) and replace the object $\mathcal{A}$ of the superclass with an object $\mathcal{A}_p$ of the subclass. The declaration *inherit:* $[Holds/Holds_{old}]$ is a directive to our compiler to replace the symbol $Holds$ with $Holds_{old}$ in all the axioms the class inherits from its superclass (as discussed above). This and the additional axiom

adjust our effect axioms to depend on $Poss$ (see Figure 3.7).

class BWPossAct : BWAct {
interface: $Poss$
inherit : $[Holds/Holds_{old}]$
axioms : $Poss(a,s) \wedge Holds_{old}(f, res(a,s)) \Rightarrow Holds(f, res(a,s))$
...[Explanation closure axioms]... }

Figure 3.7: Adding $Poss$ and explanation closure to **BWAct**.

3.4 Combining Action Theories

Now we examine the way domain theories can be joined. Consider a domain theory regarding buying and selling blocks. In this theory, buying a block decreases the amount of money a robot has, but also places the purchased block in the robot's hand. Selling a block has the opposite effect.

We use two new fluents together with $inHand$: (1) $hasItem$ accounts for named items the robot possesses; and (2) $money$ is a function (fluent) that returns the amount of money that the robot has. Figure 3.8 presents some of the new classes. Figure 3.9 displays the diagram of the complete theory with the frame solution.

class Has : Situation {
interface: $hasItem, money, inHand$
axioms: $money(s) \geq 0$
$inHand(b,s) \Rightarrow HasItem(b,s)$ }
class BuySell : Action {
interface: $hasItem, money, inHand$
axioms: $inHand(b,s) \wedge money(s) = m \Rightarrow$
$\neg hasItem(b, res(sell(b), s)) \wedge$
$money(res(sell(b), s)) = m + value(b,s)$ ... }

Figure 3.8: The domain of buying and selling blocks.

If the solution to the frame problem is not applied, we can combine the two domain theories by letting objects of corresponding positions in the two theories communicate. The two domains share only $S0, res, Holds$ and $inHand$. To see the situation diagrammatically, look at Figure 3.9 and consider only objects $\mathcal{S}, \mathcal{S}', \mathcal{A}, \mathcal{A}', \mathcal{S}0_{obj}, \mathcal{S}0'_{obj}$.

When we add the solution to the frame problem, there are two complications. First, the graph becomes more connected: UNAs potentially make all the objects dependent on one another and explanation closure axioms add bandwidth to the links. Second, if we already have the solution for

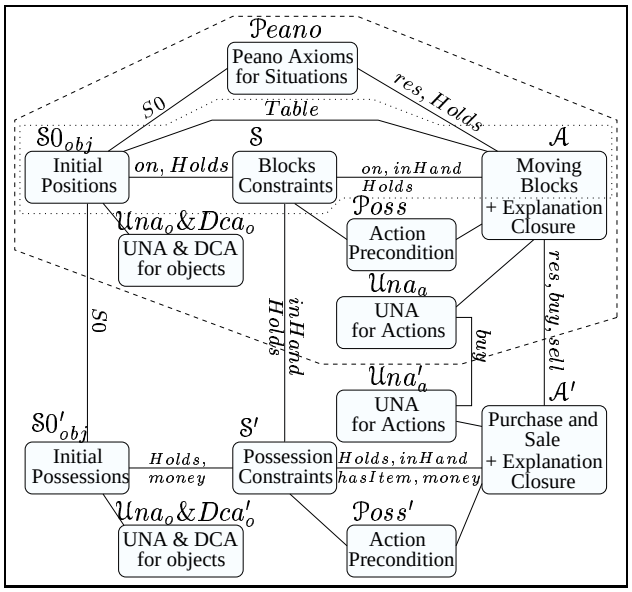

Figure 3.9: Connecting two domain theories.

the separate domain theories, there are components that will have to be recomputed. These concerns are particularly important if we have a large number of connected domains.

Proposition 3.4.1 *For a theory made of many domain theories, if no qualification constraints are involved in computing $Poss$, then there is a formulation of the frame solution above (with or without UNA and DCA for objects) such that*

1. *Any two domain theories that share fluents $\{f_i\}_{i\in I}$ are linked on exactly $\{f_i\}_{i\in I}$, $S0$, res, $Holds$ and actions $\{a_j\}_{j\in J}$ from the first domain that appear in the explanation closure for $\{f_i\}_{i\in I}$.*

2. *If we already have the frame solution for the separate domain theories, only the explanation closure axioms for $\{f_i\}_{i\in I}$ need to be recomputed (we ignore computations involved in solving the qualification problem (i.e., computing axioms for $Poss$)).*

3. *Domain theories are (directly) linked iff they share fluents or a removal of their links disconnects the graph.*

PROOF See Appendix A.2.1.

Thus, adding the frame solution on top of the simple connection between the two domains adds to the links only actions participating in explanation closure of shared fluents. Figure 3.9 displays the

complete theory of the blocks-world and buy-sell domains ($inHand$ is the only shared fluent). This result enables building composite theories like the one sketched in Figure 3.10 for a mobile robot that can perform electronic transactions, make phone calls, etc. Applying the reasoning algorithms described above to this theory results in an improved inference time, as the connectivity of this graph is kept low.

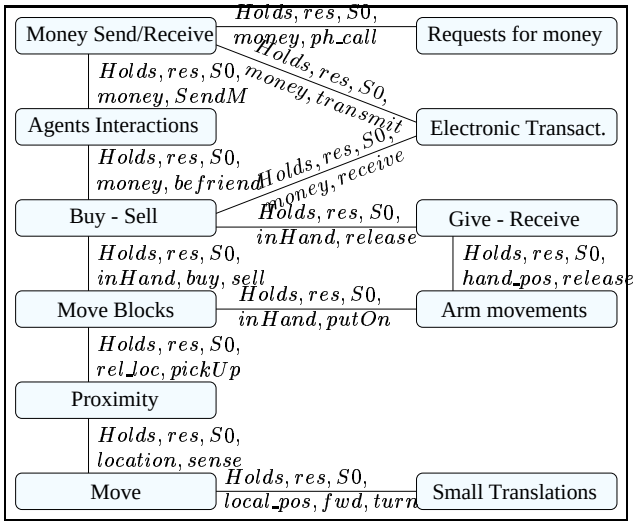

Figure 3.10: High-level diagrammatic view of a theory combining multiple domains.

3.5 Combining Different Agents

We can represent different agents in the situation calculus by giving them different objects in the theory and by adding an object to represent interactions. In the following, we assume that the agents' blocks-worlds do not interact, namely, that they deal with different fluents altogether in moving blocks and buying and selling blocks. We distinguish between actions that they perform separately (e.g., moving a block in their respective domains) and actions that influence the domains of both agents (e.g., money transfers between them). Those actions that influence both agents are dealt with in a new interaction subtheory. In some cases we may choose to use the treatments of (Reiter, 1996; Pinto, 1998) for this interaction subtheory, but in our simple example we can do without them.

It is important to notice that we also assume that the different agents have different $time$ fluents. This is used to allow their notions of time to be completely independent until they need to perform

an action that influences both of them. Only when an action that influences both of them needs to be performed do they need to have the same value for their $time$ fluent (because the action happens in the same time for both of them). The details of this issue are further explained below.

The approach we describe gives each agent a different perspective/knowledge of the same situation. It uses a single situations tree, but allows different perspectives on the *time* of a situation. Figure 3.11 is a diagrammatic view of a modification of *Daddy and Junior* (an example taken from (McCarthy and Costello, 1998)). Daddy is stacking blocks of gold in New York, while Junior is stacking blocks of silver in London. If Junior needs more money, he may ask Daddy to send him some (requiring Daddy to sell a block). We omit the treatment for the frame problem here for clarity, but the approach used above works here too.

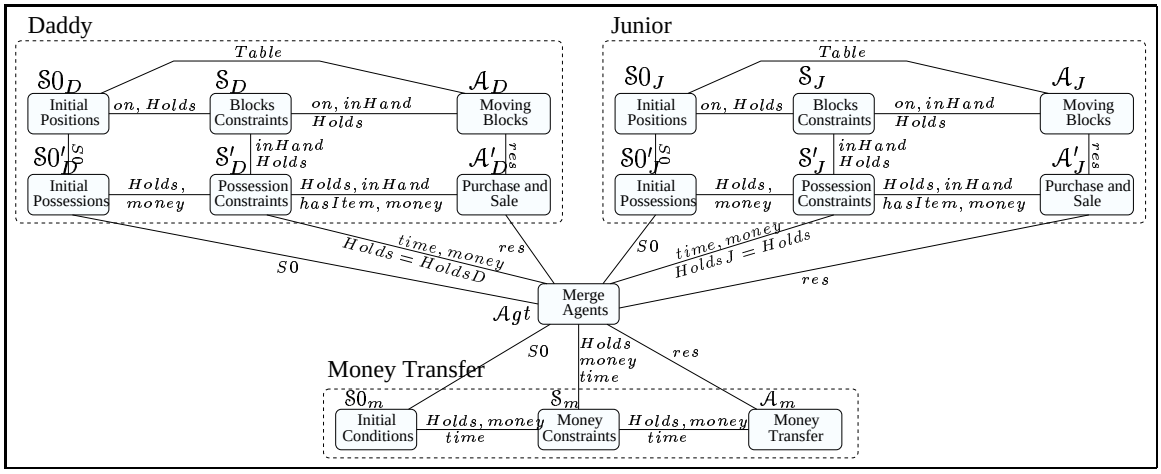

Figure 3.11: Diagrammatic view of Daddy and Junior in object-oriented situation calculus.

Figure 3.12 presents the necessary classes. **MoneySend** is an action class similar to those we have already seen. **MergeAgents** and the links to the object $\mathcal{A}gt$ of that class define our approach for merging the agents' theories (Figure 3.11), and we explain them below.

We care about time synchronization between the agents, so we assume some account of time in the different domain theories. We write $f(p)$ for the fluent that results from f after adding a dependency on p. The axioms of **MergeAgents** are explained as follows. First, if s is a legal joint situation ($legalJoint(s)$), then what holds in it ($Holds$) is exactly what holds in the different perspectives of the different agents ($HoldsD, HoldsJ$). If it is not a legal joint situation, we know nothing about it. Thus, the assumption that our agents' worlds do not interact is realized by giving each agent a different $Holds$ predicate.

class MoneySend : Action {
interface: $money, jointAction$
axioms:
$money(p1, s) = m1 \wedge money(p2, s) = m2 \wedge m1 >= m3 \Rightarrow$
$money(p1, res(send(p1, m3, p2), s)) = m1 - m3 \wedge$
$money(p2, res(send(p1, m3, p2), s)) = m2 + m3$
$jointAction(a) \iff (\exists p1, m, p2) a = send(p1, m, p2)$ }

class MergeAgents : Situation {
interface: $HoldsD, HoldsJ, time, jointAction$
axioms:
$legalJoint(s) \Rightarrow [Holds(f(D), s) \iff HoldsD(f, s)] \wedge$
$[Holds(f(J), s) \iff HoldsJ(f, s)]$
$legalJoint(s) \iff$
$[s = res(a, s') \wedge jointAction(a) \wedge legalJoint(s')] \vee$
$[time(D, s) = time(J, s) \wedge legalJoint(prevJoin(s))]$
$legalJoint(s) \Rightarrow (\forall p) time(s) = time(p, s)$
$prevJoin(res(a, s)) = lastJoin(s)$
$jointAction(a) \Rightarrow lastJoin(res(a, s)) = res(a, s)$
$\neg jointAction(a) \Rightarrow lastJoin(res(a, s)) = lastJoin(s)$
$lastJoin(S0) = S0$
$prevJoin(S0) = S0$ }

Figure 3.12: Merging perspectives on situations.

The rest of the axioms in **MergeAgents** define legal joint situations. These axioms say that s is a legal joint situation iff the agents' perspectives of its history are consistent and their $time$ fluents are synchronized. The two perspectives on history are consistent only if all joint actions (actions for which $jointAction$ holds) in history were done in a legally joint situation. We do not care about actions done separately, as they cannot influence an agent other than the one who executed them. In particular, notice that time does not progress in one agent's perspective of the world as a result of the other agent executing an action.

For example, in Daddy's world (δ_D), the following is true after adding the explanation closure axioms for Daddy: $time(res(\mathcal{A}_J.pickUp(\mathcal{A}_J.A), S0)) = 0$. This is not a legal joint situation because the agents' times are not synchronized. If we now add a *wait* action for Daddy and apply $\mathcal{A}_D.wait(t)$ with the proper time, a legal joint situation would result (in which they can perform $send(D, m, J)$).

3.6 Conditional Independence

Complete knowledge of the world of the links of $\mathcal{A}_J$ renders $\mathcal{A}_J$ independent of all other objects. Proof-theoretically, proving something in the language of $\mathcal{A}_J$ depends only on theorems proved in the language of its links to $\mathcal{S}_J$ and $\mathcal{A}'_J$ (theorems that may depend on further theorems proved elsewhere). This follows from Craig's interpolation theorem (Theorem 4.2.5 in Chapter 4).

From a modeling point of view, this kind of conditional independence allows us to build an object considering only those objects that will interact with it. $\mathcal{A}_m$ of class **MoneySend** depends only on $\mathcal{A}gt$ of **MergeAgents** and $\mathcal{S}_m$ of **MultiHas** (a class that was not detailed here), which in turn care only about money and time. Daddy's world is independent of Junior's, given $\mathcal{A}gt$. This allows the design of theories by first considering their interfaces and only then writing their details. The construction of every domain theory can be done separately, caring only about those predicted inputs from connected theories, which must be in the vocabulary of the interfaces. In particular, we never care about theories that are independent from us, given our neighboring objects/domains.

This conditional independence is also utilized in the solutions to the frame problem. Explanation closure axioms help us make sure that actions in one domain do not influence domains that are not directly connected to it. Money transfers do not change the state of blocks on the table. This analysis can computationally enhance methods for computing solutions to the frame problem.

Finally, rewriting theories is somewhat easier with this modeling technique. If a portion of the theory needs to be revised, the ramifications of this revision are easier to observe and predict. If the object structure is kept, then the influence of intra-object modifications can be predicted by checking the impact on the interface of the changed object(s).

3.7 Related Work: Building Commonsense Theories

The work presented in this chapter is related to works on object-oriented knowledge representation, situation calculus theories and theories of action and to the literature on building commonsense theories and knowledge bases. Throughout the chapter we have mentioned some of the work related to situation calculus and theories of action, and the reader is invited to look at (Reiter, 2001) for a recent book. Chapter 2 covered the related work on object-oriented knowledge representation, so this section is devoted to the work on building commonsense theories and logical knowledge bases.

The effort of building theories that include commonsense axioms was initiated in (McCarthy, 1958a). Currently, there are several branches that advocate different methodologies and several that

investigate the theoretical properties of building logical theories.

The object-oriented approach that we describe in this chapter and Chapter 2 can be seen as close to the logical approach to *Naive Physics* (Hayes, 1979; Hayes, 1985b; Hayes, 1985a; Kamps and P'eli, 1995; Cohn, 1997). Every theory has a complete semantics made of the combination of objects in this theory. The theory is expected to be consistent and correct.

Building object-oriented theories is somewhat closer in spirit and methodology to *microworlds* (de Kleer, 1977; de Kleer and Brown, 1985; Guha and Lenat, 1990; Lenat and Guha, 1990; Davis, 1998), which considers the task of building knowledge bases from a performance-oriented perspective. It takes the stance that modeling can be done for narrow domains, building models using axioms or otherwise. The selection of the proper microtheory for reasoning (or a combination thereof) is a meta-problem of the reasoner ((Davis, 1998) presents a comparison of the naive physics and microworlds methodologies). OOFOL classes are built with the narrow perspective of their performance over their interfaces. The combination of the objects of those classes is a task that is delegated to the knowledge engineer of the particular theory.

Several case studies of commonsense domains that appear in (Davis, 1990; Shanahan, 1998b; Morgenstern, 2001; Lifschitz, 1998; Kamps, 1999) can be seen as examples of microtheories. Also, the approaches to using context for building commonsense theories (Lifschitz, 1986a; Guha, 1991; Shoham, 1991; McCarthy, 1993; Akman and Surav, 1996) can be used to combine microtheories into larger theories. Most recently, (Gustafsson and Kvarnstrom, 2001) have proposed an approach to object-oriented design of TAL theories that is similar to ours, but it is based on notions that are closer to those used in theories of context.

In contrast to these methodologies are approaches that examine the process of theory change and abstraction. The work on elaboration tolerance (Amir, 1997; McCarthy, 1998; Amir, 2000b; Amir, 1999; Lifschitz, 2000; Gustafsson and Kvarnstrom, 2001) examines the difficulty of changes to logical theories that result from discovering new information. Also, the work on abstraction and reformulation (e.g., (Subramanian, 1989; Giunchiglia and Walsh, 1992; Nayak and Levy, 1995; Choueiry and Walsh, 2000)) is important in giving us tools to examine how theories can be changed to answer new tasks. These works are important in giving us theory to examine the methodologies that we use and develop. Further study is needed to examine object-oriented theories using those tools.

3.8 Conclusions

In this chapter we presented three results. First, we presented object-oriented techniques and tools that make designing large situation calculus theories simple. Conditional independence, low-bandwidth links, information hiding and encapsulation (looking at one subject at a time) are powerful tools in designing large theories in general and situation calculus theories in particular.

Second, we showed that it is possible to build situation calculus theories separately and then associate them with loosely interacting agents. The builder of a simple theory does not need to predict its use in a multiple-agents setup. Thus, agents can be treated in the situation calculus without abandoning the *result* formalism. We made an implicit assumption that the agents' separate actions are independent and had their joint actions explicitly stated as such.

Third, our theories can be designed with low-bandwidth links between interacting domains. This leads to good performance expectation for reasoning with these theories.

For simple theories, object-oriented structures are sometimes excessive. This situation is familiar from other fields that use object-oriented techniques. However, it seems that large theories are difficult to build and comprehend without such structures. In this chapter we adjusted this methodology to the modeling of situation calculus theories. We hope that the results we presented here will support the scaling up of logical theory engineering.

Chapter 4

Reasoning with Partitioned Logical Theories

4.1 Introduction

The availability of large logical knowledge bases (KBs) about everyday knowledge about the world, such as Cycorp's Cyc (Lenat and Guha, 1990) and the High Performance Knowledge Base (HPKB) systems developed by Stanford's Knowledge Systems Lab (KSL) (e.g., (Fikes and Farquhar, 1999)) and by SRI (e.g., (Cohen et al., 1998)), raises the problem of reasoning with such large KBs. These KBs comprise tens/hundreds of thousands of logical axioms and are currently beyond the scope of fully automated reasoning. However, they typically exhibit structures that are explicit in the organization of the KBs as micro-theories or in the composition of the KBs from many fragments developed by different groups. Also, some of those KBs belong to different organizations that would prefer not to consolidate them, raising the problem of combining the reasoning without combining the knowledge. Finally, these KBs exhibit structure that may be elicited automatically. An approach that we suggest for reasoning with these KBs while managing their size and complexity is to use the structure provided in these KBs or automatically elicit it.

In this chapter, we investigate how to reason effectively with partitioned sets of logical axioms that have overlap in content and that may even have different reasoning engines. Chapters 2,3 examined the problem of building KBs that are made of parts using the tool of object-oriented design. Chapter 5 examines the more general problem of how to exploit the structure inherent in a set of logical axioms to induce a partitioning of the axioms that will improve the efficiency of reasoning. Together, these chapters provide a picture of how to build KBs from parts, how

to automatically partition KBs and how to take advantage of this partitioning to perform efficient reasoning with the combined KB.

To this end, we propose *partition-based* logical reasoning algorithms for reasoning with logical theories[1] that are decomposed into related partitions of axioms. We provide forward and backward message-passing algorithms, some variants of those and an algorithm for propositional satisfiability (SAT). Our message-passing algorithms are designed so that, without loss of generality, reasoning within a partition can be realized by an arbitrary consequence-finding engine (Lee, 1967). We characterize a class of reasoning procedures that ensures completeness and soundness of our algorithms. We use Craig's interpolation theorem (Craig, 1957a) to prove the soundness and completeness of our algorithms for this class of procedures. The theorem is also used to prove the soundness and completeness of our propositional satisfiability algorithm. We investigate the impact of these algorithms on resolution-based inference and analyze the computational complexity for our partition-based SAT.

A critical aspect of partition-based logical reasoning is the selection of a *good* partitioning of the theory. The computational analysis of our partition-based reasoning algorithms suggests parameters of partitionings that influence the computation of our algorithms: the *bandwidth* of communication between partitions, the size of each partition and the topology of the partitions graph. These parameters guide us in Chapter 5 where we propose algorithms for decomposing logical theories into partitions.

Surprisingly, there has been little work on the specific problem of exploiting structure in theorem proving in the manner presented here. This can largely be attributed to the fact that theorem proving has traditionally examined mathematics domains, which do not necessarily have a structure that supports decomposition. Nevertheless, there are many areas of related work, which I discuss at the end of this chapter.

The rest of the chapter is organized as follows. Section 4.2 describes message-passing algorithms and sufficient conditions for their soundness and completeness. In Section 4.3 we specialize these algorithms to theorem proving using resolution and some of its strategies, relate these algorithms to orderings on literals and discuss the efficiency of message-passing. Section 4.4 offers an algorithm for propositional satisfiability and analyzes its computational complexity. Finally, Section 4.6 discusses some related work. The results in this chapter appeared previously in (Amir and McIlraith, 2000; Amir and McIlraith, 2001a; McIlraith and Amir, 2001).

[1] Recall that we use the convention that every set of axioms is a *theory* (and vice versa). Also, unless stated otherwise, theories, axioms and KBs are in FOL.

4.2 Partition-Based Theorem Proving

In this section we address the problem of how to reason with an already-partitioned propositional or FOL theory using theorem proving. In particular, we propose forward and backward message-passing algorithms in the spirit of Pearl (Pearl, 1988). We further identify conditions under which partition-specific theorem proving results in sound and complete partition-based logical reasoning.

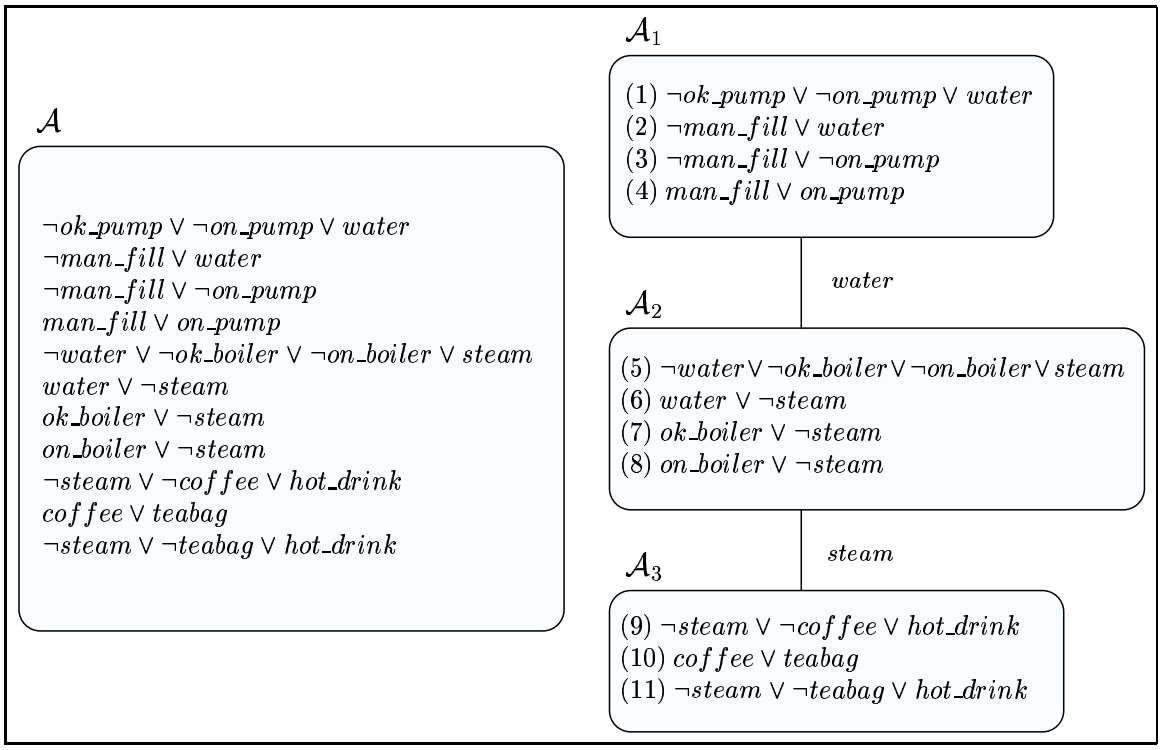

Figure 4.1: A partitioning of $\mathcal{A}$ and its intersection graph.

We define the following. $\{\mathcal{A}_i\}_{i \leq n}$ is a *partitioning* of a logical theory $\mathcal{A}$ if $\mathcal{A} = \bigcup_i \mathcal{A}_i$. Each individual $\mathcal{A}_i$ is called a *partition*, $L(\mathcal{A}_i)$ is its signature (the set of non-logical symbols), and $\mathcal{L}(\mathcal{A}_i)$ is its language (the set of formulae built with $L(\mathcal{A}_i)$). Each partitioning defines a labeled graph $G = (V, E, l)$, which we call the *intersection graph*. In the intersection graph, each node i represents an individual partition $\mathcal{A}_i$, ($V = \{1, ..., n\}$), two nodes i, j are linked by an edge if $L(\mathcal{A}_i)$ and $L(\mathcal{A}_j)$ have a symbol in common ($E = \{(i, j) \mid L(\mathcal{A}_i) \cap L(\mathcal{A}_j) \neq \emptyset\}$), and the edges are labeled with the set of symbols that the associated partitions share ($l(i, j) = L(\mathcal{A}_i) \cap L(\mathcal{A}_j)$). We refer to $l(i, j)$ as the *communication language* between partitions $\mathcal{A}_i$ and $\mathcal{A}_j$. We ensure that the intersection graph is connected by adding a minimal number of edges to E with empty labels,

$l(i,j) = \emptyset$.

We illustrate the notion of a partitioning in terms of the simple propositional theory $\mathcal{A}$, depicted on the left of Figure 4.1 (this is the clausal form of the theory presented with material implication in Figure 4.2). These axioms capture the functioning of aspects of an espresso machine. The first four axioms denote that if the machine pump is OK and the pump is on, then the machine has a water supply. Alternately, the machine can be filled manually, but it is never the case that the machine is filled manually while the pump is on. The next four axioms denote that there is steam if and only if the boiler is OK and is on, and there is a supply of water. The final three axioms state that there is always either coffee or tea, and that steam and coffee (or tea) result in a hot drink.

$ok_pump \wedge on_pump \Rightarrow water$	$man_fill \Rightarrow water$
$man_fill \Rightarrow \neg on_pump$	$\neg man_fill \Rightarrow on_pump$
$water \wedge ok_boiler \wedge on_boiler \Rightarrow steam$	$\neg water \Rightarrow \neg steam$
$\neg ok_boiler \Rightarrow \neg steam$	$\neg on_boiler \Rightarrow \neg steam$
$steam \wedge coffee \Rightarrow hot_drink$	$coffee \vee teabag$
$steam \wedge teabag \Rightarrow hot_drink$	

Figure 4.2: Axiomatization of a simplified espresso machine.

The right-hand side of Figure 4.1 depicts a decomposition of $\mathcal{A}$ into three partitions $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ and its intersection graph. The labels for the edges $(1,2)$ and $(2,3)$ are $\{water\}$ and $\{steam\}$, respectively.

4.2.1 Forward Message Passing

In this section, we propose a forward message-passing algorithm for reasoning with partitions of logical axioms. Figure 4.3 describes our forward message-passing algorithm, FORWARD-M-P (MP), for finding the truth value of query formula $Q \in \mathcal{L}(\mathcal{A}_k)$, $k \leq n$, given partitioned theory $\mathcal{A}$ and graph $G = (V, E, l)$. G may be the intersection graph of $\mathcal{A}$, but is not always so.

To determine the direction in which messages should be sent in the graph G, step 1 in MP computes a strict partial order over nodes in the graph using the partitioning together with a query, Q.

Definition 4.2.1 ($\prec$) *Given partitioned theory $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$, associated graph $G = (V, E, l)$ and query $Q \in \mathcal{L}(\mathcal{A}_k)$, let $dist(i,j)$ $(i, j \in V)$ be the length of the shortest path between nodes i, j in G. Then $i \prec j$ iff $dist(i,k) < dist(j,k)$.*

PROCEDURE FORWARD-M-P (MP)($\{\mathcal{A}_i\}_{i\leq n}$, G, Q)

$\{\mathcal{A}_i\}_{i\leq n}$ a partitioning of the theory $\mathcal{A}$, $G = (V, E, l)$ a graph describing the connections between the partitions, Q a query in $\mathcal{L}(\mathcal{A}_k)$ ($k \leq n$).

1. Determine $\prec$ as in Definition 4.2.1.
2. Concurrently,
   (a) Perform consequence finding for each of the partitions $\mathcal{A}_i$, $i \leq n$.
   (b) For every $(i, j) \in E$ such that $i \prec j$, for every consequence φ of $\mathcal{A}_j$ found (or φ in $\mathcal{A}_j$), if $\varphi \in \mathcal{L}(l(i, j))$, then add φ to the set of axioms of $\mathcal{A}_i$.
   (c) If Q is proven[a] in $\mathcal{A}_k$, return YES.

[a] Derive a subsuming formula or initially add $\neg Q$ to $\mathcal{A}_k$ and derive inconsistency.

Figure 4.3: A forward message-passing algorithm.

This algorithm exploits consequence finding (step (2a)) to perform reasoning in the individual partitions. Consequence finding was defined by Lee (Lee, 1967) to be the problem of finding all the non-tautologous logical consequences of a theory or sentences that subsume them. In clausal FOL, φ subsumes ψ if there is a substitution θ such that $\varphi\theta \subset \psi$. For the non-clausal case of reasoning with FOL, a definition that replaces $\subset$ with $\models$ is used. For precision, we call the first $\subset$-subsumption and the second $\models$-subsumption.

Figure 4.4 illustrates an execution of the MP algorithm using resolution as the consequence finder within a partition. As can be seen from the example, the partitioning reduces the number of possible inference steps by precluding the direct resolution of axioms residing in different partitions.

Completeness & Soundness

MP is sound and complete if we guarantee some properties of the graph G and the consequence finders used for each partition. The graph G is required to be a tree that is *properly labeled* for $\mathcal{A}$.

Definition 4.2.2 (Proper Labeling) *A tree-structured representation, $G = (V, E, l)$, of a partitioned theory $\mathcal{A} = \{\mathcal{A}_i\}_{i\leq n}$ is said to have a* proper labeling, *if for all $(i, j) \in E$ and $\mathcal{B}_1, \mathcal{B}_2$, the two subtheories of $\mathcal{A}$ on the two sides of the edge (i, j) in G, it is true that $l(i, j) \supseteq L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$.*

Using FORWARD-M-P to prove hot_drink			
Partition	Resolve	Generating	
$\mathcal{A}_1$	(2) , (4)	$on_pump \vee water$	(m1)
$\mathcal{A}_1$	$(m1), (1)$	$ok_pump \vee water$	(m2)
$\mathcal{A}_1$	$(m2), (12)$	$water$	(m3)
	$\Longrightarrow$	clause $water$ passed from $\mathcal{A}_1$ to $\mathcal{A}_2$	
$\mathcal{A}_2$	(m3) , (5)	$ok_boiler \wedge on_boiler \supset steam$	(m4)
$\mathcal{A}_2$	(m4) , (13)	$\neg on_boiler \vee steam$	(m5)
$\mathcal{A}_2$	(m5) , (14)	$steam$	(m6)
	$\Longrightarrow$	clause $steam$ passed from $\mathcal{A}_2$ to $\mathcal{A}_3$	
$\mathcal{A}_3$	(9) , (10)	$\neg steam \vee teabag \vee hot_drink$	(m7)
$\mathcal{A}_3$	(m7) , (11)	$\neg steam \vee hot_drink$	(m8)
$\mathcal{A}_3$	(m8) , $(m6)$	hot_drink	(m9)

Figure 4.4: A proof of hot_drink from $\mathcal{A}$ in Figure 4.1 after asserting ok_pump (12) in $\mathcal{A}_1$ and ok_boiler (13), on_boiler (14) in $\mathcal{A}_2$.

For example, every intersection graph that is a tree is properly labeled (We present a simple algorithm called BREAK-CYCLES that transforms an intersection graph into a properly labeled tree in Figure 4.6 on page 60). This property is equivalent to the *running intersection* property required of *join trees* for inference in Bayes nets (Darwiche, 1996; Becker and Geiger, 1996; Shoikhet and Geiger, 1997), constraint satisfaction problems (CSPs) (Dechter and Pearl, 1989; Gottlob et al., 1999) and databases (Jensen et al., 1990).

To ensure completeness of MP, the consequence finders applied to each partition i are required to be *complete for $\mathcal{L}_i$-generation* for a sublanguage $\mathcal{L}_i \subset \mathcal{L}(\mathcal{A}_i)$ that depends on the graph G and the query Q.

Definition 4.2.3 (Completeness for $\mathcal{L}$-Generation) *Let $\mathcal{A}$ be a set of axioms, $\mathcal{L} \subseteq \mathcal{L}(\mathcal{A})$ a language, and $\mathcal{R}$ a consequence finder. Let $C_{\mathcal{R},\mathcal{L}}(\mathcal{A})$ be the consequences of $\mathcal{A}$ generated by $\mathcal{R}$ that are in $\mathcal{L}$. $\mathcal{R}$ is* complete for $\mathcal{L}$-generation *if for all $\varphi \in \mathcal{L}$, if $\mathcal{A} \models \varphi$, then $C_{\mathcal{R},\mathcal{L}}(\mathcal{A}) \models \varphi$.*

Theorem 4.2.4 (Soundness and Completeness) *Let $\mathcal{A}$ be a partitioned theory $\{\mathcal{A}_i\}_{i \leq n}$ of arbitrary propositional or first-order formulae, G a tree that is properly labeled with respect to $\mathcal{A}$, and $Q \in \mathcal{L}(\mathcal{A}_k)$, $k \leq n$, a query. For all $i \leq n$, let $\mathcal{L}_i = \mathcal{L}(l(i,j))$ for j such that $(i,j) \in E$ and $j \prec i$ (there is only one such j), and let $\{\mathcal{R}_i\}_{i \leq n}$ be reasoning procedures associated with partitions*

$\{\mathcal{A}_i\}_{i\leq n}$. If every $\mathcal{R}_i$ is complete for $\mathcal{L}_i$-generation, then $\mathcal{A} \models Q$ iff MP($\{\mathcal{A}_i\}_{i\leq n}$, G, Q) outputs YES.

PROOF See Appendix A.3.1.

We say that $\{\mathcal{R}_i\}_{i\leq n}$ is a *generation-complete set* of reasoners for $\mathcal{A}, G, k$ if it satisfies the condition spelled out in Theorem 4.2.4.

The completeness relies on Craig's interpolation theorem.

Theorem 4.2.5 (Craig's Interpolation Theorem (Craig, 1957a)) *If $\alpha \vdash \beta$, then there is a formula γ involving only symbols common to both α and β such that $\alpha \vdash \gamma$ and $\gamma \vdash \beta$.*

Craig's interpolation theorem is true even if we take α, β to be infinite sets of sentences (Slagle, 1970) and use resolution theorem proving (Slagle, 1970; Huang, 1995) with or without equality (Craig, 1957a; Craig, 1957b) (all after proper reformulation of the theorem).

Definition 4.2.6 (Completeness for $\mathcal{L}$-Consequence Finding) *Let $\mathcal{A}$ be a theory, $\mathcal{L} \subseteq \mathcal{L}(\mathcal{A})$ a language, and $\mathcal{R}$ a consequence finder. $\mathcal{R}$ is* complete for $\mathcal{L}$-consequence finding *iff for every $\varphi \in \mathcal{L}$ that is not a tautology, $\mathcal{A} \models \varphi$ iff there exists $\psi \in \mathcal{L}$ such that $\mathcal{A} \vdash_{\mathcal{R}} \psi$ and ψ subsumes φ.*

An important difference between completeness for $\mathcal{L}$-consequence finding and completeness for $\mathcal{L}$-generation is that completeness for $\mathcal{L}$-generation does not require the reasoner to entail every sound consequence or a formula that subsumes it. Instead, it requires that the set of formulae generated in the sublanguage *collectively* entails all sound consequences of the theory in this sublanguage.

Observe that every reasoner that is complete for $\mathcal{L}$-consequence finding is also complete for $\mathcal{L}$-generation for any language $\mathcal{L}$ that is closed under $\models$-subsumption (del Val, 1999). We say that a procedure is *complete for consequence finding* if it is complete for $\mathcal{L}$-consequence for every $\mathcal{L}(V)$ for any vocabulary V.

The following simple corollary is for reasoning procedures that are complete for consequence finding.

Corollary 4.2.7 *Let $\mathcal{A}$ be a partitioned theory $\{\mathcal{A}_i\}_{i\leq n}$ of arbitrary propositional or first-order formulae, G a tree that is properly labeled with respect to $\mathcal{A}$, and $Q \in \mathcal{L}(\mathcal{A}_k)$, $k \leq n$, a query. If every reasoning procedure for partition $\mathcal{A}_i$ is complete for consequence finding, then $\mathcal{A} \models Q$ iff MP($\{\mathcal{A}_i\}_{i\leq n}$, G, Q) outputs YES.*

PROOF Immediate from the fact that every reasoning procedure that is complete for consequence finding is complete for consequence generation in any language $\mathcal{L}$ that we would like to use in Theorem 4.2.4. ■

Another simple corollary is the following for intersection graphs that are trees.

Corollary 4.2.8 *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned theory with the intersection graph G being a tree (i.e., no cycles). Let $k \leq n$ and φ a sentence in $\mathcal{L}(\mathcal{A}_k)$. If the reasoning procedure in each partition is sound and complete for consequence finding, then $\mathcal{A} \models \varphi$ iff MP outputs YES.*

PROOF See Appendix A.3.2.

In MP, we can use any sound and complete consequence-finding algorithm for reasoning within an individual partition. Reasoning procedures that are complete for *clausal* consequence finding (i.e., complete for consequence finding for languages that are clausal) can be used in MP if the language is propositional or if we use an additional mechanism that transforms the procedure to be complete for FOL consequence finding (we discuss this in more depth in Section 4.3). The *resolution rule* is complete for *clausal* consequence finding (e.g., (Lee, 1967; Slagle, 1970)) and the same is true for several *linear resolution* variants (e.g., (Minicozzi and Reiter, 1972; Inoue, 1992)). A weaker version of completeness for *clausal* consequence finding is also true for *semantic resolution* (Slagle et al., 1969) and *set-of-support resolution*.

Also, there are reasoning methods that focus on a given sublanguage (Finger and Genesereth, 1985; Bossu and Siegel, 1985; Inoue, 1992; Dechter and Rish, 1994; del Val, 1999; Lin, 2000; Kohlas et al., 1999; Marquis, 2000). Examples of such restricted consequence finders are prime implicate generators over a sublanguage. (Recall, that a clause φ is a *prime implicate* of a theory T if $T \models \varphi$ and no formula that subsumes φ is entailed by T.) Such consequence finders are commonly used for prime implicate generation in applications such as diagnosis and abduction (McIlraith, 1998). Consequence finders that focus on a sublanguage can be used directly in MP for reasoning within partitions. Alternately, they can be used in a batch mode to generate select consequences in the sublanguage that are sent as messages in batch. I discuss some examples further in Section 4.3.

Many popular reasoning strategies, such as C-ordered linear resolution, model elimination and set-of-support resolution, are not $\mathcal{L}$-generation complete. One possible way to exploit some variants of these strategies in our message-passing algorithms is to use them to generate *incremental consequences*. These consequences follow from the axioms of the original partition together with the axioms received as messages from adjacent partitions. In this case, each partition has a consequence finder that is $\mathcal{L}$-generation complete and that reasons only with that partition (not including

any messages), and an *auxiliary* reasoner that generates select consequences of the interactions with the messages. *Auxiliary* reasoners must be able to generate enough of the consequences that follow from interactions with the original partition and the messages. Generalizations of C-ordered linear resolution (Inoue, 1991) and set-of-support resolution (Finger, 1987) indeed have the property of completeness for such incremental generators. Other reasoning procedures that are complete for consequence finding in this *incremental* sense are (Kean and Tsiknis, 1990; Jackson, 1992; de Kleer, 1992; Ngair, 1993). We do not pursue this further here, but some details along those lines can be found in (McIlraith and Amir, 2001). Section 4.3 presents other adaptations of MP to some of these and other strategies.

Arbitrary Graphs

If the intersection graph of $\mathcal{A}$ is not a tree, and MP uses it as input, then MP may fail to be a complete proof procedure. Figure 4.5 illustrates the problem. The left-hand side of Figure 4.5 illustrates the intersection graph of partitioning $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4$ of a theory $\mathcal{A}$. If we try to prove s (which follows from $\mathcal{A}$) from this partitioning and graph using MP, nothing will be transmitted between the partitions. For example, we cannot send $p \Rightarrow s$ from $\mathcal{A}_2$ to $\mathcal{A}_4$ because the graph allows only the transmission of sentences containing s.

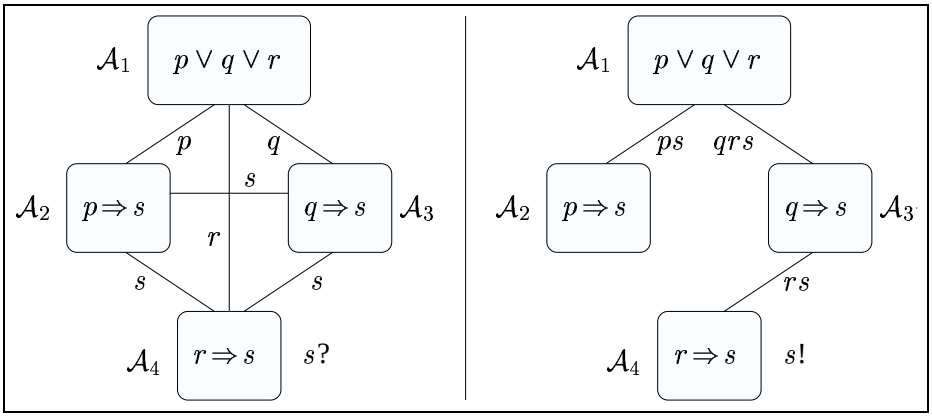

Figure 4.5: An intersection graph before (left) and after (right) applying BREAK-CYCLES.

Thus, using MP with the left-hand graph will fail to prove s. In such a case, we can first syntactically transform the intersection graph into a tree with enlarged labels (i.e., an enlarged communication language) and apply MP to the resultant tree. In particular, we would like the resultant tree to have a *proper labeling* for the given partitioning. Our BREAK-CYCLES algorithm, shown

in Figure 4.6, performs a transformation that produces a tree with a proper labeling from any labeled graph. ($|X|$ denotes the cardinality of a set X.)

PROCEDURE BREAK-CYCLES($G = (V, E, l)$)

1. Find a minimal-length cycle of nodes $v_1, ..., v_c$ ($v_1 = v_c$) in G. If there are no cycles, return G.
2. Select index a s.t. $a < c$ and $\sum_{a \neq j < c} |l(v_j, v_{j+1}) \cup l(v_a, v_{a+1})|$ is minimal (the label of (v_a, v_{a+1}) adds a minimal number of symbols to the rest of the cycle).
3. For all $j < c$, $j \neq a$, set $l(v_j, v_{j+1}) \leftarrow l(v_j, v_{j+1}) \cup l(v_a, v_{a+1})$.
4. Set $E \leftarrow E \setminus \{(v_a, v_{a+1})\}$, $l(v_a, v_{a+1}) \leftarrow \emptyset$ and go to (1).

Figure 4.6: An algorithm to transform an intersection graph G into a tree.

Using BREAK-CYCLES, we can transform the graph depicted on the left-hand side of Figure 4.5, into the tree on its right. First, we identify the minimal cycle $\langle (1,3), (3,4), (4,1) \rangle$, remove $(4,1)$ from E and add r to the labels of $(1,3), (3,4)$. Then, we find the minimal cycle $\langle (2,3), (3,4), (4,2) \rangle$ and remove $(2,3)$ from E (s already appears in the labels of $(4,2), (3,4)$). Finally, we identify the minimal cycle $\langle (1,3), (3,4), (4,2), (2,1) \rangle$, remove $(4,2)$ and add s to the rest of the cycle. The proof of s by MP now follows by sending $p \Rightarrow s$ from $\mathcal{A}_2$ to $\mathcal{A}_1$, sending $q \vee r \vee s$ from $\mathcal{A}_1$ to $\mathcal{A}_3$, sending $r \vee s$ from $\mathcal{A}_3$ to $\mathcal{A}_4$ and concluding s in $\mathcal{A}_4$.

The following corollary says that, to use MP, all we need to do is to use the graph that results from BREAK-CYCLES applied to the intersection graph of $\mathcal{A}$.

Corollary 4.2.9 *Let $\mathcal{A}$ be a partitioned theory $\{\mathcal{A}_i\}_{i \leq n}$ of arbitrary propositional or first-order formulae, G the intersection graph of $\{\mathcal{A}_i\}_{i \leq n}$, and $Q \in \mathcal{L}(\mathcal{A}_k)$, $k \leq n$, a query. If every reasoning procedure for partition $\mathcal{A}_i$ is complete for consequence finding, then $\mathcal{A} \models Q$ iff MP($\{\mathcal{A}_i\}_{i \leq n}$, BREAK-CYCLES($G$), Q) outputs YES.*

PROOF See Appendix A.3.3.

Notice that when executing BREAK-CYCLES, we may remove an edge that participates in more than one minimal cycle (as is the case of removing the edge $(4,1)$), but its removal influences the labels of only one cycle.

BREAK-CYCLES is a greedy algorithm that has a worst-case complexity of $O(|E|^2 * m)$ (where m is the number of symbols in $L(\mathcal{A})$). The rationale is roughly as follows: There are at most $|E|$

cycles that can be *broken*, step (1) takes $O(|E|)$ time, step (3) takes $O(|E| * m)$ time, and step (2) can be implemented to take $O(|E| * m)$ time using dynamic programming.

Other algorithms that we may use in this context are variants on the *cutset* method for reasoning with graphs (Becker and Geiger, 1994; Becker et al., 2000). Darwiche (Darwiche, 1996) used an algorithm that is similar to BREAK-CYCLES for the problem of creating a join tree. Our algorithm differs from Darwiche's in treating an already-formed partitioning and creating the tree in a greedy way (Darwiche's method randomly selects a tree).

4.2.2 Backward Message Passing

Our MP algorithm uses the query Q to induce an ordering on the partitions, which in turn may guide selective consequence finding for *reasoning forward*. Many theorem proving strategies exploit the query more aggressively by *reasoning backward* from the query. Such strategies have proven effective for a variety of reasoning problems, such as planning. Indeed, many theorem provers (e.g., PTTP (Stickel, 1992)) are built as backward reasoners and must have a query or *goal* in order to run.

One way to use MP for an analogous backward message-passing scheme is to assert $\neg Q$ in $\mathcal{A}_k$, choose a partition $\mathcal{A}_j$ that is most distant from $\mathcal{A}_k$ in G (where the distance between two nodes in graph G is the number of nodes comprising the shortest path between the two nodes), and try to prove $\{\}$ in $\mathcal{A}_j$ using MP. If we wish to follow the spirit of backward-reasoning more closely, we can transform G into a *chain* in a similar way as our transformation of G into a tree using BREAK-CYCLES. The resultant chain graph may then be used for query-driven backward message passing from $\mathcal{A}_k$. We present such an algorithm, called BACKWARD-M-P (BMP), in Figure 4.7. BMP takes as input a partitioned theory $\mathcal{A}$, a graph G_0 and a query Q, and returns YES if it can prove Q.

Procedure CHAINIFY is outlined in Figure 4.8. It accepts a labeled graph and returns a transformation of the graph into a chain (changing the labels appropriately). Alternately, we can create a chain directly from the partitions and a total order over them. CHAINIFY ensures that the resulting graph has a proper labeling. BMP is sound and complete if the reasoning procedure used in every partition is complete for consequence finding.

Theorem 4.2.10 (Soundness and Completeness) *Let $\mathcal{A}$ be a partitioned theory $\{\mathcal{A}_i\}_{i\leq n}$ of arbitrary propositional or first-order formulae, G the intersection graph of $\mathcal{A}$, and $Q \in \mathcal{L}(\mathcal{A}_k)$, $k \leq n$, a query. For all $i \leq n$, let $\mathcal{L}_i = \mathcal{L}(l(i,j))$ for j such that $(i,j) \in E$ and $i \prec j$ (there is only one such j determined by step 1 of BMP), and let $\{\mathcal{R}_i\}_{i\leq n}$ be reasoning procedures associated with*

PROCEDURE BACKWARD-M-P(BMP)($\{\mathcal{A}_i\}_{i\leq n}$, G_0, Q)
$\{\mathcal{A}_i\}_{i\leq n}$ a partitioned theory, $G_0 = (V, E, l)$ a graph, Q a query in $\mathcal{L}(\mathcal{A}_k)$ ($k \leq n$).

1. $G \leftarrow$ CHAINIFY(G_0, k). Let $i \prec j$ be as in Definition 4.2.1.
2. For all $i \leq n$, $i \neq k$ set $goal_i \leftarrow$ *FALSE* (the goal of $\mathcal{A}_i$ is proving FALSE). Set $goal_k \leftarrow Q$.
3. Concurrently,
   (a) For each partition $\mathcal{A}_i$, $i \leq n$, attempt to prove $goal_i$.
   (b) For every $(i, j) \in E$ such that $j \prec i$, if we generate a subgoal[a] φ in $\mathcal{A}_j$ and $\varphi \in \mathcal{L}(l(i, j))$, then set[b] $goal_i \leftarrow goal_i \vee \varphi$.
   (c) If $goal_i$ is proved in any $\mathcal{A}_i$, return YES.

[a] In resolution, every generated clause can be considered the negation of a subgoal.
[b] In resolution refutation, the goal is negated, so this step essentially adds $\neg\varphi$ to the axioms of $\mathcal{A}_i$.

Figure 4.7: A backward message-passing algorithm.

partitions $\{\mathcal{A}_i\}_{i\leq n}$. *If every* $\mathcal{R}_i$ *is complete for* $\mathcal{L}_i$*-generation, then* $\mathcal{A} \models Q$ *iff BMP(*$\{\mathcal{A}_i\}_{i\leq n}$*,* G*,* Q*) outputs YES.*

PROOF See Appendix A.3.4.

Algorithm BMP is presented for the case of *subgoal-disjunctive* systems, i.e., a proof of any subgoal yields a proof of the entire query. This is the case with resolution and its variants. The intuition behind the algorithm is that when a partition is supplied a subgoal sentence φ from another partition, φ is added to (*OR*-ed with) to the partition's goal.

We make G a chain because otherwise subgoals may have to split between partitions. Splitting subgoals requires accounting for different preconditions (as in natural deduction), which we wish to avoid here, for simplicity of inference.

4.2.3 Queries Drawn From Multiple Partitions

MP and its variants require that query Q be in the language of a single partition, $\mathcal{L}(\mathcal{A}_k)$, for some $k \leq n$. One way to answer a query Q that comprises symbols drawn from multiple partitions is to add a new partition $\mathcal{A}_Q$ with language $\mathcal{L}(\mathcal{A}_Q) = \mathcal{L}(Q)$, the language of the query. $\mathcal{A}_Q$ may contain $\neg Q$ or no axioms. Following the addition of this new partition, BREAK-CYCLES must be run on the new intersection graph to ensure a proper labeling of G for the partitioned theory (as discussed in Section 4.2.1). To prove Q in $\mathcal{A}_Q$, we run MP on the resulting graph.

PROCEDURE CHAINIFY(G, k)
$G = (V, E, l)$, a graph describing connections between partitions, $k \leq |V|$.

1. Let $dist(i, j)$ ($i, j \in V$) be the length of the shortest path between i, j in G. Let $i \prec_0 j$ iff $dist(i, k) < dist(j, k)$ ($\prec_0$ is a strict partial order).
2. Impose a total order $\prec$ on V that agrees with $\prec_0$ (i.e., $i \prec_0 j \Rightarrow i \prec j$).
3. Let $\{v_a\}_{a \leq n} = V$ such that $v_1 = k$, $\forall a \leq n\, v_a \prec v_{a+1}$.
4. Let $E' = \{(v_a, v_{a+1})\}_{i<n}$.
5. Set $l'(i, j) \leftarrow \emptyset$ for all $i, j \in V$.
6. For all $(i, j) \in E$, for all $a < n$, if $i \preceq v_a \prec j$ (i.e., v_a is between i and j), then set $l'(v_a, v_{a+1}) \leftarrow l'(v_a, v_{a+1}) \cup l(i, j)$.
7. Return $G' = (V, E', l')$.

Figure 4.8: A procedure that transforms a graph G to a chain G'.

Alternately, we can decompose the query into the appropriate partitions following the methods of (Nelson and Oppen, 1979) or (Shostak, 1984). Since the issue of decomposing a query is not simple, we describe only the simple case of a propositional query and leave the first-order case (with literals that contain symbols from multiple partitions) for future work.

Given a propositional query Q, we transform it to the form $(Q^1_1 \vee ... \vee Q^1_{r_1}) \wedge ... \wedge (Q^l_1 \vee ... Q^l_{r_l})$, where each Q^i_j is a formula in the language of a single partition $\mathcal{L}(\mathcal{A}_{k_{ij}})$ (k_{ij} is the index of a partition that includes the vocabulary of Q^i_j). For example, if Q is in CNF, it is already in this form. We check a disjunct $Q^i_1 \vee ... \vee Q^i_{r_i}$ by asserting $\neg Q^i_j$ in $\mathcal{A}_{k_{ij}}$ for all $j \leq r_i$, and proving FALSE in one of the partitions. To prove Q, we check each of the disjuncts in its transformed form. It is a valid consequence of $\mathcal{A}$ iff all the disjuncts are valid consequences of $\mathcal{A}$. We discuss this special topic no further here and assume Q is drawn from $\mathcal{L}(\mathcal{A}_k)$ for some $k \leq n$.

4.3 Resolution and Resolution Strategies

The previous section presented message-passing algorithms with an arbitrary sound and complete consequence finder. In this section, we specialize our message-passing algorithms with consequence finders that specifically employ *resolution* and several of its refinements. We focus on the first-order case of resolution and the restriction strategies of *linear resolution*, *set-of-support resolution* and

some of their variants. Also, we relate MP to resolution strategies that use order on literals or symbols. We conclude with an analysis of the effect message passing has on the computational efficiency of resolution-based inference.

The presentation in this section makes explicit reference to the forward message-passing algorithm, MP, but we wish to stress that the results in this section are equally applicable to other message-passing algorithms introduced in the previous section. For background material on resolution and resolution strategies, the reader is referred to (Genesereth and Nilsson, 1987; Eisinger and Ohlbach, 1993; Chang and Lee, 1973; Loveland, 1978).

4.3.1 Resolution Message-Passing

Resolution (Robinson, 1965) is one of the most widely used reasoning methods for automated deduction, and more specifically for consequence finding. As noted in Section 4.2, the resolution rule is complete for *clausal* consequence finding. It requires the input formula to be in clausal form, i.e., a conjunction of disjunctions of unquantified literals. For general first-order formulae, a transformation to clausal form (e.g., (Lloyd and Topor, 1985)) includes *Skolemization*, which eliminates quantifiers and possibly introduces new constant symbols and new function symbols.

We present algorithm RESOLUTION-M-P (RES-MP), which uses resolution (or resolution strategies), in Figure 4.9. The rest of this section is devoted to explaining four different implementations for subroutine RES-SEND(φ, j, i), used by this procedure to send appropriate messages across partitions: the first implementation is for clausal propositional theories; the second is for clausal FOL theories, with associated graph G, which is a properly labeled tree and whose labels include all the function and constant symbols of the language; the third is also for clausal FOL theories, but it uses *unskolemization* and subsequent Skolemization to generate the messages to be passed across partitions; the fourth is a refinement of the third for the same class of theories that avoids unskolemization.

In the propositional case, subroutine RES-SEND(φ, j, i) (Implementation 1) simply adds φ to $\mathcal{A}_i$, as done in MP. If the resolution strategies being employed, $\{\mathcal{R}_i\}_{i \leq n}$, are complete for $\mathcal{L}_i$-consequence finding (for $\mathcal{L}_i$ the clausal sublanguage of $\mathcal{L}(l(i,j))$, for $j \prec i$ and $(i,j) \in E$), then RES-MP is sound and complete. This is because completeness for $\mathcal{L}_i$-consequence finding for the clausal sublanguage $\mathcal{L}_i$ of $\mathcal{L}(l(i,j))$ implies completeness for $\mathcal{L}(l(i,j))$-consequence finding, if $l(i,j)$ is propositional.

In the FOL case, implementing RES-SEND requires more care. To illustrate, consider the case

PROCEDURE RESOLUTION-M-P(RES-MP)($\{\mathcal{A}_i\}_{i\leq n}$, G, Q)
$\{\mathcal{A}_i\}_{i\leq n}$ a partitioned theory, $G = (V, E, l)$ a graph, Q a query formula in the language of $\mathcal{L}(\mathcal{A}_k)$ ($k \leq n$).

1. Determine $\prec$ as in Definition 4.2.1.
2. Add the clausal form of $\neg Q$ to $\mathcal{A}_k$.
3. Concurrently,
   (a) Perform resolution for each of the partitions $\mathcal{A}_i$, $i \leq n$.
   (b) For every $(i, j) \in E$ such that $i \prec j$, if partition $\mathcal{A}_j$ includes the clause φ (as input or resolvent) and the predicates of φ are in $\mathcal{L}(l(i, j))$, then perform RES-SEND(φ, j, i).
   (c) If Q is proven in $\mathcal{A}_k$, return YES.

Figure 4.9: A resolution forward message-passing algorithm.

where resolution generates the clause $P(B, x)$ (B a constant symbol and x a variable). It also implicitly proves that $\exists b\, P(b, x)$. RES-MP may need to send $\exists b\, P(b, x)$ from one partition to another, but it cannot send $P(B, x)$ if B is not in the communication language between partitions (for ground theories there is no such problem (see (Slagle, 1970))). In the first-order case, completeness for consequence finding for a clausal first-order logic language (e.g., Lee's result for resolution) does not guarantee completeness for $\mathcal{L}$-generation for the corresponding full FOL language, $\mathcal{L}$. This problem is also reflected in a slightly different statement of Craig's interpolation theorem (Craig, 1957a) that applies for resolution (Slagle, 1970).

A simple way of addressing this problem is to add all constant and function symbols to the communication language between every connected set of partitions. This has the advantage of preserving soundness and completeness, and is simple to implement. In this case, subroutine RES-SEND(φ, j, i) (Implementation 2) simply adds φ to $\mathcal{A}_i$, as done in MP.

In large systems that consist of many partitions, the addition of so many constant and function symbols to each for the other partitions has the potential to be computationally inefficient, leading to many unnecessary and irrelevant deduction steps. Arguably, a more compelling way of addressing the problems associated with resolution for first-order theories is to infer the existential formula $\exists b\, P(b, x)$ from $P(B, x)$, send this formula to the proper partition and Skolemize it there. For example, if $\varphi = P(f(g(B)), x)$ is the clause that RES-SEND gets, replacing it with $\exists b\, P(b, x)$ eliminates unnecessary work of the receiving partition.

The process of conservatively replacing function and constant symbols by existentially quantified variables is called *unskolemization* or *reverse Skolemization* and is discussed in (Bledsoe and

Ballantyne, 1978; Cox and Pietrzykowski, 1984; Chadha and Plaisted, 1993). (Chadha and Plaisted, 1993) presents an algorithm *U* that is complete for our purposes and generalizes and simplifies an algorithm of (Cox and Pietrzykowski, 1984). (Space precludes inclusion of the algorithm.)

Theorem 4.3.1 ((Chadha and Plaisted, 1993)) *Let V be a vocabulary and φ, ψ be formulae such that $\psi \in \mathcal{L}(V)$ and $\varphi \Rightarrow \psi$. There exists $F \in \mathcal{L}(V)$ that is generated by algorithm* U *such that $F \Rightarrow \psi$.*

Thus, for every resolution strategy that is complete for $\mathcal{L}$-consequence finding, unskolemizing φ using procedure *U* for $V = l(i,j)$ and then Skolemizing the result gives us a combined procedure for message generation that is complete for $\mathcal{L}_j$-generation. This procedure can then be used readily in RES-MP (or in MP), upholding the soundness and completeness to that supplied by Theorem 4.2.4. The subroutine RES-SEND(φ, j, i) that implements this approach is presented in Figure 4.10. It replaces φ with a a set of formulae in $\mathcal{L}(l(i,j))$ that follows from φ. It then Skolemizes the resulting formulae for inclusion in $\mathcal{A}_i$.

PROCEDURE RES-SEND(φ, j, i) (Implementation 3)
φ a formula, $j, i \leq n$.

1. Unskolemize φ into a set of formulae, Φ in $\mathcal{L}(l(i,j))$, treating every symbol of $L(\varphi) \setminus l(i,j)$ as a Skolem symbol.
2. For every $\varphi_2 \in \Phi$, if φ_2 is not subsumed by a clause that is in $\mathcal{A}_i$, then add the Skolemized version of φ_2 to the set of axioms of $\mathcal{A}_i$.

Figure 4.10: Subroutine RES-SEND using unskolemization.

Procedure *U* may generate more than one formula for any given clause φ. For example, if $\varphi = P(x, f(x), u, g(u))$, for $l(i,j) = \{P\}$, then we must generate both $\forall x \exists y \forall u \exists v P(x,y,u,v)$ and $\forall u \exists v \forall x \exists y P(x,y,u,v)$ (φ entails both quantified formulae, and there is no one quantified formula that entails both of them). In our case we can avoid some of these quantified formulae by replacing the *unskolemize and then Skolemize* process of RES-SEND (Implementation 3) with a procedure that produces a set of formulae directly (Implementation 4). It is presented in Figure 4.11.

Steps 2 and 3 of procedure RES-SEND(φ, j, i) (Implementation 4) correspond to similar steps in procedure *U* presented in (Chadha and Plaisted, 1993), simplifying where appropriate for our setup. Our procedure differs from unskolemizing procedures in Step 4, where it stops short of replacing the Skolem functions and constants with new, existentially quantified variables. Instead,

PROCEDURE RES-SEND(φ, j, i) (Implementation 4)

φ a formula, $j, i \leq n$.

1. Set $S \leftarrow L(\varphi) \setminus l(i,j)$.
2. For every term instance, $t = f(t_1, ..., t_k)$, in φ, if $f \in S$ and t is not a subexpression of another term $t' = f'(t'_1, ..., t'_{k'})$ of φ with $f' \in S$, then replace t with "$x \leftarrow t$" for some new variable, x (if $k = 0$, t is a constant symbol).
3. Nondeterministically[a], for every pair of marked arguments "$x \leftarrow \alpha$", "$y \leftarrow \beta$" in φ, if α, β are unifiable, then unify all occurrences of x, y (i.e., unify α_i, β_i for all markings $x \leftarrow \alpha_i$, $y \leftarrow \beta_i$).
4. For every marked argument "$x \leftarrow \alpha$" in φ,
   (a) Collect all marked arguments with the same variable on the left-hand side of the "$\leftarrow$" sign. Suppose these are $x \leftarrow \alpha_1, ..., x \leftarrow \alpha_l$.
   (b) Let $y_1, ..., y_r$ be all the variables occurring in $\alpha_1, ..., \alpha_l$. For every $i \leq l$, replace "$x \leftarrow \alpha_i$" with $f(y_1, ..., y_r)$ in φ, for a fresh function symbol f (if $r = 0$, f is a fresh constant symbol).
5. Add φ to $\mathcal{A}_i$.

[a] Nondeterministically select the set of pairs for which to unify all occurrences of x, y.

Figure 4.11: Subroutine RES-SEND without unskolemization.

it replaces them with new functions and constant symbols. The nondeterminism of Step 3 is used to add *all* the possible combinations of unified terms, which is required to ensure completeness.

For example , if $\varphi = P(f(g(B)), x)$ and $l(i,j) = \{P\}$, then RES-SEND (implementation 4) adds $P(A, x)$ to $\mathcal{A}_i$, for a new constant symbol, A. If $\varphi = P(x, f(x), u, g(u))$, for $l(i,j) = \{P\}$, then RES-SEND adds $P(x, h_1(x), u, h_2(u))$ to $\mathcal{A}_i$, for new function symbols h_1, h_2. Finally, if $\varphi = P(x, f(x), u, f(g(u)))$, then RES-SEND adds $P(x, f(x), u, h(u))$ and $P(h_1(u), h_2(u), u, h_2(u))$ to $\mathcal{A}_i$, for h, h_1, h_2 new function symbols.

Theorem 4.3.2 (Soundness & Completeness of RES-MP) *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned theory of propositional or first-order clauses, G a tree that is properly labeled with respect to $\mathcal{A}$, and $Q \in \mathcal{L}(\mathcal{A}_k)$, $k \leq n$, be a sentence that is the query. $\mathcal{A} \models Q$ iff applying RES-MP($\{\mathcal{A}_i\}_{i \leq n}$, G, Q) (with Implementation 4 of RES-SEND) outputs YES.*

PROOF See Appendix A.3.5.

4.3.2 Linear Resolution

In this section we show how linear resolution, a sound and complete resolution strategy for consequence finding, can be exploited for first-order MP.

For a set of axioms $\mathcal{A}$, *linear resolution* (Loveland, 1970; Luckham, 1970; Anderson and Bledsoe, 1970; Yates et al., 1970) restricts the search space of resolution by allowing only resolutions in which one of the clauses is part of the initial set of clauses (the *input*) or those in which one of the clauses is an ancestor of the other in the resolution tree. Linear resolution is refutation-complete. It is also complete for consequence finding (Minicozzi and Reiter, 1972).

Figure 4.12 presents LIN-MP, our procedure for forward message passing using linear restriction in each of the partitions. Backward message passing with linear resolution is analogous. LIN-MP takes as input a partitioned theory $\mathcal{A}$, a graph G and a query Q, and returns YES if it can prove Q.

PROCEDURE LIN-MP($\{\mathcal{A}_i\}_{i\leq n}$, G, Q)
$\{\mathcal{A}_i\}_{i\leq n}$ a partitioning of the theory $\mathcal{A}$, $G = (V, E, l)$ a graph, Q a query formula in the language of $\mathcal{L}(\mathcal{A}_k)$ ($k \leq n$).

1. Let $dist(i, j)$ ($i, j \in V$) be the length of the shortest path between i, j in G. Let $i \prec j$ iff $dist(i, k) < dist(j, k)$ ($\prec$ is a strict partial order).
2. Concurrently,
   (a) For every $(i, j) \in E$ such that $i \prec j$, perform linear resolution for $\mathcal{A}_i$.
   (b) If at any point $\mathcal{A}_j \models \varphi$ is proven and φ's signature is in $\mathcal{L}(l(i, j))$, then add φ to the axioms of $\mathcal{A}_i$, considering φ as *input* for further linear resolutions.
   (c) If Q is proven in $\mathcal{A}_k$, return YES.

Figure 4.12: Linear resolution with forward message passing.

Theorem 4.3.3 (Soundness and Completeness) *Let $\mathcal{A} = \bigcup_{i\leq n} \mathcal{A}_i$ be a partitioned theory with a tree G that is properly labeled for $\mathcal{A}$. Let $k \leq n$ and φ be a sentence whose signature is in $\mathcal{L}(\mathcal{A}_k)$. $\mathcal{A} \models \varphi$ iff applying LIN-MP outputs YES.*

PROOF Since linear resolution is complete for clausal consequence finding (Minicozzi and Reiter, 1972), it is enough to show that the way messages are handled does not jeopardize this completeness. In LIN-MP, messages are treated as *input clauses* in the receiving partition. Thus, both the original input clauses and the messages are considered *input clauses*. This means that valid consequences of the original set of clauses and the received messages are generated by linear resolution

in each partition. This guarantees completeness for consequence generation in each partition, and the theorem follows from Theorem 4.2.4. ■

It is also possible to define a more restrictive version of message passing that exploits linear resolution. Instead of treating the messages as inputs in the receiving partition, we may treat them as resolved clauses (i.e., non-input clauses). It is not clear if this restriction can be made complete. It is more restrictive than simple linear resolution in that two clauses may not be resolved against each other even though one is a descendent of the other.

There are variants of linear resolution that include ordering strategies such as A-ordering (e.g., (Reiter, 1971)), C-ordering (e.g., (Reiter, 1971)) and Model Elimination (Loveland, 1969; Stickel, 1988a) (more on ordering strategies in Section 4.3.4). All of these ordering-based linear resolution strategies turn out to be incomplete for consequence finding, despite being refutation-complete. Nevertheless, linear resolution with A-ordering can be made $\mathcal{L}$-generation complete for the necessary $\mathcal{L}$ in a way similar to that used in Section 4.3.4.

4.3.3 Set-of-Support and Semantic Resolution

In this section we examine the relevance of set-of-support and semantic resolution to our message-passing algorithms. In the general case, semantic resolution and set-of-support resolution are both incomplete for consequence finding. Nevertheless, we are able to propose an algorithm for sound and complete *propositional* forward message passing using semantic resolution or set-of-support.

For a set of axioms $\mathcal{A}$, *set-of-support resolution* (Wos et al., 1965) restricts the search space of resolution by distinguishing a set $S \subset \mathcal{A}$ as a *support* and disallowing resolutions between axioms in $\mathcal{A} \setminus S$. Resolvents (between clauses in S or between a clause in $\mathcal{A} \setminus S$ and a clause in S) are added to S. A *set-of-support refutation* is such a sequence of resolutions leading to the empty clause ($\{\}$). This algorithm is refutation-complete if $\mathcal{A} \setminus S$ is consistent. Typically, when trying to prove $T \vdash \varphi$, we assume that T is consistent and put $\neg\varphi$ in the set of support.

Semantic resolution (Slagle, 1967) is similar to set-of-support resolution in that it divides clauses into two distinguished sets, S_1, S_2. It differs from set-of-support in two main ways. First, it requires an interpretation I to be supplied. This interpretation is used to determine the division of axioms into the two sets. S_1 includes clauses that I satisfies, and S_2 includes clauses that I does not satisfy. Second, it allows resolution of two clauses only if each comes from a different set (set-of-support resolution allows resolutions between clauses in the support). Resolvents are added to the proper set (S_1 if I satisfies them, or S_2 otherwise). A *semantic I-refutation* (with an interpretation I) is

such a sequence of resolutions, leading to the empty clause ($\{\}$).

The following theorem gives us some insight into the applicability of both semantic resolution and set-of-support resolution as reasoning procedures in our message-passing algorithms. From this theorem, we are able to propose an algorithm for sound and complete forward message passing using semantic resolution or set-of-support. The theorem holds for both FOL and propositional logic.

Theorem 4.3.4 ((Slagle et al., 1969)) *Let C be a clause, S be a finite set of clauses and I be an interpretation in $\mathcal{L}(S \cup \{C\})$. Assume C is a prime implicate of S. If $I \not\models C$, then there is a semantic I-deduction of C from S.*

This theorem implies a restricted form of completeness for consequence finding: Given an interpretation, I, semantic resolution will generate all the prime implicates that are *falsified* by I. Semantic resolution is a more restricted strategy than set-of-support (given the right interpretation, I), so this theorem applies for set-of-support as well.

Figure 4.13 presents SEM-MP, our procedure for *propositional* forward message-passing using semantic restriction in each of the partitions. Backward message-passing with semantic resolution is analogous. It takes as input a partitioned theory $\mathcal{A}$, a graph G, and a query, Q, and returns YES if it proves Q.

The reason this algorithm applies only to the propositional case is that Step 1 in procedure MULTI-SEM requires us to enumerate all interpretations over $l(i,j)$. Procedure SEM-MP applies semantic resolution in MP in a way that uses several complementary interpretations. For each message that may be sent, we keep an interpretation that does not satisfy it. For example, if an edge label includes l propositional symbols, we need to keep 2^l interpretations (each one does not satisfy a clause of size l in the language of the link). If an interpretation satisfies all the clauses, then we do not need to consider it in running our semantic resolution. This completes the procedures for consequence finding and makes SEM-MP complete.

Theorem 4.3.5 (Soundness and Completeness) *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned theory with the properly labeled tree G. Let $k \leq n$ and φ be a sentence whose signature is in $\mathcal{L}(\mathcal{A}_k)$. $\mathcal{A} \models \varphi$ iff applying SEM-MP outputs YES.*

PROOF Procedure MULTI-SEM generates all the consequences in the language in which messages need to be sent, according to Theorem 4.3.4. By Theorem 4.2.4, this procedure is sound and complete. ■

PROCEDURE MULTI-SEM($\mathcal{A}$, M, L)

$\mathcal{A}$ a theory, M an interpretation, L a set of symbols in $\mathcal{L}(\mathcal{A})$.

1. For every possible truth assignment, a, to L, let M_a be an interpretation derived from M by changing the truth assignment to the values mentioned in a. Let A be the set of all those truth assignments.
2. For every interpretation in $\{M_a\}_{a \in A}$, let $S_a = \{\varphi \mid \varphi \in \mathcal{A},\ M_a \not\models \varphi\}$.
3. For every interpretation $a \in A$, perform semantic resolution with M_a.

PROCEDURE SEM-MP($\{\mathcal{A}_i\}_{i \leq n}$, $\{M_i\}_{i \leq n}$, G, Q)

$\{\mathcal{A}_i\}_{i \leq n}$ a partitioning of a theory $\mathcal{A}$, $\{M_i\}_{i \leq n}$ interpretations in languages corresponding to the partitioning of $\mathcal{A}$, $G = (V, E, l)$ a graph, Q a query formula in the language of $\mathcal{L}(\mathcal{A}_k)$ ($k \leq n$).

1. Let $dist(i, j)$ ($i, j \in V$) be the length of the shortest path between i, j in G. Let $i \prec j$ iff $dist(i, k) < dist(j, k)$ ($\prec$ is a strict partial order).
2. Concurrently,
   (a) For every $(i, j) \in E$ such that $i \prec j$, perform MULTI-SEM($\mathcal{A}_i$, M_i, $l(i, j)$).
   (b) If at any point $\mathcal{A}_j \models \varphi$ is proven and φ's signature is in $\mathcal{L}(l(i, j))$, then add φ to $\mathcal{A}_i$ (putting it in the correct set for each concurrently running semantic resolution).
   (c) If Q is proven in $\mathcal{A}_k$, return YES.

Figure 4.13: Semantic resolution with forward message passing.

It is not clear if there is an algorithm that corresponds more closely to set-of-support resolution that is also complete (however, see Section 4.2.1 for another way of using set-of-support resolution with MP).

4.3.4 Directional Resolution, A-ordering and Lock Resolution

So far we have examined resolution strategies, demonstrating their applicability to our message-passing algorithms. The results in this section serve to place our message-passing algorithms in a broader context by relating them to some other resolution strategies. We examine three resolution strategies that are *incomplete* for consequence finding: directional resolution, A-ordering resolution and lock resolution. These strategies use orderings on symbols/literals to restrict the space of possible resolutions.

We present two interesting results relating these resolution strategies to our message-passing algorithms. First, we show that our forward message-passing algorithm, MP, can simulate the symbol ordering strategies of directional resolution, A-ordering and lock resolution if the orders correspond to orders on predicate/propositional symbols. Second, we show that in the propositional case directional resolution, A-ordering and lock resolution can likewise simulate MP, if MP uses unrestricted or directional resolution in each of the partitions. In the FOL case, the same result holds if all the function and constant symbols are on all the links between partitions.

Directional resolution[2] (Davis and Putnam, 1960; Dechter and Rish, 1994) is a procedure for concluding satisfiability of a propositional logic theory. For a set of propositional axioms $\mathcal{A}$ and an order on the propositional symbols $Q_1, ..., Q_n$ (Q_n is the *highest* symbol), directional resolution *resolves out* each of the propositions in order, if it can. Dechter and Rish (Dechter and Rish, 1994) present directional resolution using the framework of bucket elimination: Partition the clauses of $\mathcal{A}$ into $bucket_1, ..., bucket_n$, where $bucket_i$ contains all the clauses whose *highest* literal is Q_i. For $i = n$ to 1, perform unit resolution on $bucket_i$ and perform all resolutions possible on Q_i in $bucket_i$, putting the resolvents into the proper buckets. $\mathcal{A}$ is satisfiable iff the empty clause was not generated in any bucket. Directional resolution is complete for satisfiability.

A-ordering (atom ordering) (Reynolds, 1965; Kowalski and Hayes, 1969) is a FOL version of directional resolution. It accepts a total order on literals and uses it as follows: Resolving clauses C_1, C_2 is allowed only if the resolved literal l is the *highest* literal in both[3] C_1 and C_2. Typically, the order is given over predicate symbols (instances of each predicate are equivalent for this order), but it can be given over all atoms if the order satisfies some natural properties (see (Loveland, 1978)). A-ordering is refutation-complete. It is sometimes combined with other strategies (e.g., linear resolution, semantic resolution, etc.), with modifications that allow the preservation of refutation-completeness.

Lock resolution (Boyer, 1971) is another resolution strategy for first-order theories. It generalizes both directional resolution and A-ordering. In a theory $\mathcal{A}$, every literal instance is given an index. The same literal appearing in two different clauses may receive a different index for each instance. Resolution is allowed between two clauses C_1, C_2 only upon their respectively *lowest* literals (literals with *lowest* index in the clause). The literals in the resolvent keep their original index (even if they actually changed due to unification). If two identical literals appear in the resolvent, only the one with lowest index is kept. Lock resolution is refutation-complete.

[2]This name first appeared for this strategy in (Dechter and Rish, 1994).

[3]In some places (e.g., (Chang and Lee, 1973)), the reference A-ordering is made to a modified procedure in which we are not required to resolve on the highest literal in C_2; this is not the case with the original procedure.

Note that none of these ordering-based resolution strategies is complete for consequence finding. They are not even $\mathcal{L}$-generation complete, in the general case. For example, $\{p \vee q, p \vee \neg q\} \vdash p$, but we will not conclude p if the order on propositional symbols requires our algorithm to resolve upon p first. Nevertheless, these strategies are closely related to message passing. For any given partitioned theory and graph, an order can be chosen that makes these *generation-set complete* (as introduced in Section 4.2.1), and thus suitable for inclusion in MP, for some or all the partitions. Also, for any order, both directional resolution and A-ordering can be simulated by MP, and for many indices, the lock resolution can also be simulated by MP. We first present the following useful lemma:

Lemma 4.3.6 *Let $\mathcal{R}_P$ be a FOL resolution procedure that resolves two clauses C_1, C_2 only if each of them includes a literal containing P, the two respective literals are unifiable, and the resolution is done on these literals. Let $\mathcal{L}_P$ be the language including all the symbols of $\mathcal{A}$ besides P. Then, $\mathcal{R}_P$ is $\mathcal{L}_P$-generation complete.*

PROOF See Appendix A.3.6. ■

This lemma implies that the ordering-based resolution strategies we describe above can all be used as partition reasoners in MP (and its variants). To ensure completeness of MP, we make sure that the reasoner in $\mathcal{A}_j$ is $\mathcal{L}(l(i,j))$-generation complete for the right i (determined by the query and its partition). Lemma 4.3.6 says that we only need to make sure the order is such that the symbols of $l(i,j)$ are last in the order and have equal precedence.

In the following, we consider the resolutions allowed by each of the resolution strategies, assuming they are run indefinitely. We do not consider which proved the goal first, but rather we examine the respective search spaces.

Theorem 4.3.7 (MP Simulates Orderings) *The following relationships hold between the MP algorithm and the ordering strategies of directional resolution, A-ordering and lock resolution:*

1. *Let $\mathcal{A}$ be a propositional theory and $\leq_A$ a total order on its n propositional symbols. Then, there is a partitioning $\{\mathcal{A}_i\}_{i \leq n}$ of $\mathcal{A}$, a graph G and partition reasoners that are generation-set complete such that running MP does not perform more resolutions than directional resolution (alternatively, A-ordering) of $\mathcal{A}$ with order $\leq_A$.*

2. *Let $\mathcal{A}$ be a FOL theory and $\leq_A$ a total order on its n predicate symbols. Then, there is a partitioning of $\mathcal{A}$ into $\{\mathcal{A}_i\}_{i \leq n}$, a graph G and partition reasoners that are generation-set*

complete such that running MP does not perform more resolutions than A-ordered resolution of $\mathcal{A}$ with order $\leq_A$.

3. *Let $\mathcal{A}$ be a FOL theory and I an indexing of its literal instances. Let $n = max_{l \text{ literal}} I(l)$. Assume that $I(l_1) = I(l_2)$ if l_1, l_2 have the same predicate symbol. Then, there is a partitioning $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ and partition reasoners that are generation-set complete, such that running MP does not perform more resolutions than lock resolution of $\mathcal{A}$ with index I.*

PROOF See Appendix A.3.7.

Assuming we have a propositional theory, we use unrestricted resolution for each partition in MP. Then, directional resolution and lock resolution can be used to search a proof space that is no larger than that used by MP.

Theorem 4.3.8 (Orders Simulate MP) *The following relationships hold between the MP algorithm and the ordering-based resolution strategies of directional resolution and lock resolution:*

1. *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned propositional theory and $G(V, E, l)$ be a tree that is properly labeled for $\mathcal{A}$. Then, there is a total order, $\leq_A$, on $\mathcal{A}$'s propositional symbols such that if a clause C is a consequence of directional resolution of $\mathcal{A}$ with order $\leq_A$, then C is a consequence of running MP on this partitioning using unrestricted resolution in each partition.*

2. *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned FOL theory and $G(V, E, l)$ be a tree that is properly labeled for $\mathcal{A}$. Then, there is a total order, $\leq_A$, on $\mathcal{A}$'s predicate symbols such that if a clause C is a consequence of A-ordered resolution of $\mathcal{A}$ with order $\leq_A$, then C is a consequence of running MP on this partitioning using unrestricted resolution in each partition.*

3. *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned propositional theory and $G(V, E)$ be a tree that is properly labeled for $\mathcal{A}$. Then, there is an index, I, on $\mathcal{A}$'s literal instances such that if a clause C is a consequence of lock resolution of $\mathcal{A}$ with index I, then C is a consequence of running MP on this partitioning using unrestricted resolution in each partition.*

PROOF See Appendix A.3.8.

Theorem 4.3.8 does not hold for neither A-ordered resolution or lock resolution in the FOL case, if we do not include all the function and constant symbols on the links in G. Let $C_1 = P(B, x), C_1' = P(C, y), C_2 = \neg P(z, D)$, have the partitions $\mathcal{A}_1 = \{C_1, C_1'\}$, $\mathcal{A}_2 = \{C_2\}$, and

have $k = 2$ (i.e., messages go from $\mathcal{A}_1$ to $\mathcal{A}_2$). Lock resolution would always allow resolving C_1, C_2 and C_1', C_2. MP will have the intersection language include only P, so the only message sent to $\mathcal{A}_2$ from $\mathcal{A}_1$ is $\exists b P(b, x)$. This will make $\mathcal{A}_2$ resolve only one sentence against C_2 instead of two.

4.3.5 Analysis and Comparison of Resolution-Based Inference

In this final subsection relating to resolution, we analyze the effect of FORWARD-MESSAGE-PASSING (MP) on the computational efficiency of resolution-based inference and identify some of the parameters of influence. Current measures for comparing automated deduction strategies are insufficient for our purposes. Proof length (e.g., (Tseitin, 1970; Haken, 1985; Urquhart, 1987) (also see the survey in (Cook and Mitchell, 1997))) is only marginally relevant. More relevant is comparing the sizes of search spaces induced by different strategies (e.g., resolution of propositional Horn clauses (Plaisted, 1994) and contraction rules for FOL (Bonacina and Hsiang, 1996)). These measures do not precisely address our needs, but we use them here, leaving the development of better measures for comparison to future work.

In a *resolution search space*, each node in the search space includes a set of clauses and properties relevant to the utilized resolution strategy (e.g., clause parenthood information). Each arc in the search space is a resolution step allowed by the strategy. In contrast, in an *MP resolution search space*, the nodes also include partition membership information. Further, each arc is a resolution step allowed by the utilized resolution strategy that satisfies either of: (1) the two axioms are in the same partition; or (2) one of the axioms is in partition $\mathcal{A}_j$, the second axiom is drawn from its communication language $l(i, j)$, and the query-based ordering allows the second axiom to be sent from $\mathcal{A}_i$ to $\mathcal{A}_j$. Legal sequence of resolutions correspond to paths in these spaces.

Proposition 4.3.9 *Let $\mathcal{A} = \bigcup_{i \leq n} \mathcal{A}_i$ be a partitioned theory. Any path in the MP resolution search space of $\{\mathcal{A}_i\}_{i \leq n}$ is also a path in the resolution search space of the unpartitioned theory $\mathcal{A}$.*

Evaluating MP with respect to proof length, it follows that the longest proof without using MP is as long as or longer than the longest MP proof. Unfortunately, the shortest MP proof may be longer than the shortest possible proof without MP. This observation can be quantified most easily in the simple case of only two partitions $\mathcal{A}_1, \mathcal{A}_2$. The set of messages that needs to be sent from $\mathcal{A}_1$ to $\mathcal{A}_2$ to prove Q is exactly the interpolant γ promised by Theorem 4.2.5 for $\alpha = A_1$, $\beta = \mathcal{A}_2 \Rightarrow Q$. The MP proof has to prove $\alpha \vdash \gamma$ and $\gamma \vdash \beta$.

For the propositional case, there are several results relating shortest proofs and proofs using the interpolant. Carbone (Carbone, 1997) showed that, if γ is a minimal interpolant, then for many important cases the proof length of $\alpha \vdash \gamma$, together with the proof length of $\gamma \vdash \beta$, is in $O(k^2)$ (for sequent calculus with cuts), where k is the length of the minimal proof of $\alpha \vdash \beta$. In some of these cases, the minimal interpolant is shown to be of size $O(a^2)$, where a is the sum of lengths of α, β.

In general, the size of γ itself may be large. In fact, in the propositional case it is an open question whether or not the size of the smallest interpolant can be polynomially bounded by the size of the two formulae α, β. A positive answer to this question would imply an important consequence in complexity theory, namely that $NP \cap coNP \subseteq P/poly$ (Boppana and Sipser, 1990). Nevertheless, there is a good upper bound on the length of the interpolation formula as a function of the length of the minimal proof (Krajiček, 1997): If α, β share l symbols, and the resolution proof of $\alpha \vdash \beta$ is of length k, then there is an interpolant γ of length $min(kl^{O(1)}, 2^l)$.

For the first-order case there are very few results. The best known is by Meyer (Meyer, 1980), who showed that for the first-order predicate calculus with equality, there is no recursive bound on the length of the smallest interpolant as a function of the length of the input axioms. However, there is no result relating the size of the interpolant with the length of the minimal proof (in resolution or any other proof system).

The results above suggest that we can guarantee a small interpolant, if we make sure the communication language is minimal. Unfortunately, we do not always have control over the communication language. Take, for example, the case of multiple KBs that have extensive overlap. In such cases, the communication language between KBs may be large, possibly resulting in a large interpolant. In Section 5.4 we provide an algorithm for partitioning theories that attempts to minimize the communication language between partitions.

Finally, the results cited above combined with the results of Section 4.3.4, show a potential limitation of variable ordering for resolution in general. Variable ordering can be simulated by MP (Theorem 4.3.7). If the interpolants needed for the proof with MP are large, variable ordering (as in directional resolution) may lead us to spend significantly more time than we could have spent had we not used that order (or any other order). However, this observation depends on bounds on the potential size of the interpolant. Finding those bounds is a known open problem.

4.4 Partition-Based Propositional Satisfiability

In this section we propose an algorithm for partition-based logical reasoning based on propositional satisfiability (SAT) search (e.g., DPLL (Davis et al., 1962), GSAT (Selman et al., 1992) and WALK-SAT (Selman et al., 1994)). We show that the complexity of computation is directly related to the size of the labels in the intersection graph.

4.4.1 A Partition-Based SAT Procedure

The algorithm we propose uses a SAT procedure as a subroutine and is backtrack-free. We describe the algorithm using database notation (Ullman, 1988). $\pi_{p_1,...,p_k}T$ is the *projection* operation on a relation T. It produces a relation that includes all the rows of T, but only the columns named $p_1, ..., p_k$ (suppressing duplicate rows). $S \bowtie R$ is the *natural join* operation on the relations S and R. It produces the cross product of S, R, selecting only those entries that are equal between identically named fields (checking $S.A = R.A$), and discarding those columns that are now duplicated (e.g., $R.A$ will be discarded).

The proposed algorithm shares some intuition with prime-implicate generation (e.g., (Marquis, 1995; Inoue, 1992)). First, the algorithm computes $L(i)$, the set of symbols on all of $\mathcal{A}_i$'s links, in Step 2. Then, it finds out which truth values of $L(i)$ are satisfiable (akin to computing the implicates of each partition in the language $\mathcal{L}(L(i))$). Finally, the algorithm uses $\bowtie$ to combine those values to find out if there are any models for $\mathcal{A}$. The algorithm is presented in Figure 4.14.

This algorithm resembles finding all the models of each partition and then joining the consistent interpretation fragments into models for $\mathcal{A}$ (as done in (Dechter and Pearl, 1989)). The iterated join that we perform takes time proportional to the size of the tables involved. In contrast to (Dechter and Pearl, 1989), we keep table sizes below $2^{|L(i)|}$ by keeping only the consistent truth assignments for $L(i)$ (instead of all the models of $\mathcal{A}_i$) and *projecting* every table before *joining* it with another table. Figure 4.15(a) displays the result of applying LINEAR-PART-SAT up to Step 3 to the partitioned theory and input of Figure 4.4. Figures 4.15(b) and 4.15(c) show the progression of Step 5 of LINEAR-PART-SAT.

Soundness and completeness follow by an argument similar to that given for MP.

Theorem 4.4.1 (Soundness and Completeness) *Given a sound and complete SAT-search procedure, LINEAR-PART-SAT is sound and complete for SAT of partitioned propositional theories.*

PROOF See Appendix A.3.9.

PROCEDURE LINEAR-PART-SAT($\{\mathcal{A}_i\}_{i\leq n}$)
$\{\mathcal{A}_i\}_{i\leq n}$ a partitioning of the theory $\mathcal{A}$,

1. $G_0 \leftarrow$ the intersection graph of $\{\mathcal{A}_i\}_{i\leq n}$. $G \leftarrow BREAK\text{-}CYCLES(G_0)$.
2. For each $i \leq n$, let $L(i) = \bigcup_{(i,j)\in E} l(i,j)$.
3. For each $i \leq n$, for every truth assignment A to $L(i)$, perform SAT-search on $\mathcal{A}_i \cup A$, storing the result in a table $T_i(A)$.
4. Let $dist(i,j)$ $(i,j \in V)$ be the length of the shortest path between i,j in G. Let $i \prec j$ iff $dist(i,1) < dist(j,1)$ ($\prec$ is a strict partial order).
5. Iterate over $i \leq n$ in reverse $\prec$-order (the last i is 1). For each $j \leq n$ that satisfies $(i,j) \in E$ and $i \prec j$, perform:
   - $T_i \leftarrow T_i \bowtie (\pi_{L(i)} T_j)$ (*Join* T_i with those columns of T_j that correspond to $L(i)$). If $T_i = \emptyset$, return FALSE.
6. Return TRUE.

Figure 4.14: An algorithm for SAT of a partitioned propositional theory.

4.4.2 Analyzing Satisfiability in LINEAR-PART-SAT

Let $\mathcal{A}$ be a partitioned propositional theory with n partitions. Let $m = |L(\mathcal{A})|$, $L(i)$ be the set of propositional symbols calculated in Step 2 of LINEAR-PART-SAT, and $m_i = |L(\mathcal{A}_i) \setminus L(i)|$ ($i \leq n$). Let $a = |\mathcal{A}|$ and k be the length of each axiom.

Lemma 4.4.2 *The time taken by LINEAR-PART-SAT to compute SAT for $\mathcal{A}$ is*

$$Time(n, m, m_1, ..., m_n, a, k, |L(1)|, ..., |L(n)|) = O(a * k^2 \ + \ n^4 * m \ + \ \textstyle\sum_{i=1}^{n} (2^{|L(i)|} * f_{SAT}(m_i))),$$

where f_{SAT} is the time to compute SAT. If the intersection graph G_0 is a tree, the second argument in the summation can be reduced from $n^4 * m$ to $n * m$.

PROOF See Appendix A.3.10.

Corollary 4.4.3 *Let $\mathcal{A}$ be a partitioned propositional theory with n partitions, m propositional symbols and intersection graph $G = (V, E, l)$. Let $d(v)$ be the degree of node v in the graph $G(V, E, l)$, let $d = max_{v\in V} d(v)$ and let $l = max_{i,j\leq n} |l(i,j)|$. Assume $P \neq NP$. If intersection*

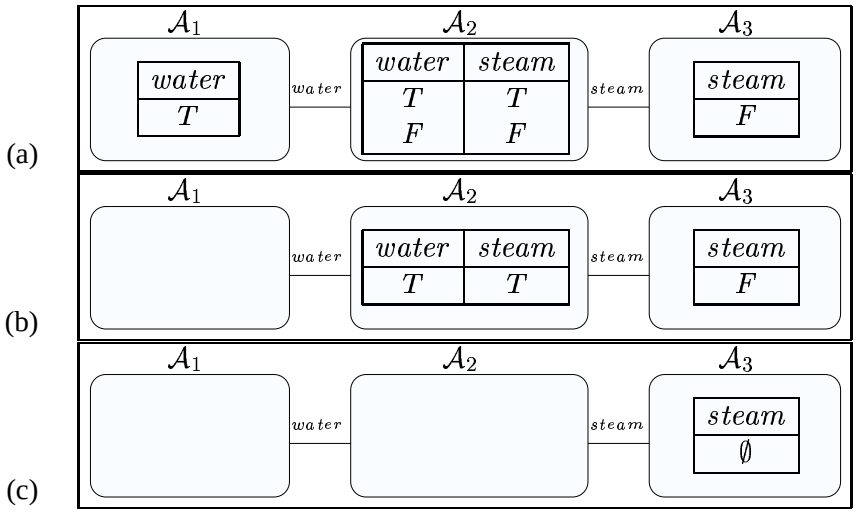

Figure 4.15: Iteratively projecting and joining tables to check satisfiability.

graph G of $\mathcal{A}$ is a tree and all the partitions $\mathcal{A}_i$ have the same number of propositional symbols, then the time taken by the LINEAR-PART-SAT procedure to compute SAT for $\mathcal{A}$ is

$$Time(m, n, l, d) = O(n * 2^{d*l} * f_{SAT}(\frac{m}{n})).$$

For example, if we partition a given theory $\mathcal{A}$ into only two partitions ($n = 2$) sharing l propositional symbols, the algorithm will take time $O(2^l * f_{SAT}(\frac{m}{2}))$. Assuming $P \neq NP$, this is a significant improvement over a simple SAT procedure for every l that is small enough ($l < \frac{\alpha m}{2}$, and $\alpha \leq 0.582$ (Schiermeyer, 1996; Cook and Mitchell, 1997)).

4.4.3 Order for Dynamic Backtracking

One way to apply our decomposition algorithm is to order the variables for backtracking SAT procedures (e.g., DPLL (Davis et al., 1962), dependency-directed backtracking (Stallman and Sussman, 1977) or dynamic backtracking (Ginsberg, 1993)). Those variables participating in the partitions that are leaves of the tree of partitions should come first, then those appearing in parent nodes in the tree, and so forth.

Figure 4.16 presents a simple algorithm that orders the variables for dynamic backtracking (Ginsberg, 1993). Order-Backtrack takes as input a partitioned propositional clausal theory $\mathcal{A}$, a

graph G and a parameter $k \leq n$. It sorts the partitions according to G and sorts the variables accordingly. It uses this order on variables to call a dynamic backtracking SAT-search procedure. It returns a model of $\mathcal{A}$ or FALSE.

PROCEDURE Order-Backtrack($\mathcal{A}$, G, k) $\mathcal{A} = \{\mathcal{A}_i\}_{i \leq n}$ is a propositional clausal theory, $G = (V, E, l)$ is an undirected graph describing the connections between the partitions, and $k \leq n$ is the (suggested) root of the tree.

1. Let $dist(i, j)$ $(i, j \in V)$ be the length of the shortest path between i, j in G. Let $i \prec j$ iff $dist(i, 1) < dist(j, 1)$ ($\prec$ is a strict partial order).
2. Let $i_1, ..., i_n$ be nodes in V such that $i_j \not\prec i_{j'}$ for all j', j such that $n \geq j' > j \geq 1$.
3. For all $q \in L(\mathcal{A})$, let $P(q)$ be an index $i = i_j$ such that $q \in L(\mathcal{A}_i)$ and $\forall j' < j \; q \notin L(\mathcal{A}_{i_{j'}})$.
4. Sort $L(\mathcal{A})$ into $q_1, ..., q_m$ such that if $a < b \leq m$, then $P(q_a) \leq P(q_b)$.
5. Return the result of running dynamic-backtracking on $\mathcal{A}$ with the order $q_1, ..., q_n$ on the propositional symbols.

Figure 4.16: An algorithm for SAT of a propositional theory, using decomposition and order over propositions.

Using order the way that we present here is common in Constrain Satisfaction Problems (CSPs) and propositional satisfiability solutions. In particular, (Dechter, 1999; Dechter, 1996) shows how graphical models for satisfiability testing, CSPs and Bayes nets inference are closely related to order on nodes for those inference methods (Section 4.3 shows that this is not the case in general for theorem proving in propositional logic or FOL).

4.5 Experimental Results

We have experimented with only a small number of data sets with the algorithms that we presented in this chapter. The results so far are mixed and leave much work for the future.

The LINEAR-PART-SAT algorithm (Section 4.4.1) was implemented by Paul Ruhlen and tested on a number of SAT problems from the DIMACS test suite (SATLIB at http://www.satlib.org). This implementation is not distributed and does not use parallel processing. At the time of writing of this thesis, the results are mixed. The decompositions that we achieved for those problems (using the methods of Chapter 5 and others) have somewhat large communication bandwidths. This requires us to use more sophisticated methods in order to make sure that the tables used in this algorithm are not too large. So far, the approach was able to solve only a small problem of the size of 116 propositional

symbols and 953 clauses (in about a minute), and the implementation ran out of memory for a problem of size 776 symbols and 3725 clauses).

The MP algorithm (Section 4.2.1) has been implemented by the author of this thesis for a special case in which we know the form of the messages that need to be sent between partitions. We know that the only message that needs to be sent from one partition to another is a ground formula, and we also know the matrix of this formula. In this case, we were able to show that running the PTTP (Stickel, 1992; Stickel, 1988b; Stickel, 1988a) theorem prover on the entire theory does not return answers after an hour while running MP with PTTP in each partition yields answers in a matter of seconds (these partitions were created by hand, not using the algorithms of Chapter 5). This implementation is distributed (see more details in Chapter 6, Section 6.6).

A more general implementation of the MP algorithm with A-ordered resolution in each partition has been written by Mark Stickel and Tomas Uribe for SRI's Snark theorem prover. Experiments with this suite are forthcoming.

4.6 Related Work: Structure in Reasoning

The related work can be divided into two parts. First is the use of decompositions for propositional reasoning; second is the work that relates to FOL theorem proving.

4.6.1 Use of Decompositions in Propositional SAT

With respect to propositional SAT problems, perhaps the most relevant previous work is that of Dechter and Pearl (Dechter and Pearl, 1989), which presented algorithms for reasoning with decomposed CSPs. These can be used for SAT under a given decomposition. In comparison, the algorithm we presented for partition-based SAT does not produce all the models possible in each partition, as proposed in (Dechter and Pearl, 1989). Instead, it finds the truth values for propositions on the links that are extendible to a satisfying truth assignment for the whole partition. This reduces our computation time and makes it more dependent on the links' sizes rather than on partition sizes.

Other SAT uses of SAT decomposition include (Park and Gelder, 1996), which proposed a decomposition procedure that represents the theory as a hypergraph of clauses and divides the propositional theory into two partitions (heuristically minimizing the number of hyperedges). It finds the set of possible truth-value assignments to the propositions associated with the hyper-edges and tests them recursively for the two partitions. This is a modification of ideas described in (Gallo and Urbani, 1989). Cowen and Wyatt (Cowen and Wyatt, 1993) developed an algorithm that partitions

a propositional CNF theory into connected components that can be tested for satisfiability individually. Their partitioning algorithm is an adaptation of a best first search as used to find components in a graph or strongly connected components in a digraph. The authors tested their decomposition algorithm together with a SAT solver, demonstrating a dramatic decrease in the runtime of the SAT solver on the decomposed theories described in that paper.

Prior to (Amir and McIlraith, 2000), there was no work on using decompositions for automated deduction in propositional logic in the manner we propose. Concurrent to this work, Rish and Dechter (Rish and Dechter, 2000) proposed an algorithm similar to our MP algorithm for the case of propositional ordered resolution. However, their work looks at a limited case (ordered resolution, propositional logic), and they allow excessive computation by performing all possible resolutions in each partition, twice. Our MP algorithm is opportunistic in the sense that it does not wait for each partition to perform all of the possible resolutions. (In FOL this is not possible at all.) Thus, Rish and Dechter's algorithm may use exponential amounts of space and time over and above MP in the same settings.

4.6.2 Use of Decompositions in FOL Theorem Provers

Surprisingly, there has been little work on the specific problem of exploiting structure in theorem proving in the manner we propose in this chapter. We conjecture that this can largely be attributed to the fact that theorem proving has traditionally examined mathematics domains, which do not necessarily have structure that supports decomposition. Nevertheless, there is related work both in the parallel theorem proving community and in the work on combining logical systems.

The majority of work on parallel theorem proving implementations followed decomposition of the search space (Conry et al., 1990; Ertel, 1992; Sutcliffe, 1992; Cowen and Wyatt, 1993; Bonacina and Hsiang, 1996; Suttner, 1997) or allowed messages to be sent across the different provers working in parallel, using heuristics to decide on what messages are relevant to each prover (Denzinger, 1995; Denzinger et al., 1997; Denzinger and Fuchs, 1999) (surveys can be found in (Bonacina and Hsiang, 1994; Denzinger and Dahn, 1998)). Both approaches typically look at decompositions into very few sub-problems (typically fewer than ten). In addition, the first approach typically requires complete independence of the sub-spaces or the search is repeated on much of the space by several reasoners. The second approach is more similar to ours, but there are some major differences still. First, there is no clear methodology for deciding what messages should be sent from one partition to another or which partitions should receive messages from which other partitions. Second, there are no clear criteria for decomposing a theory into sub-problems.

Another line of work focuses on combining logical systems (Nelson and Oppen, 1979; Shostak, 1984; Baader and Schulz, 1992; Ringeissen, 1996; Tinelli and Harandi, 1996). Here, the computational focus has been on treating combinations of signature-disjoint theories (allowing the queries to include symbols from all signatures), such as in (Baader and Schulz, 1992). Recent work introduced the sharing of function symbols between two theories (e.g., (Ringeissen, 1996)), but no algorithm allowed any sharing of relation symbols. All approaches either nondeterministically instantiate the (newly created) variables connecting the theories (e.g., (Tinelli and Harandi, 1996)) or restrict the theories to be convex (disjunctions are intuitionistic) and have information flowing back and forth between them. In contrast, we focus on the structure of interactions between theories with signatures that share symbols and the efficiency of reasoning with consequence finders, theorem provers and SAT procedures. We do not have any restrictions on the language besides finiteness.

Finally, work on formalizing and reasoning with *context* (e.g., (McCarthy and Buvač, 1998; Akman and Surav, 1996)) can be related to partition-based logical reasoning by viewing the contextual theories as interacting sets of theories. Unfortunately, a language that is more expressive than FOL is needed to introduce explicit contexts. Consequently, a number of researchers have focused on context for propositional logic, while much of the reasoning work has focused on proof-checking (e.g., GETFOL (Giunchiglia and Traverso, 1995; Giunchiglia, 1994)). There have been few reported successes with automated reasoning; (Bonzon, 1997) presents one example.

4.7 Conclusions

In this chapter we have shown that decomposing theories into partitions and reasoning over those partitions has potential computational advantages for theorem provers and SAT solvers. Theorem proving strategies, such as resolution, can use such decompositions to constrain search. Partition-based reasoning will improve the efficiency of propositional SAT solvers if the theory is decomposable into partitions that share only small numbers of symbols.

We have provided sound and complete algorithms for reasoning with partitions of related logical axioms, both in propositional logic and in FOL. Different reasoning algorithms can be plugged in for different partitions in these algorithms. We gave conditions on those reasoners that ensure that the combined reasoning procedure is sound and complete. Specialized versions of these algorithms for resolution strategies in FOL were provided. We showed that some of these algorithms simulate some order-based resolution strategies, while order-based strategies may simulate some of our algorithms in restricted cases. All our reasoning algorithms are suited for parallel processing.

We examined the efficiency of our theorem-proving algorithms and our SAT algorithm. Guided by both analyses, we suggested guidelines for achieving a good partitioning and proposed an algorithm for the automatic decomposition of theories that tries to minimize identified parameters. This algorithm generalizes previous algorithms used to decompose CSPs by finding single-vertex separators.

Our work was motivated in part by the problem of reasoning with large multiple KBs that have overlapping content. The results in this thesis address some of the theoretical principles that underlie such partition-based reasoning. We are currently performing experimental analysis of Stanford KSL's and SRI's KBs to analyze structure in these KBs, to test the effectiveness of our automatic partitioning algorithms and to investigate the effectiveness of proposed partition-based reasoning algorithms. We are also involved in further theoretical investigation.

Chapter 5

Automatic Decomposition of Logical Theories

5.1 Introduction

The algorithms presented in Chapter 4 assumed a given partitioning. In this chapter we address the critical problem of automatically decomposing a set of propositional or FOL clauses into a partitioned theory. Guided by the results of previous sections, we propose guidelines for achieving a good partitioning and present several algorithms that decomposes a theory following these guidelines. We look at a simple greedy method for decomposition that attempts to optimize the result according to the guidelines that we identify. Then, we show how the problem of automated decomposition can be solved using algorithms that find triangulations of minimum treewidth.

Given an undirected graph, G, and an integer k, TREEWIDTH is the problem of deciding whether the treewidth of G is at most k (Robertson and Seymour, 1986). An equivalent constructive problem is finding a *triangulation* of G with a clique number that is at most $k+1$ (the clique number of a graph is the size of the largest clique in this graph). This is also equivalent to finding a *tree decomposition* or a *junction tree* of G with width at most k.

Efficiently soluving this problem is key in many applications in artificial intelligence, databases and logical-circuit design. Exact inference in Bayesian networks using the junction tree algorithm (Lauritzen and Spiegelhalter, 1988) requires us to first find a junction tree and then perform inference using that tree. The time complexity of the junction tree algorithm depends exponentially on the width of the tree, so it is important to try to find a close to optimal clique tree. Reasoning with

structured CSPs, propositional SAT and FOL problems also benefits from efficiently finding close-to-optimal tree decompositions, as seen in (Dechter and Pearl, 1989) and Chapter 4. The solution time of many graph-related NP-hard problems is possible in polynomial time if the graph has low treewidth and a triangulation of minimum treewidth is given (e.g., (Arnborg et al., 1991)).

In this chapter, following the introduction of our simply greedy algorithm, we present four approximation algorithms for finding triangulations of minimum treewidth. The first algorithm improves an algorithm of (Robertson and Seymour, 1986) and produces factor-4 approximations in time $O(2^{4.38k}n^2k^2)$, where n, k are the number of nodes and the treewidth of the given graph, G, respectively. The second algorithm improves an algorithm of (Becker and Geiger, 1996) and produces factor-$3\frac{2}{3}$ approximations in time $O(2^{3.6982k}n^3k^3lg^4n)$. The third algorithm produces factor-$4\frac{1}{2}$ triangulations in time $O(2^{3k}n^2k^{\frac{5}{2}})$. The last algorithm produces produces a factor-$O(lgk)$ approximation in time $O(n^3lg^4nk^5\,lgk)$.

The time bounds achieved by the first and second algorithms are faster by factors of $O(2^{0.4k})$ and $O(2^k poly(n))$, respectively, than previously available algorithms for these approximation factors. The third algorithm has faster combined n, k time than any known algorithm that produces a constant-factor approximation. The last algorithm is the first polynomial-time algorithm that approximates the optimal by a factor that does not depend on n. We have implemented the 4-approximation and $4\frac{1}{2}$-approximation algorithms and used them to find tree decompositions of graphs used in the a subset of the HPKB project (Cohen et al., 1999), a subset of the CYC knowledge base (Lenat, 1995), and several CPCS Bayesian networks (Pradhan et al., 1994). These graphs have between 100 and 600 nodes and between 400 and 4000 edges. Our results compare favorably with the algorithms of (Becker and Geiger, 1996; Shoikhet and Geiger, 1997).

The chapter is organized as follows. Section 5.2 provides guidelines for finding good decompositions. Section 5.4 describes algorithm SPLIT, a simple greedy algorithm that decompose a theory into a graph and recursively splits the graph in a way that the resulting decomposition is a tree. Section 5.5 defines the main notions involved in computing treewidth and recalls some theorems proved elsewhere. Section 5.6 presents our 4- and $4\frac{1}{2}$- approximation algorithms. Section 5.7 presents our $3\frac{2}{3}$- and $O(lgk)$- approximation algorithms. The chapter concludes with experimental results and related work. A good survey paper on TREEWIDTH is (Bodlaender, 1997). A good book on the subject is (Kloks, 1994).

There are several notions from the graph algorithms literature that we use in this chapter. We repeat the most important notions, assuming only basic knowledge of graph theory. The reader is referred to (Cormen et al., 1989) for further background information. Most of the results in this

chapter appeared previously in (Amir and McIlraith, 2000; Amir, 2001).

5.2 What is a Good Partitioning?

The analysis done in Section 4.4.2 in Chapter 4 does not assume any particular time complexity for $f_{SAT}(m)$ (aside from $P \neq NP$ in the corollary). If we assume that $f_{SAT}(m) = \Theta(2^m)$, then we can conclude that the time for our reasoning algorithm is dominated by the largest partition (including its links). If the largest partition is of size s (i.e., it has s propositional symbols in its language, link languages included), then the time for the algorithm is $O(n * 2^s)$. This is the analysis done for CSPs and Bayes networks (e.g., (Dechter and Pearl, 1989; Dechter and Rish, 1994; Rish and Dechter, 2000; Becker and Geiger, 1996)), where the utilized algorithms do in fact use time $\Theta(2^m)$ for a problem with m variables. For satisfiability the situation is slightly different. There are known stochastic algorithms (e.g., (Selman et al., 1992; Selman et al., 1994)) that perform much better than this pessimistic forecast. These algorithms are not complete, but they can be used in our algorithm, if we are willing to give up completeness. Efficient complete algorithms also typically exhibit better than worst-case behavior (see the analysis of (Selman et al., 1997)). All of this suggests that emphasizing link sizes together with partition sizes is more accurate for the satisfiability problem.

Thus, given a theory, we wish to find a partitioning that minimizes the formula derived in Lemma 4.4.2. To that end, assuming $P \neq NP$, we want to minimize the following parameters in roughly the following order. For all $i \leq n$:

1. $|L(i)|$ - the total number of symbols contained in all links to/from node i. If G_0 is already a tree, this is the number of symbols shared between the partition $\mathcal{A}_i$ and the rest of the theory $\mathcal{A} \setminus \mathcal{A}_i$.

2. m_i - the number of symbols in a partition, less those in the links, i.e., in $\mathcal{A}_i$ but not in $L(i)$. This number is mostly influenced by the size of the original partition $\mathcal{A}_i$, which in turn is influenced by the number of partitions of $\mathcal{A}$, namely, n. Having more partitions will cause m_i to become smaller.

3. n - the number of partitions.

Also, a simple analysis shows that given *fixed* values for l, d in Corollary 4.4.3, the maximal n that maintains l, d such that also $n \leq ln2 * \alpha * m$ ($\alpha = 0.582$ (Schiermeyer, 1996; Cook and

Mitchell, 1997)) yields an optimal bound for LINEAR-PART-SAT. In Section 4.3.5 we saw that the same parameters influence the number of derivations we can perform in MP: $|L(i)|$ influences the interpolant size and thus the proof length, and m_i influences the number of deductions/resolutions we can perform. Thus, we would like to minimize the number of symbols shared between partitions and the number of symbols in each partition less those in the links.

The question is, how often do we get large n (many partitions), small m_i's (small partitions) and small $|L(i)|$'s (weak interactions) in practice. We believe that in domains that deal with engineered physical systems, many of the domain axiomatizations have these structural properties. Indeed, design of engineering artifacts encourages modularization, with minimal interconnectivity (see (Amir, 2000a; Lenat, 1995; Cohen et al., 1998)). More generally, we believe axiomatizers of large corpora of real-world knowledge tend to try to provide structured representations following some of these principles. Recent experiments with the HPKB knowledge base of SRI and a part of the Cyc knowledge base support this belief (those experiments are reported in Section 5.8).

5.3 Principles of Partitioning Logical Theories

To exploit the partitioning guidelines proposed in Section 5.2, we represent our theory $\mathcal{A}$ using a *symbols graph* that captures the features we wish to minimize. $G = (V, E)$ is a symbols graph for theory $\mathcal{A}$ such that each vertex $v \in V$ is a symbol in $\mathcal{L}(\mathcal{A})$, and there is an edge between two vertices if their associated symbols occur in the same axiom of $\mathcal{A}$, i.e., $E = \{(a, b) \mid \exists \alpha \in \mathcal{A}$ s.t. a, b appear in $\alpha\}$.

Figure 5.2 (top) illustrates the symbols graph of theory $\mathcal{A}$ from Figure 5.1 and the connected symbols graphs (bottom) of the individual partitions $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$. Notice that each axiom creates a clique among its constituent symbols. To minimize the number of symbols shared between partitions (i.e., $|L(i)|$), we must find partitions whose symbols have minimal *vertex separators* in the symbols graph.

Generally speaking, we decompose theory $\mathcal{A}$ by first creating the symbols graph of $\mathcal{A}$, then partition this graph into partitions (similar to the bottom part of Figure 5.1), and finally use the partitioning of the graph to define a partitioning of the axioms. Section 5.4 presents a complete algorithm for this task.

Examples of the recursive procedure that we are going to use throughout this chapter are presented in Figure 5.5 on page 93 and Figure 5.6 on page 96.

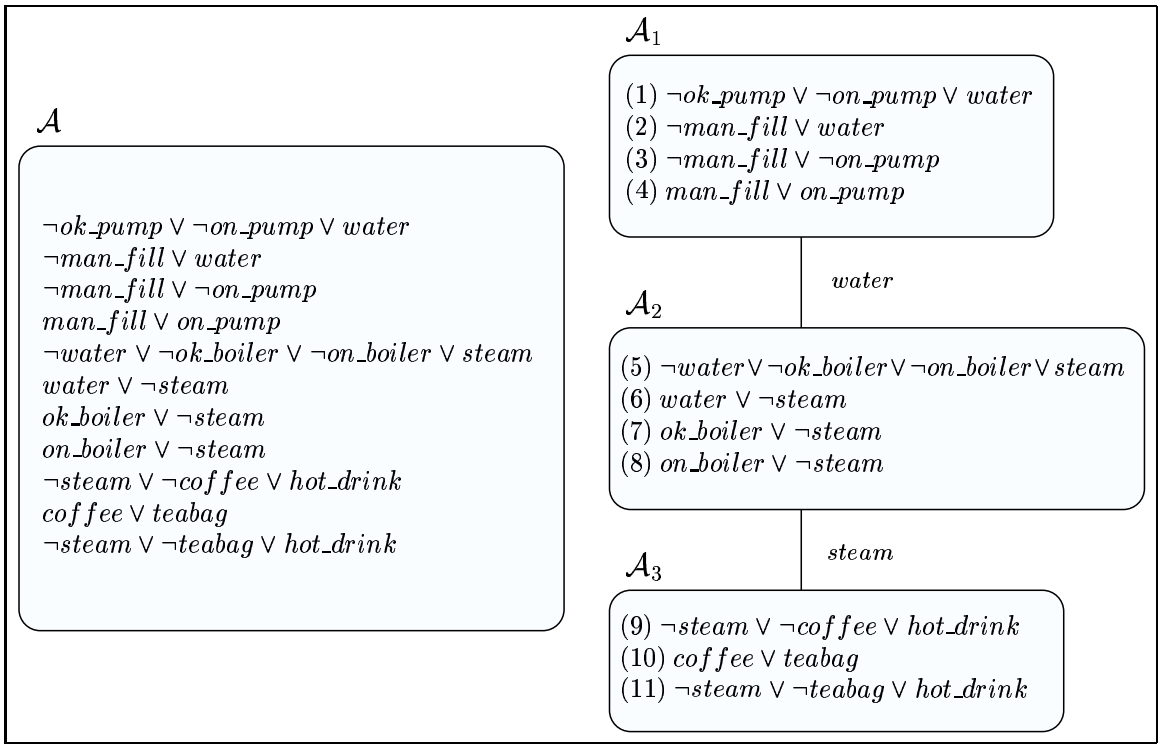

Figure 5.1: A partitioning of $\mathcal{A}$ and its intersection graph.

5.4 Split: Greedy Vertex Min-Cut in the Graph of Symbols

5.4.1 Minimum Vertex Separators

We briefly describe the notion of a vertex separator. Let $G = (V, E)$ be an undirected graph. A set S of vertices is called an (a, b)-*vertex-separator* if $\{a, b\} \subset V \setminus S$ and every path connecting a and b in G passes through at least one vertex contained in S.

Let $N(a, b)$ be the least cardinality of an (a, b)-vertex-separator. The *connectivity* of the graph G is the minimum $N(a, b)$ for any $a, b \in V$ that are not connected by an edge. An (a, b)-vertex-separator of minimum cardinality is said to be a *minimum* (a, b)*-vertex-separator*. The weaker property of a vertex separator being *minimal* requires that no subset of the (a, b)-vertex-separator is an (a, b)-vertex-separator.

We briefly review an algorithm by Even and Tarjan (Even, 1979; Even and Tarjan, 1975) for finding minimum vertex separators, building on (Dantzig and Fulkerson, 1956). It is shown in Figure 5.3. The algorithm is given two vertices, a, b, and an undirected graph, G. It transforms

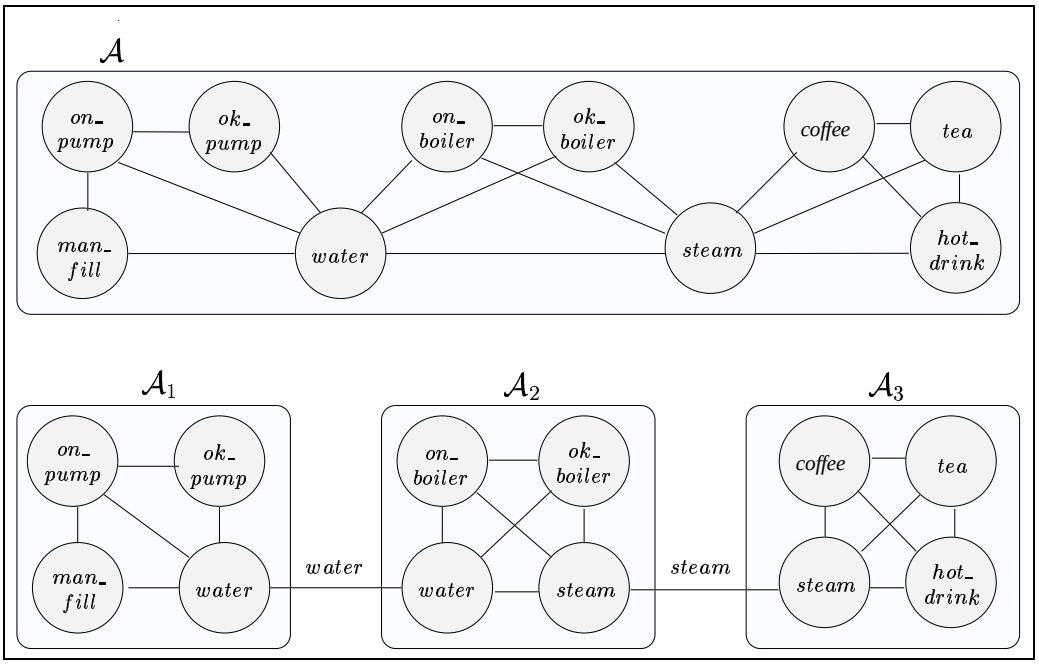

Figure 5.2: Decomposing $\mathcal{A}$'s symbols graph.

G into a directed graph, $\bar{G}$, that has two vertices (corresponding to *input* and *output*) for each original vertex of G, directed edges connecting the corresponding input and output vertices, and edges corresponding to those of G, but only from output to input vertices. It then runs a max-flow algorithm on $\bar{G}$ (lines 1–3). The produced flow, f, has a throughput of $N(a, b)$. To extract a minimum separator, it produces a *layered network* (see (Even, 1979) p.97) from $\bar{G}$ and the flow found, f, in line 5. The layered network includes a subset of the vertices of $\bar{G}$. The set of edges between this set of vertices and the rest of $\bar{G}$ corresponds to the separator.

Algorithms for finding maximal flow are abundant in the graph algorithms literature. Prominent algorithms for max-flow include the Simplex method, Ford and Fulkerson's (Ford Jr. and Fulkerson, 1962), the push-relabel method of Goldberg and Tarjan (Goldberg and Tarjan, 1988) (time bound of $O(|V| \cdot |E| \cdot log\frac{|V|^2}{|E|})$ and several implementations (Chekassky and Goldberg, 1997)), and Dinitz's algorithm (Dinic, 1970). When Dinitz's algorithm is used to solve the network problem, Even and Tarjan's algorithm has time complexity $O(|V|^{\frac{1}{2}}|E|)$ (Even, 1979). The unit-capacity network-flow algorithm of (Ahuja and Orlin, 1991) can also be used here, giving Even and Tarjan's algorithm time complexity of $O(|V|^{\frac{1}{2}}|E|)$ as well.

Another possibility is to use the Ford-Fulkerson flow algorithm (Ford Jr. and Fulkerson, 1962)

PROCEDURE MIN-V-SEP-A-B($G = (V, E)$, a, b)

1. Construct a digraph $\bar{G}(\bar{V}, \bar{E})$ as follows. For every $v \in V$ put two vertices v', v'' (*input/output vertices*) in $\bar{V}$ with an edge $e_v = \overrightarrow{(v', v'')}$ (*internal edge*). For every edge $e = (u, v)$ in G, put two edges $e' = \overrightarrow{(u'', v')}$ and $e'' = \overrightarrow{(v'', u')}$ in $\bar{G}$ (*external edges*).
2. Define a network, with digraph $\bar{G}$, source a'', sink b' and unit capacities for all the edges.
3. Compute the *maximum flow* f in the network.
4. Set the capacities of all the external edges in $\bar{G}$ to infinity.
5. Construct the *layered network* $\{V_i\}_{i \leq l}$ from $\bar{G}$ using f. Let $S = \bigcup_{i \leq l} V_i$.
6. Let $R = \{v \in V \mid v' \in S,\ v'' \notin S\}$. R is a minimum (a, b) vertex-separator in G.

Figure 5.3: An algorithm for finding a minimum separator between a and b in G.

(alternatively, see (Cormen et al., 1989)), for computing maximum flow. For an original graph of treewidth $< k$ (treewidth is defined in Section 5.5) this involves finding at most k augmenting paths of capacity 1. Thus, the combined algorithm using the Ford-Fulkerson maximum flow algorithm finds a minimum (a, b)-vertex-separator in time $O(k(|V| + |E|))$.

Finally, to compute the vertex connectivity of a graph and a minimum separator, without being given a pair (a, b), we check the connectivity of any c vertices (c being the connectivity of the graph) to all other vertices. When Dinitz's algorithm is used as above, this procedure takes time $O(c \cdot |V|^{\frac{3}{2}} \cdot |E|)$, where $c \geq 1$ is the connectivity of G. When we use Ford-Fulkerson's algorithm for a graph of treewidth k, this procedure takes time $O(c \cdot k \cdot |V| \cdot (|V| + |E|))$, where $c \geq 1$ is the connectivity of G. For the cases of $c = 0, 1$ there are well known linear time algorithms. (Even, 1975) also showed a way to test for k connectivity of a graph using only $n + k^2$ pairs of vertices.

5.4.2 Procedure *Split*

Procedure Split-Thy, presented in Figure 5.4, uses procedure Split to decompose a theory into a tree of partitions. It is given a theory [1], $\mathcal{A}$, and limitations on the partition size (a lower limit, M) and the separators between partitions (an upper limit, l). Split initially considers the theory as one big partition, and at every recursive iteration it breaks one of the partitions in two. It represents the tree structure of the partitions in a global variable, G_{str}. This tree structure and the set of partitions, $\{\mathcal{A}_i\}_{i \leq p}$, is returned as the result of Split-Thy. An example of the input and the output is shown in Figure 5.1.

Split partitions the theory $\mathcal{A}$ by taking as input its symbols graph, $G = (V, E)$, the two limiting

[1] Recall that in this thesis, a theory is a set of propositional clauses.

PROCEDURE Split-Thy($\mathcal{A}$, M, l)
$\mathcal{A}$ is a theory. M limits the number of symbols in a partition from below. l limits the number of symbols shared between partitions.

1. Let $G(V, E)$ be an undirected graph with $V = L(\mathcal{A})$ and $E = \{(l_1, l_2) \mid \exists C \in \mathcal{A}\ l_1, l_2 \in L(C)\}$.
2. Let $G_{str}(V_{str}, E_{str})$ be an undirected graph with $V_{str} = \{\{V\}\}$ and $E_{str} = \emptyset$.
3. Run Split(G, M, l, nil, nil).
4. For every $v \in V_{str}$, let $\mathcal{A}_v = \{C \in \mathcal{A} \mid L(C) \subset v\}$. Return $\{\mathcal{A}_v\}_{v \in V_{str}}$ and G_{str}.

PROCEDURE Split(G, M, l, a, b)
$G = (V, E)$ is an undirected graph. M, l as above. a, b are in V or are nil.

1. If $|V| < M$, then return V.
2. (a) If $a, b{=}nil$, find a minimum vertex separator, R, in G. (b) Otherwise, if $b{=}nil$, find a minimum vertex separator, R, of a in G. (c) Otherwise, find minimum vertex separators, R_a of a in G, and R_b of b in G. Let R be the smaller of R_a, R_b.
3. If $R = V$ or $|R| > l$ then return V.
4. Let G_1, G_2 be the two subgraphs of G separated by R, with R included in both subgraphs.
5. Let $V_{str} \leftarrow V_{str} \setminus \{\{V\}\} \cup \{\{V_1\}, \{V_2\}\}$ and $E_{str} \leftarrow E_{str} \cup \{(\{V_1\}, \{V_2\})\}$. Change the edges that connected to $\{V\}$ to connect to one of $\{V_1\}, \{V_2\}$.
6. Create G'_1, G'_2 from G_1, G_2, respectively, by aggregating the vertices in R into a single vertex r, removing all self edges and connecting r with edges to all the vertices connected by edges to some vertices in R.
7. Run $Split2(G'_1, M, l, r, a), Split2(G'_2, M, l, r, b)$. Replace r in the nodes of V_{str} by the members of R.

Figure 5.4: An algorithm for generating partitions of axioms.

parameters, M and l, and nodes $a, b \in V$ that are initially set to nil. Split updates the global variable G_{str} to represent the progressing decomposition. In each recursive call, Split finds a minimum vertex separator of a, b in G (i.e., a minimum-size set of vertices that crosses every path between a, b). If one of a, b or both are nil, it finds the overall minimum vertex separator between all vertices and the non-nil vertex (or all other vertices). This separator splits G into two graphs, G_1, G_2, and the process continues recursively. An example of the progress made on the input graph G is shown in Figure 5.5.

Different variants of the algorithm yield different structures for the intersection graph of the resulting partitioning. As is, Split returns sets of symbols that result in a chain of partitions. We obtain arbitrary *trees*, if we change step 3(c) to find a minimum separator that does not include a, b (not required to separate a, b). We obtain arbitrary *graphs*, if in addition we do not aggregate R into

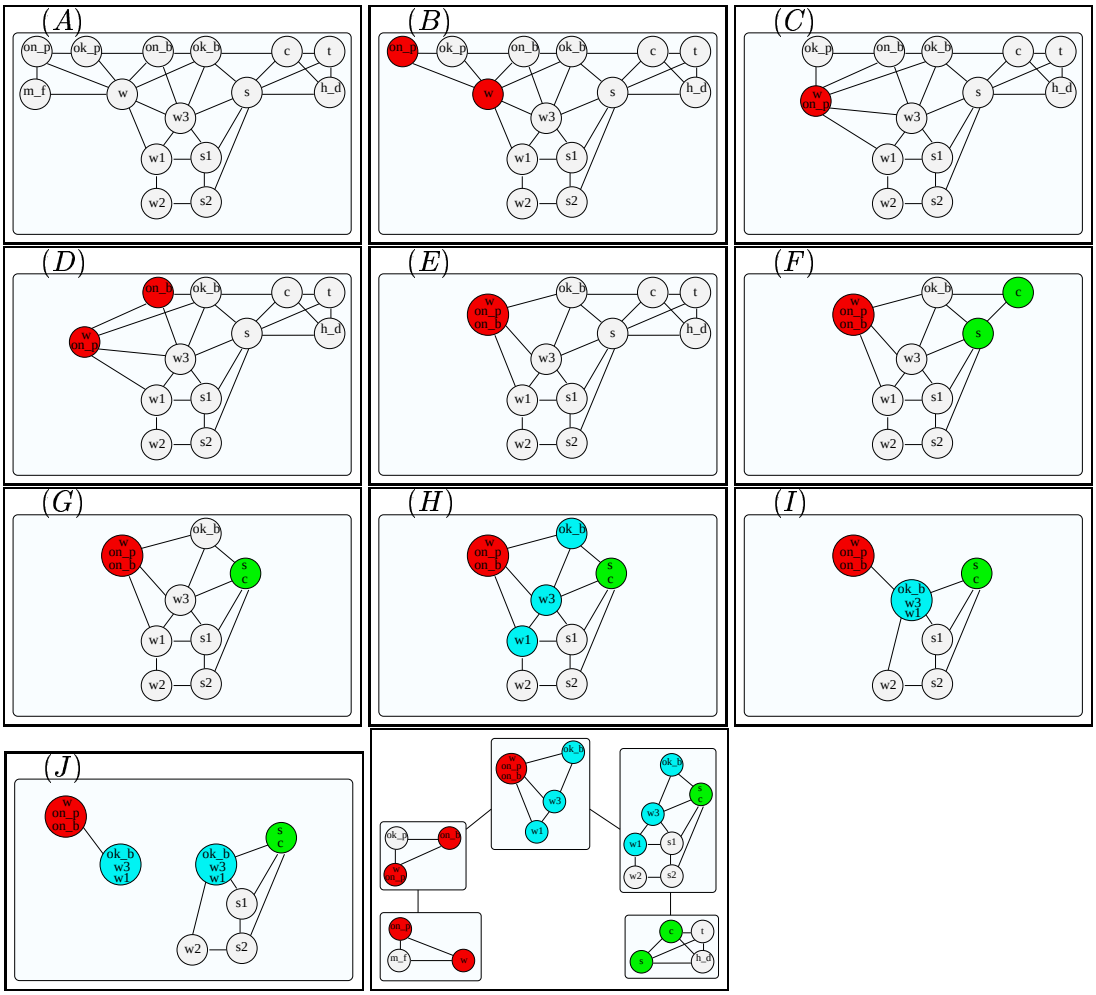

Figure 5.5: Recursive use of *Split* by aggregating minimal separators into single nodes. Only the larger side of the leftover graph is shown after each split.

r in step 6.

Proposition 5.4.1 *Procedure Split takes time* $O(|V|^{\frac{5}{2}} * |E|)$.

PROOF See Appendix A.4.1.

5.4.3 Fine-Tuning *Split*

Since $f_{SAT}(m)$ is not known, and the time for reasoning with FOL theories in MP is not bounded, it is not clear what is an *optimal* decomposition. Nevertheless, the analyses done throughout this

chapter and the preceding one suggests minimizing the parameters mentioned in the last section. If we assume that $f_{SAT}(m) = \Theta(2^{\alpha m})$, then the problem of finding an optimal partition for LINEAR-PART-SAT is equivalent to finding triangulations of minimum clique number (a.k.a. finding *treewidth*; we discuss this problem more thoroughly in the following sections). With this assumption, M should be chosen to be 1, l should be chosen to be m, and the algorithm will stop the recursive decomposition only when reaching a graph that is a clique. This is justified by the observation that any further decomposition can only decrease the size of the maximum partition (including the links). Thus, Assuming $f_{SAT}(m) = \Theta(2^{\alpha m})$, further decompositions only decrease the asymptotic time function of LINEAR-PART-SAT.

For reasoning with FOL theories it may sometimes be useful to choose M (the limit on the number of symbols in a partition) to be large, so that sentences are aggregated more closely to *topics*. This can be useful for managing large axiom sets as well as for applying specialized reasoning algorithms for each partition. This can be combined with replacing our vertex separator algorithm with a *balanced separator* algorithm. A balanced separator is a vertex separator that separates the graph such that the separated subgraphs are of comparable sizes (typically, they are chosen to be no larger than a constant times the size of the original graph). The problem of finding balanced separators is NP-hard, but several approximations exist (e.g., (Leighton and Rao, 1988; Klein et al., 1993; Feige and Krauthgamer, 2000)).

Our time bound for Split is lower than $\Theta(2^{\alpha m})$ when $l \leq \frac{\alpha m - \alpha m_i - lgn}{d}$ ($i = argmax_j m_j$). In particular, if $l > \frac{m}{2}$, a standard deterministic SAT procedure will be better (compared to the best time bound known for SAT procedures (Schiermeyer, 1996; Cook and Mitchell, 1997)).

All the observations above are predicated on the assumption the $\mathcal{A}$ is propositional and that $f_{SAT}(n) = O(2^{\alpha n})$, for some $\alpha > 0$ constant. If our theory is in FOL, or we drop the assumption on f_{SAT}, then there is no clear good way to choose M, l. In those circumstances, l and M are perhaps best determined experimentally.

5.5 Treewidth

In this section we briefly recall some of the main definitions pertaining to treewidth.

A cycle in a graph is *chordless* if no proper subset of the vertices of the cycle forms a cycle.

Definition 5.5.1 *A graph is* triangulated *(or* chordal*) if it contains no chordless cycle of length greater than three.*

A *triangulation* of a graph G is a graph H with the same set of vertices such that G is a subgraph of G and such that H is triangulated.

Definition 5.5.2 ((Robertson and Seymour, 1986)) *A* tree-decomposition *of a graph $G(V,E)$ is a pair $D = (S,T)$ with $S = \{X_i \mid i \in I\}$ a collection of subsets of vertices of G and $T = (I,F)$ a tree, with one node for each subset of S, such that the following three conditions are satisfied:* (1) $\bigcup_{i \in I} X_i = V$. (2) *For all edges $(v,w) \in E$ there is a subset $X_i \in S$ such that both v, w are contained in X_i.* (3) *For each vertex x, the set of nodes $\{i \mid x \in X_i\}$ forms a subtree of T.*

The *width* of a *tree-decomposition* $(\{X_i \mid i \in I\}, T = (I,F))$ is $max_{i \in I}(|X_i| - 1)$. The *treewidth* of a graph G equals the minimum width over all tree-decompositions of G. Equivalently, the treewidth of G is the minimum $k \geq 0$ such that G is a subgraph of a triangulated graph with all cliques of size at most $k+1$. Any triangulation of a graph defines a tree-decomposition of a graph of the same treewidth. Similarly, every tree-decomposition of a graph defines a triangulation of it of the same treewidth.

Definition 5.5.3 *Let $G(V,E)$ be a graph, $W \subseteq V$ a subset of the vertices and $\alpha \in (0,1)$ a real number. An α-vertex-separator of W in G is a set of vertices $X \subseteq V$ such that every connected component of $G[V \setminus X]$ has at most $\alpha|W|$ vertices of W. A* two-way *α-vertex-separator is required in addition to have exactly two sets, S_1, S_2, separated by X such that $S_1 \cup S_2 \cup X = V$ and $|S_i| \leq \alpha|W|, i = 1,2$.*

Lemma 5.5.4 ((Robertson and Seymour, 1986)) *If $G(V,E)$ is a graph with n vertices that has treewidth k, then there exists a set X with $k+1$ vertices such that every connected component of $G[V \setminus X]$ has at most $\frac{1}{2}(n-k)$ vertices.*

Corollary 5.5.5 ((Becker and Geiger, 1996)) *Let $G(V,E)$ be a graph with $n \geq k+1$ vertices and treewidth k. For every $W \subseteq V$, $|W| > 1$, there is a vertex separator X and sets $A, B, C \subset V$ such that $A \cup B \cup C \cup X = V$, A, B, C are separated by X, $|X| \leq k$ and $|W \cap C| \leq |W \cap B| \leq |W \cap A| \leq \frac{1}{2}|W|$.*

5.6 Using 2-Way Vertex Separators

The two algorithms presented in this section use two-way separators recursively. They differ on their choice of actual separator: $2/3$ versus $1/2$.

An example of these algorithms general procedure is shown in Figure 5.6.

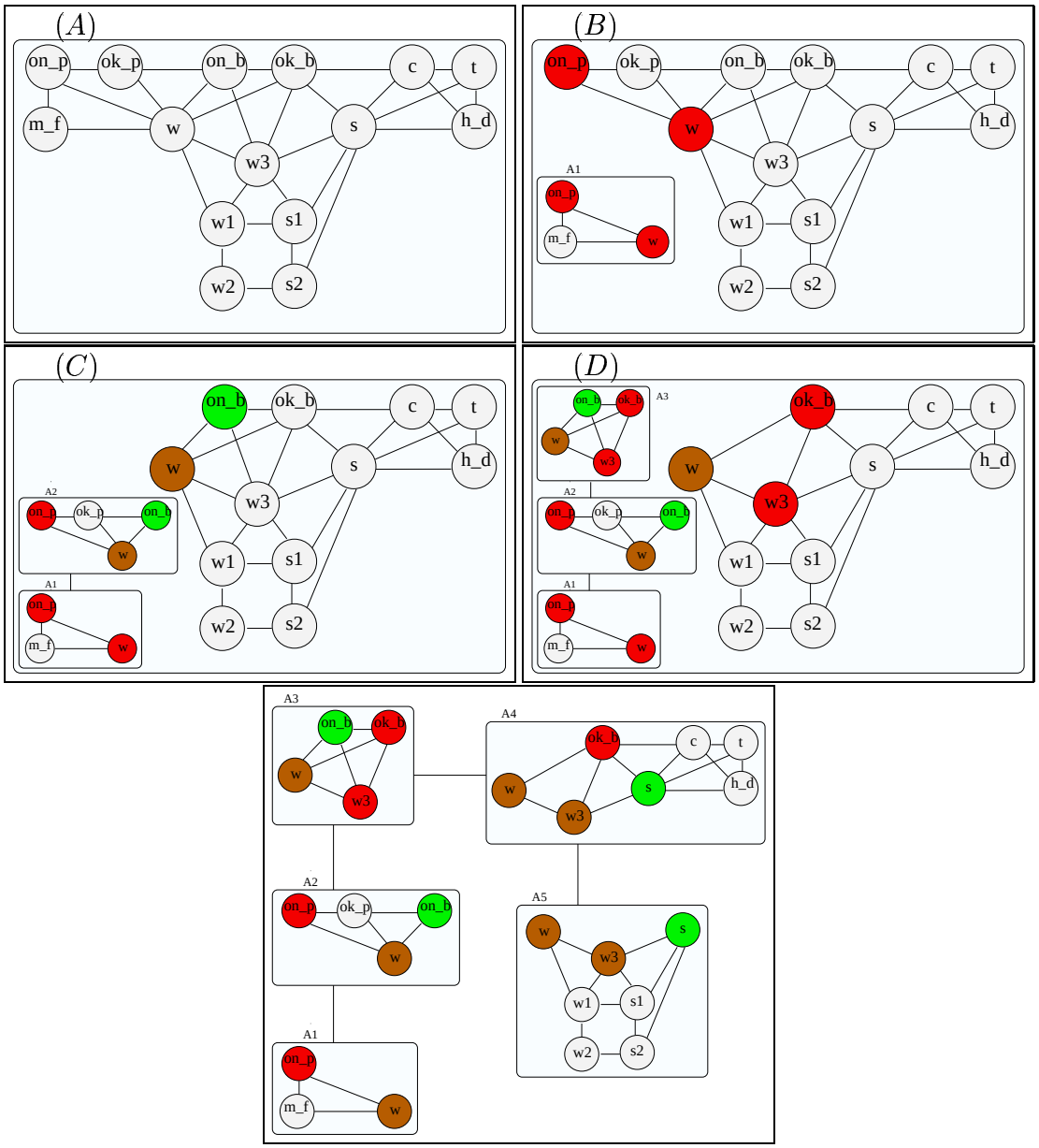

Figure 5.6: Recursively using a *balanced separator* of previous separators. The algorithms' progression is depicted in (A) through (D), culminating with the tree at the bottom.

5.6.1 Factor-4 Approximation Algorithm

Procedure *2way-2/3-triang*, displayed in Figure 5.7, finds factor-4 approximations. For a graph G and a parameter k, running 2way-2/3-triang(G, $\emptyset$, k), either returns a valid answer that the the

treewidth of G is of size $> k-1$ or it returns a triangulation of G of clique number at most $4k+1$.

PROCEDURE 2way-2/3-triang(G, W, k)
$G = (V, E)$ with $|V| = n$, $W \subseteq V$, k integer.

1. If $n \leq 4k$, then make a clique of G. Return.
2. Let $W' \leftarrow W$. Add to W' vertices from V such that $|W'| = 3k+2$.
3. Find X, a minimum $\frac{2}{3}$-vertex-separator of W' in G, with S_1, S_2 two nonempty parts separated by X ($S_1 \cup S_2 \cup X = V$) and $|X| \leq k$. If there is no such separator, then output "*the treewidth exceeds* $k-1$" and exit.
4. For $i \leftarrow 1$ to 2 do
   (a) $W_i \leftarrow S_i \cap W$.
   (b) call 2way-2/3-triang($G[S_i \cup X]$, $W_i \cup X$, k).
5. Add edges between vertices of $W \cup X$, making a clique of $G[W \cup X]$.

Figure 5.7: A factor-4 approximate triangulation algorithm.

This algorithm is very similar to that of (Robertson and Seymour, 1995), as presented in (Reed, 1992). The main difference is the more efficient algorithm that we use for exact vertex separation, which we provide below. The addition of elements to W' in step 2 ensures completeness of our separator (see Lemma 5.6.2's proof).

Lemma 5.6.1 *If $G(V,E)$ is a graph, k an integer and $W \subseteq V$ such that $|W| \leq 3k+2$, then* 2way-2/3-triang*(G,W,k) either outputs correctly that the treewidth of G is more than k or it triangulates G such that the vertices of W form a clique and the clique number of the result is at most $4k+1$.*

The proof is identical to that presented in (Robertson and Seymour, 1986; Reed, 1992).

Figure 5.8 presents the algorithm we will use for finding a $\frac{2}{3}$-vertex-separator of W' in G (step 3 in procedure *2way-2/3-triang*). It checks choices of sets of vertices to be separated until a solution is found or the choices are exhausted. The intuition behind making a clique from each selected set, W^i, is that doing so prevents any element from that clique from becoming an element in the separated subset of the other side. Given an arbitrary vertex separator of v_{W^1}, v_{W^2}, any vertex in the clique of W^1 must be either in the separator itself or in S_1.

Lemma 5.6.2 *Let $G(V,E)$ be a graph, $k \geq 0$ an integer, and $W \subseteq V$ of size $3k+2$. Algorithm $\frac{2}{3}$-vtx-sep(W, G, k) finds a $\frac{2}{3}$-separator of W in G of size $\leq k$, if it exists, returning* failure *otherwise. It does so in time $O(\frac{2^{4.38k}}{k} f(|V|, |E|+k^2, k))$, given a* min-$(a,b)$-vertex-separator *algorithm taking time $f(n,m,k)$.*

PROCEDURE $\frac{2}{3}$-vtx-sep(W, G, k)
$G = (V, E)$ with $|V| = n$, $W \subseteq V$, k integer.

1. Nondeterministically take a set W^1 of $\lceil \frac{|W|}{2} \rceil$ vertices from W and a set W^2 of $\lceil \frac{|W|}{3} \rceil$ vertices from $W \setminus W^1$.
2. Let $G' \leftarrow G$. Add edges to G' so that W^1 is a clique and W^2 is a clique. Create new vertices v_{W^1}, v_{W^2} in G' and connect them to all the vertices of W^1, W^2, respectively.
3. Find a minimum (v_{W^1}, v_{W^2})-vertex-separator, X. If $|X| \leq k$, return $|X|$ and two separated subsets S_1, S_2, discarding v_{W^1}, v_{W^2}. Otherwise, return "*failure*".

Figure 5.8: Find a $\frac{2}{3}$-vertex-separator of W in G.

PROOF See Appendix A.4.2.

Proposition 5.6.3 (cf (Reed, 1992)) *If the treewidth of $G(V, E)$ is $k - 1$, then $|E| \leq |V|k$.*

Theorem 5.6.4 *Procedure 2way-2/3-triang(G, $\emptyset$, k) finds a triangulation of G of clique number $\leq 4k + 1$, if the treewidth of G is at most $k - 1$, in time $O(2^{4.38k}|V|^{\frac{5}{2}})$ or $O(2^{4.38k}|V|^2 k)$ if we use the minimum (a, b)-vertex-separator algorithm of (Even, 1979) or (Ford Jr. and Fulkerson, 1962), respectively.*

PROOF See Appendix A.4.3.

5.6.2 Factor-$4\frac{1}{2}$ Approximation Algorithm

We can avoid many of the choices examined in procedure $\frac{2}{3}$-vtx-sep if we allow the resulting separator to be slightly larger. Procedure 2way-half-vtx-sep, presented in Figure 5.10 does that, returning a minimum two-way $\frac{1}{2}$-vertex separator. The combined procedure, called *2way-half-triang*, is identical to procedure 2way-2/3-triang besides replacing step 3. It is presented in Figure 5.9.

Lemma 5.6.5 *If $G(V, E)$ is a graph with treewidth $< k$ and $W \subseteq V$, then there is a two-way $\frac{1}{2}$-vertex-separator of W in G with size at most $k + \frac{1}{6}|W|$*

PROOF See Appendix A.4.4. ■

Lemma 5.6.6 *If $G(V, E)$ is a graph with n vertices, k an integer and $W \subseteq V$ such that $|W| \leq 3k + 2$, then* 2way-half-triang*(G,W,k) either outputs correctly that the treewidth of G is more than $k - 1$ or it triangulates G such that the vertices of W form a clique and the clique number of the resulting graph is at most $4\frac{1}{2}k + 2$.*

PROCEDURE 2way-half-triang(G, W, k)
$G = (V, E)$ with $|V| = n$, $W \subseteq V$, k integer.

1. If $n \leq 4k$, then make a clique of G. Return.
2. Let $W' \leftarrow W$. Add to W' vertices from V such that $|W'| = 3k + 2$.
3. Find X, a minimum two-way $\frac{1}{2}$-vertex-separator of W' in G, with S_1, S_2 the two nonempty parts separated by X ($S_1 \cup S_2 \cup X = V$) and $|X| \leq 1\frac{1}{2}k$. If there is no such separator, then output "*the treewidth exceeds* $k - 1$" and exit.
4. For $i \leftarrow 1$ to 2 do
   (a) $W_i \leftarrow S_i \cap W$.
   (b) call 2way-half-triang($G[S_i \cup X]$, $W_i \cup X$, k).
5. Add edges between vertices of $W \cup X$, making a clique of $G[W \cup X]$.

Figure 5.9: A factor-$4\frac{1}{2}$ approximate triangulation algorithm.

PROOF See Appendix A.4.5. ■

Procedure 2way-half-vtx-sep is very similar to procedure $\frac{2}{3}$-vtx-sep with one main difference. While $\frac{2}{3}$-vtx-sep selects two sets of sizes $\frac{1}{2}|W|$ and $\frac{1}{3}|W|$, procedure 2way-half-vtx-sep selects two sets of size $\frac{1}{2}|W|$. This precludes finding two-way separators in which one of the sets is of size $\frac{2}{3}|W|$ (as we did before).

PROCEDURE 2way-half-vtx-sep(W, G, k)
$G = (V, E)$ with $|V| = n$, $W \subseteq V$, k integer.

1. Nondeterministically choose a set W^1 of $\frac{|W|}{2}$ vertices from W. Let W^2 be $W \setminus W^1$.
2. Let $G' \leftarrow G$. Add edges to G' so that W^1 is a clique and W^2 is a clique. Create new vertices v_{W^1}, v_{W^2} in G' and connect them to all the vertices of W^1, W^2, respectively.
3. Find a minimum (v_{W^1}, v_{W^2})-vertex-separator, X. If $|X| \leq 1\frac{1}{2}k$, return $|X|$ and two separated subsets S_1, S_2, discarding v_{W^1}, v_{W^2}. Otherwise, return "*failure*".

Figure 5.10: Find a two-way $\frac{1}{2}$-vertex-separator of W in G.

Lemma 5.6.7 *Let $G(V, E)$ be a graph, $k \geq 0$ an integer, and $W \subseteq V$ of size $3k + 2$. Algorithm $\frac{1}{2}$-vtx-sep(W, G, k) finds a two-way $\frac{1}{2}$-separator of W in G of size $\leq 1\frac{1}{2}k$, if it exists, returning* failure *otherwise. It does so in time $O(\frac{2^{3k}}{\sqrt{k}} f(|V|, |E| + k^2, k))$, given a* min-$(a, b)$-vertex-separator *algorithm taking time $f(n, m, k)$.*

PROOF See Appendix A.4.6.

Theorem 5.6.8 *Procedure 2way-half-triang(G, $\emptyset$, k) finds a triangulation of G of clique number $\leq 3k+2$, if the treewidth of G is at most $k-1$, in time $O(2^{3k}n^{\frac{5}{2}}k^{\frac{1}{2}})$ or $O(2^{3k}n^2k^{\frac{3}{2}})$ if we use the minimum (a,b)-vertex-separator algorithm of (Even, 1979) or (Ford Jr. and Fulkerson, 1962), respectively.*

The proof of this theorem is similar to that of Theorem 5.6.4.

5.7 Using 3-Way Vertex Separators

The last section presented algorithms that recursively divide the set of vertices into two sets. Doing so we give up some of the separators guaranteed by Lemma 5.5.4. In this section we present a different angle on the tradeoff between the size of the separator, the size of each of the separated sides and the computational complexity of finding the separator. We find approximate *three-way separators*, and use them in a similar way to the one used above.

5.7.1 Multiway Vertex Cut

A generalization of the minimum (a,b)-vertex-cut problem is the minimum multiway-vertex-cut. Given an undirected graph, $G(V,E)$, and a set of nodes, $v_1,...,v_l \in V$, a minimum multiway cut is a minimum-cardinality set of nodes $S \in V$ such that $v_1,...,v_l$ are in different connected components in $V \setminus S$. The weighted version requires a minimum-weight set of nodes.

Unlike the minimum (a,b)-vertex-cut problem, the problem of finding a minimum multiway-vertex-cut is NP-hard and MAXSNP-hard for $l \geq 3$ (Cunningham, 1991; Garg et al., 1994) (i.e., there is $\epsilon > 0$ such that approximating the problem within a factor of $(1+\epsilon)$ is NP-hard). For the (a,b)-vertex-cut problem the maximum flow is equal to the minimum capacity cut in both directed and undirected graphs. This is not the case for multiway-vertex-cut. Nevertheless, (Garg et al., 1994) showed that by solving a maximum multicommodity flow problem, one can find an l-way vertex cut (in an undirected graph) that is of size within a factor $(2-\frac{2}{l})$ to the optimal (*multicommodity flow* is a generalization of maximum-flow for multiple sources, sinks and commodities sent between them (Leighton and Rao, 1999)).

This algorithm was used subsequently by (Becker and Geiger, 1996) to offer an algorithm for minimum-treewidth triangulation. This algorithm takes time $O(2^{4.66k}n\,poly(n))$, for $poly(n)$ the time required to solve a linear program of size n.

5.7.2 Factor-$3\frac{2}{3}$ Approximation Algorithm

Figure 5.11 recalls the main loop of the algorithm of (Becker and Geiger, 1996). The algorithm differs from that of (Robertson and Seymour, 1986) in using a 3-way separator instead of a 2-way separator. The separator, X of W, is required to satisfy $|(S_i \cap W) \cup X| \leq (1+\alpha)k$, for all three sets S_i, $i = 1, 2, 3$, for a given $\alpha \geq 1$. Let us call such a separator a α-sum-separator.

PROCEDURE 3way-triang(G, W, k)
$G = (V, E)$ with $|V| = n$, $W \subseteq V$, k integer.

1. If $n \leq (2\alpha + 1)k$, then make a clique of G. Return.
2. Let $W' \leftarrow W$. Add to W' vertices from V such that $|W'| = (1+\alpha)k + 1$.
3. Find X, a minimum α-sum-separator of W' in G, with S_1, S_2, S_3 three parts separated by X (at least two are nonempty) and $S_1 \cup S_2 \cup S_3 \cup X = V$. If there is none, then output "*the treewidth exceeds* $k-1$" and exit.
4. For $i \leftarrow 1$ to 3 do
   (a) $W_i \leftarrow S_i \cap W$.
   (b) call 3way-triang($G[S_i \cup X]$, $W_i \cup X$, k).
5. Add edges between vertices of $W \cup X$, making a clique of $G[W \cup X]$.

Figure 5.11: A factor-$3\frac{2}{3}$ approximate triangulation algorithm.

Figure 5.12 presents a new procedure for producing an α-sum-separator. It calls a procedure for 3-way vertex separation $3^{|W|}$ times instead of $4^{|W|}$ times as in the algorithm of (Becker and Geiger, 1996).

PROCEDURE α-sum-sep(W, G, k)
$G = (V, E)$ with $|V| = n$, $W \subseteq V$, k integer.

1. Nondeterministically divide $|W|$ into three sets, W^1, W^2, W^3, such that $\frac{|W|}{2} \geq |W^1| \geq |W^2| \geq |W^3|$.
2. If $|W^1| > k$, then set $W^2 \leftarrow W^2 \cup W^3$ and return the result of steps 2–3 of algorithm $\frac{2}{3}$-vtx-sep (Figure 5.8).
3. Let $G' \leftarrow G$. Add edges to G' so that each of W^1, W^2, W^3 is a clique. Create new vertices, $v_{W^1}, v_{W^2}, v_{W^3}$ in G' and connect them to all the vertices of W^1, W^2, W^3, respectively.
4. Find an α-approximation to a minimum $(v_{W^1}, v_{W^2}, v_{W^3})$-vertex-separator, X. If $|X| \leq \alpha k$, return X and the three separated sets, S_1, S_2, S_3, discarding $v_{W^1}, v_{W^2}, v_{W^3}$. Otherwise, return "*failure*".

Figure 5.12: Find an α-sum-separator in G of size at most k.

Lemma 5.7.1 *Let $G(V,E)$ be a graph, $k \geq 0$ an integer, and $W \subseteq V$ of size $(1+\alpha)k+1$. Algorithm α-sum-sep(W, G, k) finds a α-sum-separator of W in G, if it exists, returning* failure *otherwise. It does so in time $O(2^{3.6982k} f(|V|, |E|+k^2, k))$, for $\alpha = \frac{4}{3}$, with $f(n,m,k)$ the time taken by an algorithm for α approximation to* min-(a,b,c)-vertex-separator.

PROOF See Appendix A.4.7.

Theorem 5.7.2 ((Becker and Geiger, 1996)) *If $G(V,E)$ is a graph with n vertices, $k \geq 1$ an integer, $\alpha \geq 1$ a real number, and $W \subset V$ such that $|W| \leq (\alpha+1)k+1$, then 3way-vtx-sep(G,W,k) triangulates G such that the vertices of W form a clique and such that the size of a largest clique of the triangulated graph $\leq (2\alpha+1)k$ or the algorithm correctly outputs that the cliquewidth of G is larger than k.*

Solutions for linear programs of multicommodity flow problems are typically slow. The linear programming subroutine used by the procedure of (Becker and Geiger, 1996) for the subroutine of (Garg et al., 1994) can be replaced by the multicommodity flow algorithm of (Leighton et al., 1995). This combined algorithm finds a factor-$(1+\epsilon)\frac{4}{3}$ approximation to the optimal 3-way separator in time $O(\epsilon^{-2} nm\, lg^4 n)$, given $\epsilon > 0$. Selecting $\epsilon = \frac{1}{8k}$ guarantees that the separator is in fact a factor-$\frac{4}{3}$ approximation to the optimal (because the separator size is integral).

The complexity of running the algorithm with $\alpha = \frac{4}{3}$ is $O(2^{3.6982k} n\, f(n, m+k^2, k)) = O(2^{3.6982k} nk^3 n^2 lg^4 n) = O(2^{3.6982k} n^3 k^3 lg^4 n)$ using this procedure. This is an improvement in running time by a factor of at least $2^k n$ over the $O(2^{4.66k} n\, poly(n))$ of (Becker and Geiger, 1996).

5.7.3 Factor-$O(lgk)$ Approximation

In this section we use a procedure reported in (Leighton and Rao, 1999) for finding factor-βlgk approximations ($\beta = 720$) to minimum $\frac{2}{3}$-vertex-separator of W in G. This procedure calls a subroutine that solves multi-commodity flow at most $O(|W|)$ times. Using the algorithm of (Leighton et al., 1995) for multicommodity flow it takes time $O(|W| k^3 n^2 lg^4 n)$ for a graph of treewidth $k-1$. The triangulation algorithm uses the main loop of algorithm 3way-triang (Figure 5.11), replacing steps 2,3 by

2-3. Find X, an approximate minimum 3-way $\frac{2}{3}$-vertex-separator of W in G, with S_1, S_2, S_3 the three parts separated by X . If $|X| > \beta k$, then output "*the treewidth exceeds $k-1$*" and exit.

The combined algorithm for triangulation is used in the same way as before: we call lgk-triang(G, $\emptyset$, k).

Lemma 5.7.3 *If $G(V,E)$ is a graph with n vertices, k an integer and $W \subseteq V$ such that $|W| = \gamma_1 k \cdot lgk)$, for $\gamma_1 = 3\beta$, then* lgk-triang*(G,W,k) that uses the procedure of (Leighton and Rao, 1999) for finding 3-way $\frac{2}{3}$-vertex separators either outputs correctly that the treewidth of G is more than $k-1$ or it triangulates G such that the vertices of W form a clique and the clique number of the resulting graph is at most $\gamma lgk)$, for $\gamma = 4\beta$.*

PROOF See Appendix A.4.8.

Theorem 5.7.4 *Procedure lgk-triang(G, $\emptyset$, k) finds a triangulation of G of clique number $\leq \gamma lgk)$, for $\gamma = 4\beta$, if the treewidth of G is at most $k-1$, in time $O(n^3 lg^4 n\, k^5 lgk)$.*

The proof is similar to those for the previous algorithms.

PROOF Lemma 5.7.3 guarantees the correctness of the procedure. The time bound is seen noticing that there are at most n invocations of lgk-triang for a graph of n vertices. ■

5.8 Experimental Results

We have implemented a constructive variant of algorithms 2way-2/3-triang and 2way-half-triang. Given a graph, G, they return a tree decomposition of G. The main difference between the description given above and our implementation is that we do not increase the size of W' to be $3k+2$ in step 2) of Figure 5.7 (we do not know what k is, a priori). Instead, we gradually increase W''s size during the execution of $\frac{2}{3}$-vtx-sep, until we find a cardinality of $|W'|$ for which a minimum separator has both separated sets non-empty. This is particularly useful when only some of the partitions of the tree decomposition are of size close to the limit.

Graph	Nodes	Edges	$4\frac{1}{2}$-apx Time	4-apx Time	$4\frac{1}{2}$-apx Width+1	4-apx Width+1	*min-degree* Width+1	*min-discrepancy* Width+1
CYC1	142	469	62sec	394sec	21	21	14	14
CPCS1	360	1036	530sec	4260sec	28	26	21	21
CPCS2	421	1704	940sec	14Ksec	33	33	24	24
HPKB1	446	2637	7620sec	51Ksec	58	45	37	32
HPKB2	570	3840	28Ksec	516Ksec	70	60	41	37

Table 5.1: Graphs, their processing time and the resulting width of the decomposition.

For our implementation we use an implementation of Chekassky and Goldberg (Chekassky and Goldberg, 1997) for Dinitz's max-flow algorithm. We have experimented with several graphs of

various sizes and treewidths that are associated with real-world problems. The results are depicted in Figure 5.1. They were achieved on a Sun SuperSparc 60. For comparison[2] we ran the implementation of the algorithm of (Shoikhet and Geiger, 1997). That algorithm did not return answers for any of these graphs after more than three days. This is not surprising if we compare our theoretical results to those reported in (Becker and Geiger, 1996; Shoikhet and Geiger, 1997). These algorithms have been tested with graphs of treewidths ≤ 6, $n \leq 50$, $m \leq 110$ (real-world graphs) and treewidth ≤ 10, $n \leq 100$ (artificially generated), respectively, an order of magnitude lower than those used here.

It is important and interesting to notice that the *min-degree* and *min-discrepancy* heuristics[3] consistently achieve better tree decompositions than our approximation-guaranteed algorithms. These heuristics take between 1 second and 7 minutes on our sample graphs with a sub-optimal implementation, but are not guaranteed to approximate the optimal by a constant factor (there are examples in which those heuristics return a treewidth of $\sqrt{n}$ on graphs of low treewidth[4]).

5.9 Related Work: Automated Decomposition

5.9.1 Automated Decomposition for Reasoning

Decomposition techniques for CSPs, Bayes Nets and other NP-hard problems are most relevant to our work on automated decomposition. These typically look for a *separation vertex* (Dechter, 1990), use various heuristics for order on symbols (that translates to a decomposition of the graph) (Dechter and Pearl, 1989; Selman and Kautz, 1993; Dechter and Rish, 1994) and use approximations for *tree decomposition of minimum width* (equivalent to finding *triangulations* of minimum clique number, computing *treewidth*, and finding optimal *clique trees*) (Robertson and Seymour, 1986; Robertson and Seymour, 1995; Kloks, 1994; Becker and Geiger, 1996; Shoikhet and Geiger, 1997).

The last approach is applicable to our setup, if we assume that $f_{SAT}(m) = \Theta(2^{\alpha m})$. In contrast to our SPLIT, these algorithms find weak approximations (factor $O(log\ n)$) to the optimal in polynomial time and constant-factor approximations or optimal results in quasi-polynomial time (polynomial time, assuming the treewidth is bounded by a constant, and exponential time otherwise). Furthermore, work on implementing SAT and automated deduction strongly suggests that

[2] We could not get the implementation of (Becker and Geiger, 1996).

[3] (Rose, 1974) calls the latter *minimum deficiency*, while (Kjaerulff, 1993) calls the two *minimum size* and *minimum fill*, respectively.

[4] Dan Geiger, private communication.

the assumption of $f_{SAT}(m) = \Theta(2^{\alpha m})$ is overly pessimistic. For this reason we prefer to minimize the links first, and then look at minimizing the partitions, leading to our proposed algorithm.

Cut-set conditioning (Pearl, 1988; Dechter, 1990; Becker and Geiger, 1994; Becker et al., 2000) and hypertree decompositions of CSPs (Gottlob et al., 1999) are other methods for using decompositions, that are fairly different from the one we use in this chapter.

5.9.2 Optimal Decomposition

An optimal solution for the TREEWIDTH problem is known to be NP-hard (Arnborg et al., 1987). It is an open question whether a constant-factor approximation can be found in polynomial time. Nevertheless, several algorithms were found with either a constant approximation factor and an exponential time-dependency on k, the treewidth of the graph, or a $O(log(n))$ approximation factor and a polynomial execution time. We briefly recount some of the previous work done on finding tree decompositions of minimum width.

First, the best approximation using polynomial-time for this problem is due to (Kloks, 1994; Bodlaender et al., 1995). Their algorithm yields a factor-$12 \cdot k \cdot \Delta \cdot log(n+1)$ (Δ is a large unspecified constant) approximation, taking time $O(poly(n \cdot k))$, where $poly(n \cdot k)$ is the time to compute an approximate balanced vertex separator in the sense of (Klein et al., 1990) (Lemma 4.1 in (Klein et al., 1990) promises an algorithm, using the algorithm of (Leighton and Rao, 1988) which in turn uses linear programming; the randomized algorithm of (Leighton et al., 1995) does the task in *expected* time $O(nm\Delta log^3 n)$, Δ is the maximum degree of any node in the graph).

For a constant k, several authors provided polynomial algorithms for the exact solution. (Robertson and Seymour, 1986) provided the first algorithm using $O(n^{2k^2+4k+8})$ time (the complexity bound is due to (Arnborg et al., 1987)). (Arnborg et al., 1987) improved this algorithm to takes time $O(n^{k+2})$. In 1986 (Robertson and Seymour, 1995) gave a non-constructive proof of the existence of a decision procedure that uses $O(n^2)$ time. The procedure would first compute a tree decomposition that is a constant-ratio approximation to the optimum. It would then use this bounded-treewidth decomposition to compute the optimal decomposition. Later, (Reed, 1992) showed that the first part of their algorithm takes time $O(k^2 \cdot 3^{3k} \cdot n^2)$ and finds a decomposition that is of treewidth $\leq 4k+3$.

(Lagergren, 1990) developed an algorithm that performs the first part in $O(f(k)nlog^2 n)$ time and produces a treewidth of size $\leq 8k+7$, where $f(k)$ is

$$k \cdot 2^{k+3} \cdot (k+2))^k \cdot (k+1)!^2 \cdot k \cdot 2^k$$

which is more than $2^{k^2} \cdot (k+1)!^2$. (Reed, 1992) proposed an algorithm that takes $O(k^2 3^{4k} nlogn)$ time to solve the first part and provides an approximation of factor 5. Both algorithms yield a valid answer that the width is larger than k, or output a tree decomposition that has treewidth bounded by a linear function in k.

(Bodlaender and Kloks, 1991; Kloks, 1994) provide algorithms for the second part that work in linear time (again, with k fixed). If we have a tree decomposition of width l, and look for a tree decomposition of width k, then the time is

$$O(l^{l-1} \cdot ((2l+3)^{2l+3} \cdot (\frac{8}{3} \cdot 2^{2k+2})^{2l+3})^{2l-1} \cdot n)$$

(Bodlaender, 1996; Bodlaender, 2000) (this is more than $O(2^{8k^3} n)$ time). (Lagergren and Arnborg, 1991; Abrahamson and Fellows, 1993) provided algorithms for the second part with similar super-exponential dependency on k. (Bodlaender, 1996) provided a combined linear-time algorithm (for both parts) having a similar dependency on k, l, using the algorithm of (Bodlaender and Kloks, 1991).

(Becker and Geiger, 1996) presented an approximation algorithm that finds a factor 3.66 approximation to the optimal decomposition in time $O(2^{4.66k} n \cdot poly(n))$, where $poly(n)$ is the running time of linear programming. (Shoikhet and Geiger, 1997) presented an algorithm that finds an optimal triangulation in time $O(Rn^5 + R_k n^3 C_{max})$ where R is the number of all minimal separators of G, R_k is the number of minimal separators of G of size at most k, and C_{max} is the maximal number of maximal cliques in a *validity graph*[5] of some fragment in $\mathcal{F}_G$ ($\mathcal{F}_G$ is the set of fragments of the graph obtained by a separator of G of size $\leq k$). (Broersma et al., 1998) presented an algorithm that finds the optimal triangulation in time $O(Rn^5 + KR^{K+1}(n+m)n \cdot logn)$, where R is the number of all minimal separators of G and K is the *asteroidal number* of G (the algorithm does not require prior knowledge of the asteroidal number of G).

5.9.3 Other Decomposition Approaches for Logic

There are many possible approaches to decomposing a set of logical axioms. One complementary approach that we have briefly experimented with is a normalized cut algorithm (Shi and Malik, 1997), using the dual graph of the theory. The dual graph represents each axiom, rather than each symbol, as a node in the graph to be split. The possible advantage of this approach is that it preserves the distinction of an axiom. Also, since the min-cut algorithm is normalized, it helps preclude

[5]The notion of a *validity graph* is defined in their paper.

the creation of small isolated partitions by both maximizing the similarity within partitions and minimizing the similarity between partitions.

A different decomposition is conceived from a semantic approach. Our reasoning algorithms and our computational analysis suggested a syntactic approach to decomposition. Semantic approaches are also possible along lines similar to (Chang and Lee, 1973) (Chapter on semantic resolution) or (Slaney and Surendonk, 1996). Such decomposition approaches may require different reasoning algorithms to be computationally useful.

5.10 Summary and Discussion

This chapter presented a greedy algorithm and four related approximation algorithms for triangulation of minimum treewidth. Two of the approximation algorithms (2way-2/3-triang and 3way-triang) are modifications of previous algorithms that improve their running speed by a factor exponential in k and polynomial in n. A third algorithm (2way-half-triang) has the best combined n, k time bound known for any constant-factor approximation algorithm. The fourth algorithm (lgk-triang) is the first polynomial-time algorithm for an approximation factor that does not depend on n. Our greedy algorithm has no approximation guarantee, but is the fastest among these algorithms. We showed that our algorithms are efficient enough to solve large problems of practical importance. The results of some of the tree decompositions that we produced are currently being used in reasoning with the HPKB and CYC knowledge bases (Cohen et al., 1999; Lenat, 1995) using the algorithms of chapter 4.

Chapter 6

A Logic-Based Subsumption Architecture

6.1 Introduction

In (Brooks, 1986), Rodney Brooks provided a decomposition of the problem of robot control into layers corresponding to levels of behavior, rather than a sequential, functional form. Within this setting, he introduced the idea of subsumption, that is, that more complex layers not only depend on lower, more reactive layers, but could also influence their behavior. The resulting architecture was one that could simultaneously service multiple, potentially conflicting goals in a reactive fashion, giving precedence to high-priority goals.

Because of its realization in hardware, the subsumption architecture lacks declarativeness, making it difficult to implement higher-level reasoning and making its semantics unclear. The increasing hardware complexity with new layers introduces scaling problems. And, relying on hardware specifications, the architecture is specifically oriented towards robot control and is not applicable to software-based intelligent agents. The problem of extending similar architectures to more complex tasks and goals and to agents that are not necessarily physical has already been raised and discussed in general terms by (Minsky, 1985) and (Stein, 1997), but to our knowledge, no practical AI architecture has been developed along these lines.

In this chapter we describe an architecture that is modeled in the spirit of Brooks' subsumption architecture but relies on a logical framework and has wider applicability and extendibility in the manner described above. Our *Logic-Based Subsumption Architecture* (LSA) includes a set of

First-Order Logic (FOL) theories, each corresponding to a layer in the sense of Brooks' architecture. Each layer is supplied with a separate theorem prover, allowing the system of layers to operate concurrently. After proving a goal, each layer sends the result to lower layers. We use nonmonotonic reasoning to allow layers to make default assumptions. These assumptions may be subsumed when information arrives from higher layers or from the agent's sensors. This allows each layer's performance to be independent of the performance of other layers, supporting reactivity. We also use nonmonotonic reasoning to provide semantics for the combined (static) system.

The architecture has been implemented and tested on a mobile robot. It exhibits real-time performance and performs navigation and control tasks. The layers can receive and incorporate new axioms from the user at run-time, allowing the user to give advice to the robot and to correct behaviors that are erroneous. The architecture also allows incorporating layers that perform diagnosis and can be extended to layers that remember experiences for other layers. Our experiments show that real-time commonsense control of AI agents can be achieved with an architecture based on logical theories and general-purpose theorem provers with the use of subsumption. These theorem provers can be replaced with special-purpose reasoners, for improved efficiency, but we show that in simple cases this is not needed with current theorem-proving technology. We report on the set of experiments that we executed with this system on a Nomad200 mobile robot.

This work improves over results presented using GOLOG (e.g. (Levesque et al., 1997)) and the work of (Shanahan, 1996) in that our system is fully declarative and has the full expressiveness of FOL. We provide a more detailed comparison to these and other related work at the end of the chapter. Some of the results in this chapter appeared previously in (Amir and Maynard-Reid II, 1999; Amir and Maynard-Reid II, 2001a).

6.2 Principles of Subsumption Architectures

6.2.1 Brooks' Subsumption Architecture

Brooks (Brooks, 1986) showed that it is often advantageous to decompose a system into parallel tasks or behaviors of increasing levels of competence rather than the standard functional decomposition. Whereas a typical functional decomposition might resemble the sequence

$$\text{sensors} \rightarrow \text{perception} \rightarrow \text{modeling} \rightarrow \text{planning} \rightarrow \text{task recognition} \rightarrow \text{motor control},$$

Brooks would decompose the same domain as

avoid objects < wander < explore < build maps < monitor changes < identify objects < plan actions < reason about object behavior

where < denotes increasing levels of competence. Potential benefits from this approach include increased robustness, concurrency support, incremental construction and ease of testing.

An underlying assumption is that complex behavior can be the product of many simple behaviors interacting with each other and a complex environment. This focus on simplicity leads to a design where each individual layer is composed of simple state machine modules operating asynchronously without any central control.

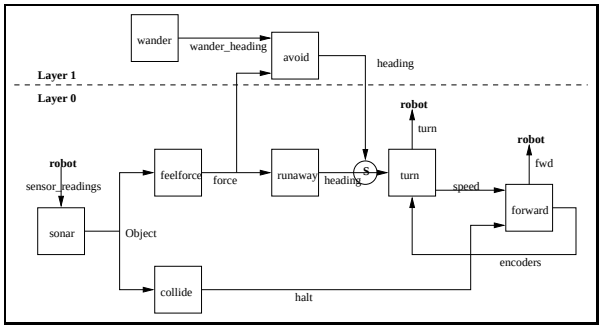

Figure 6.1: Layers 0 and 1 of Brooks' subsumption architecture robot control system.

In general, the different layers are not completely independent. In the decomposition above, wandering and exploring depend on the robot's ability to avoid objects. But the system may be able to service multiple goals in parallel, despite the dependence. The goals of one layer will occasionally conflict with those of another layer, in which case higher-priority goals should override lower-priority ones. Consequently, the subsumption architecture provides mechanisms by which higher, more competent layers may observe the state of lower layers, inhibit their outputs and override their inputs, thus adjusting their behavior. High-priority tasks in lower layers (such as reflexively halting when an object is dead ahead) will still have a default precedence if the designer disallows any tampering with these particular tasks.

Following Brooks' work and others (e.g., see (Arkin, 1998)) there have been much work on implementing behavior-based robots, when the different behaviors are stacked in a subsumption architecture or are simply switched by an automaton. In the former, layers typically avoid intervening with the internals of layers below, opting to inhibit the output of lower layers, replacing it with the higher layer's output.

Brooks implemented a control system of layers corresponding to the first three levels of competence described above (avoidance, wandering and exploration). The first two layers are shown in Figure 6.1. The `avoid` layer endows the robot with obstacle avoidance capabilities by moving it in directions that avoid obstacles as much as possible and forcing it to stop if a head-on collision is imminent. The `wander` layer causes the robot to move around aimlessly when it is not otherwise occupied.

The `avoid` layer accepts sonar readings of the robot's surroundings into its `sonar` module which outputs a map of the vicinity based on these readings. The `collide` module checks if there is an obstacle directly ahead and, if there is, forces the robot to stop regardless of what other modules are doing. The `feelforce` module uses the map to calculate a combined repulsive "force" that the surrounding objects exert on the robot. The `runaway` module checks if this force is significant enough to pay attention to and, in the case that it is, determines the new heading and speed for the robot to move away from the force. The `turn` module commands the robot to make the required turn, then passes the speed on to the forward module which, if not in a halt state, commands the robot to move forward with the specified speed. The further away the robot gets, the smaller the speed computed by the `runaway` module.

Every so often, the `wander` module chooses a new random direction for the robot to move in. The `avoid` module combines it with the output of the `avoid` layer's feelforce module, computing an overall heading that suppresses the input to the `avoid` layer's `turn` module.

The `explore` layer gives the robot some primitive goal-directed behavior by periodically choosing a location in the distance and heading the robot towards it if idle. While in explore mode, this layer inhibits the `wander` layer so that the robot remains on track towards its destination. When either the `wander` or the `explore` layer is active, it overrides the default heading computed by the `avoid` layer, but the `avoid` layer still ensures that the robot does not have a collision. We refer the reader to (Brooks, 1986) for further details.

6.2.2 Behavioral Decomposition

The first important idea we borrow from Brooks' architecture is that of decomposing the domain along behavioral lines rather than along the standard, sequential functional lines. A *Logic-Based Subsumption Architecture* (LSA) is composed of a sequence of FOL theories. Each represents a layer with an axiomatization of the layer's behavior, that is, the layer's inputs, outputs (goal), state and any dependencies between them. The inputs are axioms coming from either the sensors or higher layers. The outputs are proved theorems determined by running a separate theorem prover

for that layer only. These outputs may be sent to lower layers or to the robot effectors.

Because the axiomatization of a layer is usually much smaller than that of the whole system, each cycle is less computationally expensive than running one theorem prover over the whole compound axiomatization, leading to an overall higher performance. Another advantage of the layer-decoupling is the possibility of achieving more reactive behavior. As in Brooks' system, lower layers controlling basic behaviors are trusted to be autonomous and do not need to wait on results from higher layers (they assume some of them by default) before being able to respond to situations. Because these layers typically have simpler axiomatizations, and given the default assumptions, the cycle time to compute their outputs can be shorter than that of the more complex layers.

6.2.3 Subsumption Principles

Of course, the layers are *not* fully independent. We adopt the view that, together with the task-based decomposition idea, the coupling approach represented by subsumption in the subsumption architecture is an important and natural paradigm for intelligent agents in general, and robot control in particular (see (Stein, 1997)). We want each layer in an LSA to be able to communicate with those underneath it in the hierarchy.

In general, however, when one layer overrides another, the two disagree on what some particular input should be. In a classical logic setting, the two corresponding theories will be inconsistent. We need to formalize the higher-layer theory's precedence over the lower layer's in such a way that (a) if there is no conflict, both layers keep their facts and the higher layer asserts its relevant conclusions in the lower layer, and (b) if there is conflict, the lower layer tries to give up some assumptions to accommodate the higher layer's conclusions. A number of techniques developed in the logic community are applicable, e.g., nonmonotonic techniques and belief revision. We have chosen to use circumscription, although other approaches may be equally interesting and appropriate.

6.3 Logic-Based Subsumption

This section describes how we implement the principles discussed above. The first important idea that this architecture borrows from Brooks' architecture is that of decomposing the domain along behavioral lines into simple layers. Unlike systems that followed Brooks' work, it allows the layers to work in synergy to produce the compound behavior.

6.3.1 Basic Machinery

A *Logic-Based Subsumption Architecture* (LSA) is built of layers corresponding to behaviors (see Figure 6.2). The layers work concurrently and assynchronously to each other.

We distinguish four parts of a logical layer: (1) the *body* of the layer, (2) the *sensory and input Latches*, (3) the *output*, and (4) the *default assumptions*. The *body* of the layer is a fixed axiomatization describing the behavior of that layer. The *latches* are used to accept input axioms from the sensors and from higher layers and replace them at the beginning of every cycle (rather than accumulate this input). The *output* is a fixed set of *goal* sentences (possibly with some free variables) whose proof and instantiation determine the behavior sanctioned by the layer's theory (including the latches axioms). The *default assumptions* are used to implement the idea of subsumption between layers. These assumptions are implemented using nonmonotonic reasoning methods, which we describe in more detail in Section 6.3.2.

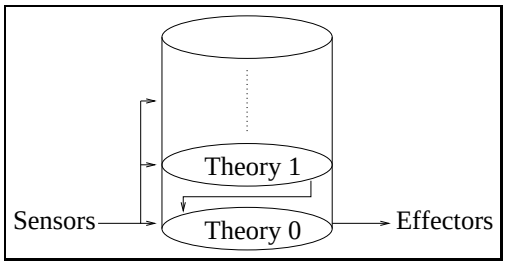

Figure 6.2: An abstract diagram of the LSA.

A logic-based subsumption system is comprised of a set of layers, each equipped with a theorem prover and concurrently running its own processing loop. The processing loop of each layer proceeds as follows: First, collect any pertinent sensor data and assert it in the form of logical axioms. Simultaneously, assert any inputs from higher-level theories. The theorem prover of that layer then attempts to prove the layer's goal, from the theory including the default assumptions. Upon proving its goal, the layer transmits the goal instantiation to the layer below or (in the case of the lowest layer) to the robot manipulators.

Because the axiomatization of a layer is usually much smaller than that of the whole system, each cycle is less computationally expensive than running one theorem prover over the whole compound axiomatization. As in Brooks' system, lower layers controlling basic behaviors are trusted to be autonomous and do not need to wait on results from higher layers (they assume some of them by default) before being able to respond to situations.

We will typically include a form of nonmonotonicity that is not computationally expensive or that is a fast approximation for a more computationally heavy form of nonmonotonicity. Using a fast form of nonmonotonicity for implementing default assumptions and having lower layers typically having simpler axiomatizations, the cycle time to compute these layers' outputs can be significantly shorter than that of more complex layers.

6.3.2 Circumscription-Based Subsumption

We use nonmonotonic reasoning to introduce defaults for each layer. Without nonmonotonicity in each layer, goals that were proved once without input from higher layers cannot be rejected upon the introduction of new axioms arriving from higher layers.

An example of a suitable nonmonotonic-reasoning system, is McCarthy's circumscription (McCarthy, 1986b) formula:

$$Circ[A(P,Z);P;Z] = \\ A(P,Z) \wedge \forall p,z\,(A(p,z) \Rightarrow \neg(p < P))$$

It says that in the theory A, with parameter relations and function sequences P, Z, P is a minimal element such that $A(P,Z)$ still holds while Z is allowed to vary in order to allow P to become smaller. Roughly speaking, adding this formula allows us to say that the predicate P is true for only those elements for which it must be true. In other words, P is false by default. To state more complicated defaults one can add axioms and predicates. For example, if we want to say that P *is true by default*, then we can add a new predicate symbol, P', and the axiom $\forall x\ P(x) \iff \neg P'(x)$, and minimize P' in the circumscription formula.

Take, for example, the theory

$$A \equiv block(B_1) \wedge block(B_2)$$

The circumscription of $block$ in A, varying nothing, is $Circ[A;block;] = A \wedge \forall p\ [A_{[block/p]} \Rightarrow \neg(p < block)]$ and is equivalent to $\forall x\ (block(x) \Leftrightarrow (x = B_1 \vee x = B_2))$. By minimizing $block$, we have concluded that there are no other blocks in the world besides those mentioned in the original theory A.

In the LSA, we use circumscription for two distinct tasks: assuming defaults in the layers and giving semantics to the system of layers as one big logical system. The first is the one used to implement subsumption in the actual system, influencing both the semantics and the implementation.

The second is needed for giving semantics to the directional nature of the complete system (i.e., that messages between layers go only in one direction).

To implement the idea of subsumption, we let each layer make default "assumptions" about the inputs that later may be adjusted by other (higher-level) layers. These assumptions typically take the form of the Closed-World Assumption (CWA) by minimizing a predicate in the layer's input language (Extended CWA, a generalization of CWA, was shown to be equivalent to circumscription (Gelfond et al., 1989)).

More formally, for a set of axioms, A, let $L(A)$ be the set of nonlogical symbols (predicates, functions, and constants) that appear in A. Also, let $\mathcal{L}(A)$ be the FOL language built using the symbols in $L(A)$ (a language here is the set of all FOL sentences that can be built from those symbols). Let $Layer_i$ be the combined theory of layer i, i.e., the combination of the body axioms, $Base_i$, the sensory-latch axioms, $Sensors_i$, and the input-latch axioms, $Input_i$. Let $\overrightarrow{C}_i$ be a set of predicates in $\mathcal{L}(Layer_i)$ for which we wish to assert CWA. Then, subsumption is achieved for layer i by using the parallel circumscription policy

$$Circ[Layer_i; \overrightarrow{C}_i; L(Layer_i)] \tag{6.1}$$

When implemented, this formula often can be substituted with a simple (external to the logic) mechanical interference determining the value of the minimized predicates; we discuss this issue in section 6.6. Other systems for nonmonotonic reasoning can also be used instead of circumscription, depending on the intended behavior and the designer's choice of tradeoffs (e.g., time versus expressivity).

6.3.3 Putting It All Together

In the case of using Circumscription for nonmonotonic assertions, each layer tries to prove

$$Circ[Layer_i; \overrightarrow{C}_i; \overrightarrow{Z}_i] \models \exists \overrightarrow{x}\, Goal_i(\overrightarrow{x})$$

Here, $\overrightarrow{C}_i$, $\overrightarrow{Z}_i$ are specified as part of the defaults for layer i, $Layer_i$ is the set of axioms including the body and the latches and $Goal_i(\overrightarrow{x})$ is a goal formula specified for layer i ($\overrightarrow{x}$ is a vector of variables open in $Goal_i(\overrightarrow{x})$). Upon proving $Goal_i(\overrightarrow{a})$, the layer transmits $Goal_i(\overrightarrow{a})$ either to the layer below or (in the case of the lowest layer) to the robot manipulators. Figure 6.3 summarizes this algorithm, while Figure 6.4 illustrates the process.

PROCEDURE LSA($\{Layer_i\}_{i\leq n}$, $\{Goal_i\}_{i\leq n}$)
$\{Layer_i\}_{i\leq n}$ a layered theory T, $Goal_i$ a fixed goal in $\mathcal{L}(Layer_i)$ ($i \leq n$).

Concurrently, for each layer, i:

1. Request sensory data from the robot and assert it into the Sensory Latch, $Sensors_i$.
2. Combine axioms in the Body theory with those in the sensory and input latch: $Layer_i \leftarrow Base_i \cup Sensors_i \cup Input_i$.
3. Try to prove $Circ[Layer_i; \overrightarrow{C}_i; \overrightarrow{Z}_i] \models \exists \overrightarrow{x}\ Goal_i(\overrightarrow{x})$ (try to prove the goal from $Layer_i$ given the default assumptions).
4. If $Goal_i(\overrightarrow{a})$ was proved for assignment $\overrightarrow{a}$, assert $Goal_i(\overrightarrow{a})$ in the Input Latches of layer $i-1$, $Input_{i-1}$.

Figure 6.3: The LSA algorithm.

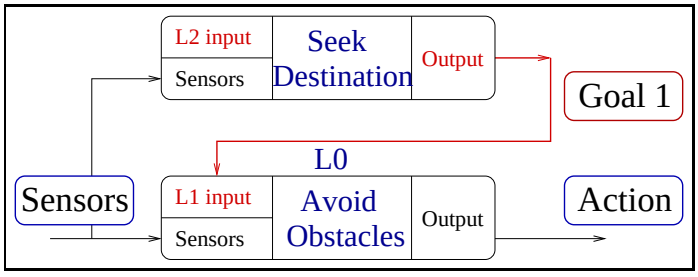

Figure 6.4: A detailed look at two layers.

This description of LSA hides two issues: First, what happens when a layer cannot prove something? Second, what happens to the input latch of a receiving layer after some time has passed? For the first question, in general we assume that the theorem prover for each layer works without interruptions until it finds a proof. If the theorem prover did not find a proof after some timeout, we restart the prover (possibly on a different sub-space of the search space) with the new latch information. Alternatively, one can assume that the sensory latch and the input latch are refreshed assynchronously, and the prover immediately takes any new information into account, discarding any old information from that latch (and any of the consequences it may have made on the basis of the old latches). For the second question, we assume in this thesis that latch information disappears after some time. Thus, if layer 1 did not prove its goes for the last few seconds, then layer 0 will no longer consider an axiom sent previously by layer 1 as valid.

6.4 Static Semantics for LSA

An LSA system has layers that are assynchronous and acting in a changing world. There is also discrepancy between sensing and acting, as the action is never executed in exactly the same world in which the sensing was done. Also, information collected by the sensors is always imprecise. We do not try to model these here. In this section we assume that sensing and acting is done in a stationary world.

First, we prove a technical result, showing the existence of an interpolation theorem for Circumscription.

Theorem 6.4.1 *Let T_1, T_2 be two theories, P, Q vectors of symbols in $L(T_1) \cup L(T_2)$ such that $P \subseteq L(T_1)$ and $P \cup Q \supset L(T_2)$. Let γ be the set of prime impliactes of T_2 in $\mathcal{L}(L(T_1) \cap L(T_2))$. Then, for every $\varphi \in \mathcal{L}(T_1)$,*

$$Circ[T_1 \cup \{\gamma\}; P; Q] \models \varphi \iff Circ[T_1 \cup T_2; P; Q] \models \varphi$$

PROOF See Appendix A.5.1. ■

Corollary 6.4.2 (Interpolation Theorem for Circumscription) *Let T be a theory, P, Q vectors of symbols in $L(T)$ such that $(P \cup Q) \supseteq L(T)$, $P \subseteq L(\varphi)$. Then, if $Circ[T; P; Q] \models \varphi$, then there is $\gamma \in \mathcal{L}(T) \cap \mathcal{L}(\varphi)$ such that*

$$T \models \gamma \quad \textit{and} \quad Circ[\gamma; P; Q] \models \varphi.$$

Furthermore, this γ can be the set of prime implicates of T in $L(T) \cap L(\varphi)$.

PROOF See Appendix A.5.1. ■

If we ignore the time differences between the theorem provers in different layers and consider the entire system of layers as one logical theory, we can give the system a simple semantics. In what follows we are interested in the output of the system to the actuators of the robot.

Let $Layer_i$ be the theory of layer i, and assume that we use first-order circumscription to assume defaults for layer i. We include any default as a FOL schemata. Let $Goal_i(\overrightarrow{x})$ be the goal formula of layer i (i.e., the formula that we try to prove in that layer). We call such a system of layers, T, a *layered theory*. Let T have $n+1$ layers (i.e., layers are numbered $0, ..., n$). For $i \leq n$,

$\varphi \in \mathcal{L}(Layer_i)$, we write $T \vdash \varphi$ if the mechanical entailment we described above derives φ in step 3 (for any layer $j \leq n$).

Definition 6.4.3 (Output Semantics for Layered Theories) *$\mathcal{M}$ is a model of the layered theory T (written $\mathcal{M} \models T$) iff it is a first-order model of*

$$Circ[Layer_0 \cup Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]; \overrightarrow{C}_0; \overrightarrow{Z}_0]$$

where $\overrightarrow{Z}_i = L(\bigcup_{j=i}^{n} Layer_j)$, and $\overrightarrow{C}_i$ is taken as in formula (6.1).

We assume that $L(Layer_i) \cap L(Layer_j)$ includes no predicate symbols if $i \neq j$ and $i \neq j+1$, and that the predicate symbols that appear in $L(Layer_i) \cap L(Layer_{i+1})$ appear also in $Goal_{i+1}(\overrightarrow{x})$.

Let T be a layered theory with $n+1$ layers, $0, ..., n$, and assume that for every $i \leq n$, $Goal_i(\overrightarrow{x})$ is a single literal. The LSA obeys this semantics, assuming that transferring a single instantiation of the goal between any pair of layers is sufficient (i.e., that the only prime implicate of $Layer_{i+1}$ in $L(Goal_{i+1})$ is a single literal). In case one layer proves only a disjunction of goal instantiations, we need to refine LSA to support such a transfer, but this refinement can be done for any fixed size of disjunctions.

Theorem 6.4.4 (Output Completeness) *Assume that $T = \{Layer_i\}_{i \leq n}$ is a layered theory and $\varphi \in \mathcal{L}(Layer_0)$. If $T \models \varphi$, then there is $k \geq 0$ and a sequence of sentences $\varphi_k, \ldots, \varphi_0$ such that $\varphi = \varphi_0$, $Circ[Layer_k; \overrightarrow{C}_k; \overrightarrow{Z}_k] \models \varphi_k$, and for all i such that $0 \leq i < k$, φ_i includes predicate symbols only from $L(Goal_i)$ and*

$$Circ[Layer_i \cup \{\varphi_{i+1}\}; \overrightarrow{C}_i; \overrightarrow{Z}_i] \models \varphi_i.$$

PROOF See Appendix A.5.2.

Theorem 6.4.5 (Output Soundness) *Assume that there are sets of formulae $\varphi_n, ..., \varphi_0$ such that $Circ[Layer_n; \overrightarrow{C}_n; \overrightarrow{Z}_n] \models \varphi_n$ and for all i such that $0 \leq i < n$, φ_i is the set of prime implicates of $Circ[Layer_i \cup \varphi_{i+1}; \overrightarrow{C}_i; \overrightarrow{Z}_i]$ in $L(Goal_i) \cup L_f$ (L_f is the set of function and constant symbols in T). Then, $T \models \varphi_0$.*

PROOF See Appendix A.5.3.

6.5 Logical Layers for a Mobile Robot

In this section we describe the logical theories used in a control system we have implemented for a Nomad200 mobile robot operating in a multi-story office building. The Nomad200 is a cylindrical robot with sonar sensors on its perimeter, wheels that control its motion, and encoders that compute an estimate of the robot's position and angular heading (see Figure 6.5).



Figure 6.5: Nomad 200

The system includes five logical layers. A schematics view of the combined system is given in Figure 6.6. Each layer's body theory can be seen as having at two main parts that we categorize roughly into sensory-focused and goal-focused. Most theories also include other axioms that describe domain-dependent relationships in the world. This distinction into parts is not reflected in the system or in its behavior at the moment.

The following description uses upper-case letters to denote variables and lower-case letters for constants. We bring here only example axioms from each theory. The complete theories can be found in (Amir and Maynard-Reid II, 2001b). Also, currently, there is no low-level implementation of the elevator control instructed by layer 2. It is done by the human operator, if needed.

6.5.1 LAYER -1: Halt or Go

The lowest layer, *layer -1*, is responsible for sending actions to the robot or halting the robot. It has a theory similar to *layer 0* but with additional axioms that check if there are objects with which the robot is about to collide.

$$object(Obj) \wedge distance(Obj, Dist) \wedge Dist < min_dist \wedge$$
$$direction(Obj, Dir) \wedge (Dir < \tfrac{1}{3}\pi \vee Dir > 1\tfrac{2}{3}\pi) \Rightarrow$$
$$object_ahead$$

$$\neg HaltRobot \wedge \neg NeedTurn(Heading_angle) \wedge$$
$$NeedFwd(Heading_speed) \Rightarrow Fwd(Heading_speed).$$
$$HaltRobot \vee NeedTurn(Heading_angle) \Rightarrow Fwd(0).$$

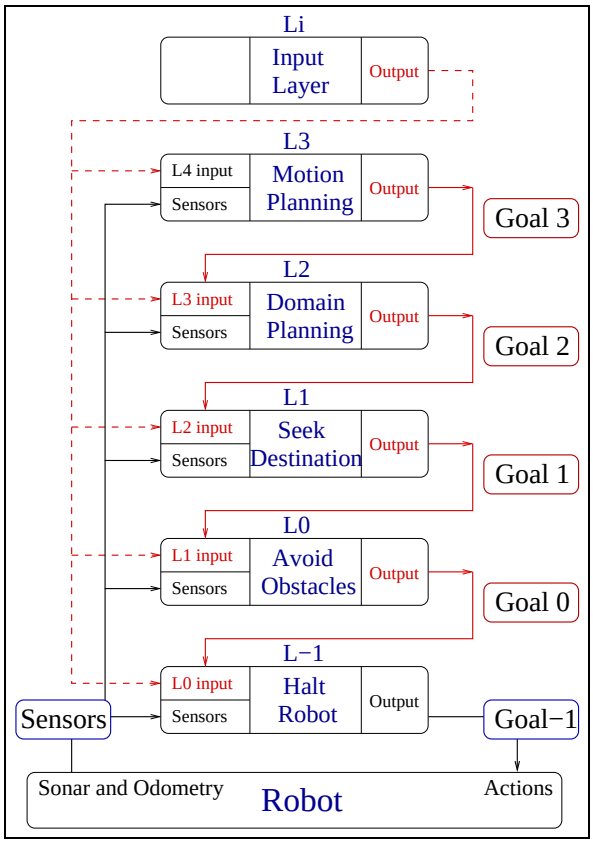

Figure 6.6: Diagrammatic view of an LSA system controlling a robot.

6.5.2 LAYER 0: Obstacle-avoidance

Layer 0 is responsible for deciding what low-level action the robot should perform. The theory has two main parts: sensory-focused and control-focused. We divide it to conceptually correspond to the modules shown in figure 6.1. During each cycle of layer 0, the theorem prover of layer 0 is asked to find the required actions for the modules *Turn* and *Forward* described below. It attempts to prove $Fwd(heading_speed)$ and $turn(heading_angle)$, where $heading_speed$ and $heading_angle$ are instantiated by the proof. The results are translated into the appropriate robot actuator commands. The inputs for this layer are the sonar data and the output from Layer 1. The input language includes the symbols $SonarReading$, $SonarDirection$, $Object$, $Direction$ and $Distance$. The output includes Fwd and $Turn$.

Sonar The sensory-focused part considers sensory input only from the sonars. It takes its input, asserted in the form of the axiom schema $SonarReading(sonar_number) = dist$, from the physical sonars and translates it into a map of objects, recording their distance and direction (relative to the robot)[1]. It may also discover "virtual" objects by way of layer 1's subsumption latch.

$$\begin{aligned}
&\forall dist, dir.\ (\exists sonar_number.\\
&\quad SonarReading(sonar_number) = dist \wedge\\
&\quad SonarDirection(sonar_number) = dir \wedge\\
&\quad dist \geq 0 \wedge dir > -\pi \wedge dir \leq \pi) \Rightarrow\\
&\qquad (\exists obj.\ Object(obj) \wedge Distance(obj) = dist \wedge\\
&\qquad\quad Direction(obj) = dir)\,.
\end{aligned}$$

We only have an implication from sonars to objects because we minimize $Object$ in our circumscription (see Section 6.5.6). For the same reason, we do not include axioms stating that there is at most one object at any location.

Runaway The control-focused decides which of the actions to perform, summing up the forces that the different objects around the robot exert on it (forces that are correlated to the distances of the objects to the robot). It uses the resulting force to determine whether the robot should turn or move forward and with what velocities.

$$\begin{aligned}
&heading_angle(A) \wedge need_turn(A) \wedge heading_speed(S)\\
&\quad \wedge need_fwd(S) \wedge angle_speed(Asp, A) \Rightarrow turn(Asp)
\end{aligned}$$

[1]The robot's 0-radians reference point is straight ahead, the front sonar is numbered 0, and the sonars are numbered consecutively, counter-clockwise from 0 to $NSONARS - 1$.

Feelforce Feelforce does the dirty work of computing the combined repulsive force from the different detected objects[2].

$$force_direction = \tan^{-1}(\frac{force_y}{force_x}).$$
$$force_strength = \sqrt{{force_y}^2 + {force_x}^2}.$$

$$force_x = Sum_x(0, 2\pi).$$
$$force_y = Sum_y(0, 2\pi).$$

Using the map, layer 0 executes the function $GetForce$, computing the combined "repulsive force" exerted on the robot by the detected objects as $Force_direction$ and $Force_strength$. It uses the former to specify a heading angle for the robot away from this force. Once headed in the right direction, the robot is commanded to move away at a speed proportional to the strength of the force, slowing down as it moves farther away from the objects.

$$Heading_angle = (Force_direction \bmod 2\pi) - \pi.$$
$$Heading_speed = Force_strength.$$

$$NeedTurn(Heading_angle) \Rightarrow Turn(Heading_angle).$$

$$\neg NeedTurn(Heading_angle) \wedge$$
$$\quad NeedFwd(Heading_speed) \Rightarrow Fwd(Heading_speed).$$
$$NeedTurn(Heading_angle) \Rightarrow Fwd(0).$$

6.5.3 LAYER 1: Destination-seeking

Layer 1 supports simple movements towards a goal location, more closely resembling the exploration layer of Brooks' system than the wandering layer. Given a particular pair of coordinates specified by the input $MoveCmd$ from layer 2 and given the orientation of the robot[3] and the distance and direction to the goal location it concludes the existence of a "virtual pushing object" in a particular location close to the robot, in the opposing quadrant.

[2]Feelforce may be implemented as a library function rather than as a logical theory, as it does not gain much by the logical representation, it can be implemented more efficiently as a procedure, and the logical representation uses some library functions anyway.

[3]These coordinates are with respect to the fixed coordinate system of the domain.

The theory can be seen to have two main parts: sensory-focused and goal-focused. The sensory-focused part translations the subjective odometry and direction of the robot to a global view of the robot in the world. The goal-focused uses to goal location, the distance to the goal and a set of quadrants to decide where to place a pushing object (if at all). For example,

$$\forall X, Y, X2, Y2, Qd \ \ curr_loc(X, Y) \wedge dest(X2, Y2) \wedge$$
$$quadrant(X - X2, Y - Y2, Qd) \Rightarrow push_object(Qd)$$

Simple Move

$$Object(PUSH_OBJECT).$$

$$\forall x_0, y_0, x, y.\ CurrLoc = \{x_0, y_0\} \wedge MoveCmd(x, y) \Rightarrow$$
$$HasPushObject(Quadrant(x_0 - x, y_0 - y)).$$

Push

$$\forall quad.\ HasPushObject(quad) \Rightarrow$$
$$Direction(PUSH_OBJECT) = quad * \tfrac{2\pi}{NQUADS} \wedge$$
$$Distance(PUSH_OBJECT) = PUSH_OBJ_DIST.$$

6.5.4 LAYER 2: Local Action Planning

Layer 2 is responsible for translating target landmarks into cartesian coordinates for the robot, and for planning and low-level interaction and control with the elevators. The theory can be seen as comprising four main parts: sensory-focused, motion-focused, elevator-focused and spatial relationships in the world.

The motion-focused subtheory uses a map and a simple axiom to translate landmarks to cartesian locations.

$$target_landmark(L) \wedge \neg close_to(L) \Rightarrow ShouldMove(L)$$
$$ShouldMove(L) \wedge \neg elevator_related(L)$$
$$\wedge cartesian(L, [X, Y]) \Rightarrow move_cmd(X, Y)$$
$$(\forall L \neg ShouldMove(L)) \wedge curr_loc(X, Y) \Rightarrow move_cmd(X, Y)$$

The elevator-focused part of this layer is a situation calculus theory (McCarthy and Hayes, 1969) (see also Chapter 3) with four main fluents: the location of the robot, the locations of the two elevators and whether two locations are visually connected. There are three action schemas: $moveto(L)$,

which moves the robot to location L; $callElev$, which calls the elevator; and $orderElev(floor(F))$, which commands the elevator to go to floor F. There are three effect axioms for the three schemas. For example, the situation calculus effect axiom

$$\begin{aligned}\forall F, S\ (&at(r, front(elev(floor(F))), S) \Rightarrow \\ &\exists S2, Ein_future_of(result(callElev, S), S2) \wedge \\ &\quad elevator(E) \wedge at(E, floor(F), S2))\end{aligned}$$

says that at some point after calling the elevator one of the elevators (there are two) would come ($in_future_of(S1, S2)$ is similar to $S1 < S2$ used in some papers on situation calculus (e.g., (Reiter, 1996; Lin, 1997; Lin, 1999)). For this situation calculus theory we need some frame or explanation-closure assumptions, which we add. Since the number of effect axioms and fluents is small, there is no harm in specifying frame axioms, which we do.

The sensory-focused and spatially-focused theories are similar to the ones used in *layer 3*. The main difference from *layer 3* is that the property of two locations being visually connected is now dependent on the situation (the elevators may be connected to their entrances or not).

The inputs for this layer are the current location data from the robot ($CurrLandmark$) and the output from layer 3 ($TargetLandmark$). During each cycle, it tries to plan for the next landmark and prove $MoveCmd(next_landmark)$.

$$Cartesian(RM218, [0, 0]).$$

$$\begin{aligned}\forall x, y, lm.\ &\neg NeedElevPlan \wedge TargetLandmark(lm) \wedge \\ &Cartesian(lm, [x, y]) \Rightarrow MoveCmd(x, y).\end{aligned}$$

$$\begin{aligned}\forall x, y, lm, s_{l1}, inter, s_1.\ &NeedElevPlan \wedge \\ &TargetLandmark(lm) \wedge At(ROBOT, lm, s_{l1}) \wedge \\ &FirstSit(s_{l1}, s_1) \wedge At(ROBOT, inter, s_1) \wedge \\ &Cartesian(inter, [x, y]) \Rightarrow MoveCmd(x, y).\end{aligned}$$

6.5.5 LAYER 3: Wide-Range Motion Planning

The top layer, *layer 3*, is responsible for high-level robot motion planning. The theory can be seen as comprising three main parts: sensory-focused, goal-focused and spatial relationships in the world. The goal-focused part represents the effects of robot motions in situation calculus. There is only one

fluent, the robot's location, and only one action schema, $moveto(L)$, where L is a location variable. For this simple situation calculus theory it is convenient to consider the actions as having duration and the situations as histories of actions from the initial situation, $S0$. This theory has a single effect axiom:

$$\forall L0, L, S\ at(r, L0, S) \wedge visuallyConnected(L0, L) \Rightarrow$$
$$at(r, L, result(moveto(L), S))$$

where $visuallyConnected(L0, L)$ means that there is a line of sight between $L, L0$. No frame axioms or explanation closure axioms are needed, as this effect axiom specifies the value of the only fluent in our theory.

$S0$ is considered to be the situation the robot is in when the layer receives the sensory and other input axioms. The sensory-focused part of the theory includes a representation of the relationships between landmarks in the world and the cartesian coordinates supplied by the robot's odometry sensors. Among others, the robot knows when it is between landmarks using the axiom

$$\begin{aligned} &visuallyConnected(P1, P2) \wedge curr_loc(X, Y) \wedge \\ &\quad cartesian(P1, C1) \wedge cartesian(P2, C2) \wedge C1 \neq C2 \wedge \\ &\quad pos_between(C1, [X, Y], C2) \Rightarrow \\ &\qquad current_landmark(between(Pos1, Pos2)) \end{aligned}$$

where $[X, Y]$ is considered between $C1, C2$ if it is close enough to the line crossing them.

Axioms for spatial relationships describe the relationships between rooms, room entrances, corridors, floors and elevators. For example, rooms are visually linked to their entrances, and locations that are in the same corridor are visually connected as well. Other axioms describe invariants of the domain, such as the commutativity of $visuallyConnected$, the fact that a position between two visually connected positions is visually connected to both positions.

6.5.6 The Default Assumptions of Robots' Layers

Each one of the layers has some default assumptions that it makes, as described in Section 6.3. In this section we list those nonmonotonic assumptions that are made by the layers of the system described in Figure 6.6.

Layer -1's Assumptions We add the circumscription formula

$$Circ[Layer_{-1}; HaltRobot; \overrightarrow{Z}_{-1}].$$

During each cycle of layer -1, it receives information as to the action it needs to take from layer 0. It then tries to prove that this action is indeed taken, if there is no need to halt the robot.

Layer 0's Assumptions We add the circumscription formula

$$Circ[Layer_0; Object; \overrightarrow{Z}_0].$$

During each cycle of layer 0, it applies the CWA to the symbols $Object$, $Distance$, $Direction$ in the input language. It then uses its theorem prover to try to prove $Fwd(speed)$ and $Turn(angle)$, where $speed$ and $angle$ are instantiated by the proof. The results are translated into the appropriate actuator commands.

Layer 1's Assumptions We add the parallel circumscription formula

$$Circ[Layer_1; MoveCmd; \overrightarrow{Z}_1].$$

During each cycle, layer 1's theorem prover attempts to prove $Object(obj)$, $Direction(obj) = dir$, and $Distance(obj) = dist$, and introduces them into layer 0's input latch if successful. The avoidance capabilities of layer 0 effectively push the robot away from the object in the direction of the goal, although it may deviate from a direct path if there are physical objects in the vicinity.

Layer 2's Assumptions We add the parallel circumscription formula

$$Circ[Layer_2; ShouldMove; \overrightarrow{Z}_2].$$

In the absence of a target landmark, we should stay in our place. Thus, we wish to minimize the movement of the robot, varying everything else that we know. If we prove that there is a target landmark, then the robot must move. Otherwise, it can stay put (notice that the layers below may have to take this decision into account and revise it, if there are other factors that they need to consider (e.g., objects about to collide with the robot)).

Layer 3's Assumptions We add the parallel circumscription formula

$$Circ[Layer_3; RobotLost; L(Layer_3)].$$

This layer tries to prove that the next action to be performed by the robot is a movement to a particular location. It minimizes the proposition $RobotLost$ such that the robot knows that it is lost if it could not determine where it is.

6.5.7 Layers Do Not Always Have a Model of Time

Many theories can be used to represent the effects of events and the flow of time (e.g., (McCarthy and Hayes, 1969; Reiter, 1991; Gelfond and Lifschitz, 1998; Miller and Shanahan, 1999; Doherty et al., 1998; Thielscher, 1998; Sandewall, 1994)). Typically, these theories are considered to have a fixed relationship to the world. For example, situation calculus includes a constant symbol $S0$ which refers to a particular situation.

In contrast to those, many of the theories described for the layers mentioned above do not include any notion of time. In particular, layers -1, layer 0 and layer 1 include very reactive-like theories. They are not concerned with the flow of time but rather with the current state of the world. For these theories time is implicit and so are situations.

Even layers 2 and 3, which have a dynamic picture of the world, do not reason about the flow of time before the current situation. At every cycle, they are asked to prove the goal for their respective layer. For that computation, they regard $S0$ as the current situation, and the sensors determine what we know about this situation. In particular, these layers do not include information from earlier times. This limits the amount of reasoning we can do with these layers, reprenting a tradeoff made by these layers to avoid costly computations.

Future layers (e.g., a *layer 4* or perhaps parallel layers to layer 3 and 2) can include such information, having different connections between their symbols and the environment. For example, such layers can regard $S0$ as a fixed situation attached to the beginning of the world of the robot (the time when the robot was turned on). This would entail regarding the current situation as the one that resulted from the actions performed by the robot and other events that occured since the beginning of the world of the robot ($S0$). This may also require some accounting of the time that passed since the beginning of an action or the beginning of time (as used in (Reiter, 1996), for example).

6.6 Implementation on a Mobile Robot

We have implemented the above architecture using the PTTP theorem prover (Stickel, 1992; Stickel, 1988b; Stickel, 1988a), on a cluster of Sun SuperSparc 1 stations running SWI Prolog or Quintus Prolog as the underlying interpreter for PTTP. The system runs on a Nomad 200 robot.

6.6.1 Choosing a Theorem Prover

Choosing a theorem prover is not easy. A theorem prover that is embedded in an autonomous agent needs to be a fully *automated reasoner* (no human intervention can be used here), needs to allow easy analysis of its proof progression (the system designer uses this information to change the theory or to detect problems in it), needs to allow for proof strategies (to be specified either by a human or by another layer), needs to allow for easy embedding of semantic attachments (and possibly theory resolution (Stickel, 1985)), and needs to allow for nonmonotonic reasoning (or at least some approximation of it).

Our theory includes several mathematical computations (such as several trigonometric functions) that are much better suited for a systematic algorithm than a theorem prover. Since we also wanted to have some algebraic sophistication in our theory, we needed semantic attachments. We examined many theorem provers and none of them seemed to support semantic attachments easily, nor did we have any convenient control over the theorem proving process (via strategies or otherwise).

Some of the provers that we examined more closely are Otter[4] (a resolution theorem prover), ACL2[5] (an industrial-strength version of the Boyer Moore theorem prover) and ATP[6] (a model elimination theorem prover). The major difficulties we encountered with them (not all difficulties were encountered with all) were the inability to append semantic attachments easily, complexity of making the theorem prover run on a given platform, the inability to control the inference process easily, and the lack of documentation.

In addition we also examined a few proof checkers such as PVS[7] (a proof checker), HOL[8] and GETFOL[9], all of which were found unsuitable due to their need for at least some human intervention.

PTTP (Prolog Technology Theorem Prover) is a model-elimination theorem prover using iterative deepening in the proof space. Given a theory made of clauses (not necessarily disjunctive) without quantifiers, PTTP produces a set of Prolog-like Horn clauses, ensures that only sound unification is produced, and avoids the negation-as-failure proofs that are produced by the Prolog inference algorithm. It is sound and complete for refutation in FOL (general first-order sentences are

[4] http://www-unix.mcs.anl.gov/AR/otter/
[5] http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html
[6] http://www.ksl.stanford.edu/people/axf/reference-manual.html
[7] http://www.csl.sri.com/pvs.html
[8] http://www.comlab.ox.ac.uk/archive/formal-methods/hol.html
[9] http://www-formal.stanford.edu/clt/ARS/Entries/getfol

translated into clausal form in the usual way, using Skolemization).

Compared to other theorem provers we examined (such as Otter[10] (a resolution theorem prover), ACL2[11] (an industrial-strength version of the Boyer Moore theorem prover), and ATP[12] (a model elimination theorem prover)), we found PTTP to be simple and easily customizable. Its close relationship with the underlying programming language (it has been implemented in both Prolog and LISP) allows for easy use of semantic attachments. Also, its output at run-time together with its iterative-deepening procedure allow for proof-progression analysis and control.

One of the features that we liked the most about the PTTP was that, despite the lack of suitable documentation (although there are a fair number of examples), the theorem prover is very easy to customize and its reliance on the underlying language (it was implemented in both PROLOG and LISP) allows the easy use of semantic attachments. The collection of examples together with the PTTP software is very illustrative and, despite some difficulties incurred by the use of semantic attachments and built-in predicates (such as the algebraic $<$ relation), it gave us relatively few troubles in either installation or use.

6.6.2 Offline Experiments with PTTP and Simulated Sensors

	Layer 0				Layer 1				Layer 2		Layer 3	
	Time		Infer.		Time		Infer.		Time	Infer.	Time	Infer.
	Mean	SD	Mean	SD	Mean	SD	Mean	SD				
Scen. 1	0.09	0.02	3598	629	0.02	0.01	394	2	0.01	4	0.00	20
Scen. 2	0.10	0.01	3703	613	0.02	0.01	384	4	0.52	27184	0.47	34056
Scen. 3	0.09	0.02	3575	640	0.02	0.01	389	1	0.00	4	11.24	694966

Table 6.1: Proof time and inference steps measurements for the LSA during experiments in three different scenarios: (1) single-floor planning, (2) lower-level elevator planning, and (3) multi-floor planning. (*SD* is *standard deviation*.)

We subjected our system to a set of experiments in a simulated office building environment. Table 6.1 summarizes the results for three scenarios of varying difficulty: (1) planning a path towards a location on the same floor as the robot, (2) creating a plan that requires a low-level plan for using the elevator, and (3) planning a path towards a location on a different floor. In each scenario, we experimented with various robot orientations and obstacle positions in the robot's vicinity. For each

[10] http://www-unix.mcs.anl.gov/AR/otter/

[11] http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html

[12] http://www.ksl.stanford.edu/people/axf/reference-manual.html

layer, we measured the number of inference steps and time taken to prove its goal[13] (results from a sample run are shown in figure 6.7 and 6.8)

Layer 0, the critical layer, achieved its results in an average of 0.1 seconds when a turn action was required, and 0.3 seconds when a forward action was required. (Because of space concerns, we have included in Table 6.1 only the data for cases of the former kind.) Layers 1, 2, and 3 worked fairly fast, although the long planning involved in Scenario 1 took more than 10 seconds (for a depth of 30 in the proof space). However, because we rely on the speed of only layer 0, safety is not compromised; the avoidance capabilities ensure that the robot does not fall off a cliff while planning a way to avoid the cliff edge.

We attribute the speed achieved to three optimizations. First, we used a few semantic attachments in Layer 0. In particular, the predicate $GetForce$ was embodied in a C function that returns the force vector $[Strength, Direction]$. It calls Prolog's bagof operator to collect all the objects for which existence proofs can be found, then computes the sum of the forces contributed by each object. This CWA is achieved by limiting proofs to be no longer than a specified constant (after some experimentation, we settled on a constant of 20).

Second, we applied caching to the proof of $GetForce$. Since every proof "re-proved" $GetForce$ many times, this improved the performance of Layer 0 significantly (from approximately 10 seconds to 0.1 seconds per proof).

Third, we divided the planning so that layer 2 executes "local planning" for the elevator domain. This allowed layer 3 to avoid an explosion of the proof space, which otherwise would have occurred since there are four principal actions as well as a number of frame axioms associated with the robot and the elevator. The separation also helped prevent complex unifications.

```
        curr_loc(0, 0),
        curr_dir(-1.5),
        current_landmark(rm218),
        sonar_reading(0, 0.5),
        sonar_reading(1, 2),
        sonar_reading(2, 2),
        sonar_reading(3, 2),
        sonar_reading(4, 2),
        sonar_reading(5, 2),
        sonar_reading(6, 2),
        sonar_reading(7, 2),
```

Figure 6.7: An example of simulated sonar input asserted in the theory.

[13] We do not list averages or standard deviations for layers 2 and 3 because their performances are independent of both the robot's orientation and sonar readings.

```
| ?- prove((turn(A); fwd(S))),
       print('turn angle = '), print(A), nl,
       print('fwd speed = '), print(S)
search for cost 0 proof...  3 inferences so far.
search for cost 1 proof...
.
.
.
search for cost 9 proof...
[MAIN PROOF]
Proof:
Goal# Wff# Wff Instance
----- ---- ------------
  [0]   0  query :- [1].
  [1]  34    turn(-314) :- [2] , [5].
  [2]  32      heading_angle(-314) :- [3] , [4].
  [3]  41        not_ab_avoid.
  [4]  33        get_move_dir(0,-314).
  [5]  36      need_turn(-314) :- [6] , [7].
  [6]   5        min_angle(5).
  [7]  32        heading_angle(-314) :- [8] , [9].
  [8]  41          not_ab_avoid.
  [9]  33          get_move_dir(0,-314).
Proof time: 12664 inferences in 0.24 seconds,
including printing
turn angle = -314
fwd speed = _6872
```

Figure 6.8: An example proof by the theory.

6.6.3 The Software and Dynamics of the System

Our implementation is written in C++ with classes allowing prover-specific implementation: *Layer* is the superclass of *Layer_qp* (Quintus prolog with PTTP), *Layer_swi* (SWI prolog with PTTP) and *Layer_input* (a layer used in an executable that allows the user to send new axioms to the other running layers). An executable consists of a *Layer* object (the central piece of the executable), objects for communication (TCP/IP), and an embedded theorem prover (in the case of PTTP, this means an object file consisting of a prolog implementation and a compiled PTTP). There is a separate layer for the communication with the robot that translates *layer -1*'s proven goals to robot motion commands, and sends sensory information to the layers on request.

Each time a layer is run (with whichever theorem prover implementation) a configuration file specifies the theory it should initially load and the communication pattern of the layer. The communications part specifies the layers from which it should accept axioms, the host/port of that layer, and the mode of communication (is it synchronous (request data explicitly) or asynchronous (use the data in the latch)).

After initializing the communication it initializes the theorem prover and loads the body theory into it. Then it runs the following infinite loop: First, the layer reads the messages that are on the ports and asserts the latest ones from each port into the theorem prover; then, the layer attempts to prove the goal; finally, upon successful conclusion it send the result of the proof to listening layers below it.

The information that the layer reads from the ports overrides previous latch data. However, if no information arrived on a specific port, we reuse the information that arrived previously. This allows the layers to work in different frequencies without confusing delay in communication or computation for a directive that overrides the latest information from that layer. To avoid ambiguity, most layers prove *layer_i_failed* if they failed to prove the goal (depending on the defaults asserted for each layer). In that case, this proved assertion is sent to the listening layers, which use this message to override previously received assertions.

6.6.4 LiSA's behavior

We ran several experiments on LiSA with goals of traveling to different rooms in the building. Figure 6.9 presents average total time measurements for each layer during these experiments. This figure shows that the current implementation is somewhat slow for a truly reactive behavior. A few improvements can be made to the implementation that would make it faster (e.g., use a faster theorem prover, faster hardware, and a few optimizations of the code), but these are outside the scope of this thesis.

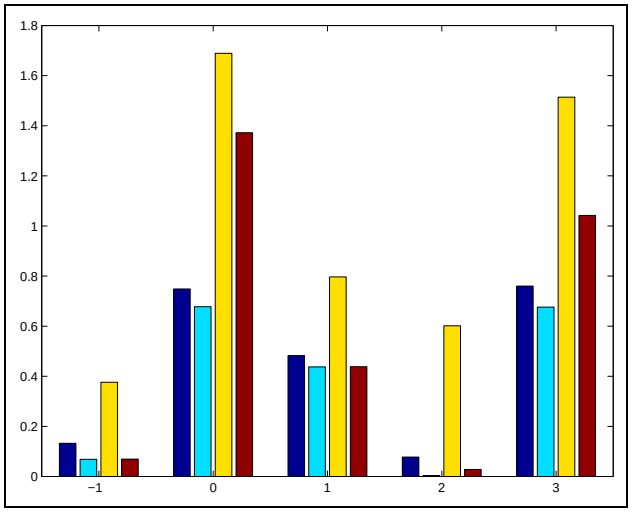

Figure 6.9: Average time (in seconds) per cycle for each of the layers[15]. The bars for each layer correspond to total cpu-time per cycle, cpu-time for proofs in a cycle, total clock time per cycle and clock time for proofs in a cycle, in this order from left to right.

Each experiment with LiSA starts with running the logical layers, the nomad layer and the input layer. LiSA has a given map that is used by layers 2 and 3. We reset the robot in a position and

heading that matches this map. Using the input layer, we tell the robot that its goal is to navigate to one of the rooms or across the lab. It takes from a few seconds to a minute before layer 3 finds a plan from its current location, and sends a goal landmark to layer 2. That one instantaneously translates the landmark into a goal location and sends it to layer 1. Layer 1 then provides a pushing object to layer 0. Layer 0 sends a motion command to layer -1, and layer -1 executes it, if there are no direct obstacles in front of it. The robot typically starts turning until it faces in the direction that it intends to go and then proceeds forward towards that target. The transition between turning and moving forward is smooth and without delays.

In our experiments, LiSA took from 30 seconds to two minutes to move from one landmark to the next, with landmarks that are approximately 4-10 meters apart and with obstacles that are close to its path. When we put obstacles such as chairs and humans in front of the robot, it managed to go around them without colliding or hesitating.

In our earlier experiments, before we improved the theory of layer 3, LiSA sometimes got *lost*. If an obstacle was put in LiSA's path and she had to go around it, the next time layer 3 tried to plan a path to the goal it sometimes did not know where LiSA is, as she was not close to any landmark (and may have moved significantly away from the path to the next landmark). In that case, we were able to use the input layer and send an axiom into LiSA's layer 3 telling it that it is between the two landmarks. Currently, LiSA's layer 3 re-uses the last proved goal landmark as the default that is sent to layer 2 if layer 3 fails. This sidesteps the problem, but is not consistent with our semantics, as our theoretical layers have no *memory* or *state*. We plan to extend the system and the semantics using models of belief update to allow memory and belief change in a consistent manner.

6.6.5 Tuning the System

Theorem provers are notoriously slow, which is one of the main reasons they are usually not used for time-sensitive applications. However, in this implementation we were able to sidestep this difficulty by using only small and simple theories and using a fast approximation of nonmonotonic reasoning to conclude defaults. We attribute the speed achieved by our system to several optimizations that we mention below.

During our experiments it became clear to us that much of the bottleneck in some layers is in concluding that there is no proof. Without this conclusion the layer cannot terminate the cycle and start a new cycle (with new sensory information and new latch information). For this reason, each layer has an associated limit on the depth of the search allowed for a proof of the goal. If no proof is found up to this depth, we conclude that for any run-time purpose there is no proof. This depth

limit is determined experimentally and is specified together with the goal formula.

Depth limit is also used to implements a rough approximation for defaults. For example, layer 0 implements the CWA for objects by aggregating all the objects that it can find up to a specified proof depth. Under the same assumption we consider the depth limit on the proof of the goal as a default that says that the goal is false by default.

The same mechanism allows us to provide *islands* in the search space of the prover. For example, in layer 3 we first try to prove that the robot is currently at a particular location. If we succeed, we add the assertion $at(r, CurrLandmark, s0)$ as a temporary new axiom to the prover, where $CurrLandmark$ is the landmark that we found the robot to be in. This sometimes cuts the depth of the proof search space by 10, which has a significant influence on the proof time (cuts the proof time in those cases by a factor of approximately 10). The applications of caching to the proof of get_force is also of the same nature. Since every proof in layer 0 re-proves get_force many times, this improved the performance of Layer 0 significantly (from approximately 10 seconds to 0.1 seconds per proof on a Sun UltraSparc 60).

Finally, we use a semantic attachment in Layer 0 for the predicate get_force. It is embodied in a C function that returns the force vector $[Strength, Direction]$. It calls Prolog's setof operator to collect all the objects for which existence proofs can be found, then computes the sum of the forces contributed by each object.

6.7 Related Work: Cognitive Robotics and Subsumption Architectures

Compared to other approaches to robot-control that use logic, the LSA is the only system using full FOL theorem provers for reasoning and control. This is the first presentation of a robot-control architecture that is built on theorem provers and is suitable for realizing complex tasks in real time.

Shanahan (Shanahan, 1996) describes a map-building process using abduction, and then implements his theory in an algorithm that is proved to have an abductive semantics. Later works (Shanahan, 1998a; Shanahan and Witkowski, 2000; Shanahan, 2000) use *abductive logic programming* to solve planning and sensory problems for mobile robots. In particular, some of these implemented systems use hierarchical planning and can deal with some noisy sensory data. The approach described there uses online hierarchical planning, allowing fast planning and execution monitoring. This work chooses to use logic programming tools (especially Prolog) instead of general-purpose theorem provers. In comparison, the work we describe in this chapter is based on the expressivity

of FOL theorem provers, and makes no assumptions as to the form of the axioms or appeal to restricted expressivity. The form of hierarchical planning used in our work somewhat predates that of (Shanahan, 2000).

Baral and Tran (Baral and Tran, 1998) define control modules to be of a form of Stimulus-Response (S-R) agents, relating them to the family of action languages $\mathcal{A}$ (e.g., (Gelfond and Lifschitz, 1993; Giunchiglia et al., 1997)). They provide a way to check that an S-R module is correct with respect to an action theory in $\mathcal{A}$ or $\mathcal{AR}$ and provide an algorithm to create an S-R agent from an action theory.

Systems based on the GOLOG project (e.g., (Levesque et al., 1997; Giacomo et al., 1997; Giacomo et al., 1998; Reiter, 1998; Lakemeyer, 1999; Grosskreutz and Lakemeyer, 2000; Boutilier et al., 2000)) have a planner that computes/plans a high-level GOLOG program off-line. This plan is specified using the situation calculus ((McCarthy and Hayes, 1969)) and is also given semantics using it. The plan is executed and monitored later by the robot. This family of systems allows plans that are specified on a high level, having nondeterministic actions and actions that require further elaborations. During execution the GOLOG controller uses the plan specification to guide a search in the plan space as needed. This allows the efficient absorption of sensory information, plans that require some limited planning (or re-planning) during execution and the consideration of events that do not depend on the robot (natural events). Logic and situation calculus ((McCarthy and Hayes, 1969)) are used to give semantics for GOLOG programs, keeping the system formally clean.

Another, somewhat earlier line of work, concentrated on compilation of logical description into control languages (e.g., (Kaelbling, 1987; Kaelbling, 1990; Kaelbling and Rosenschein, 1990; Rosenshein and Kaelbling, 1995)). This approach takes a description in logic of the planning problem, the condition-achievement goal, or the condition-maintenance goal. It uses this representation to create a rule-based representation that can be readily used to control an agent in real-time. This approach provides a way to specify agents using logic or a formal model and then to compile this representation to a reactive one.

Compared to all of this work, our system is the first to allow declarative representation and reasoning in real time, enabling the full power of first-order logic. Also, being able to send logical-formulae advice to the robot at run-time is a property that has not seen (to our knowledge) since Shakey the robot (Nilsson, 1984).

Compared to subsumption systems for robot control (e.g., (Brooks, 1990b; Brooks and Flynn, 1989; Matarić, 1992; Horswill, 1993; Brooks and Stein, 1994)) our system allows the user to send new axioms to each of the layers as the robot is running. This allows the user to give advice to the

robot and to correct behaviors in runtime. In addition, our system has no voting scheme for deciding on the behavior that should be followed. Instead, the layers work in synergy, sending messages to each other, together providing the compound behavior.

Clearly, we have not solved the age-old problems with using theorem provers, and there are limitations to our approach. However, with proper tuning and given recent advances in automated reasoning, this kind of system seems to support high-level reasoning that is still reactive, offering a major advantage to robotic systems and systems that wish to perform commonsense reasoning online.

6.8 Conclusion

In this chapter we have shown that theorem provers can be used for robot control by employing them in a layered architecture. We demonstrated that the architecture and the versatility of theorem provers allow us to realize complex tasks, while keeping individual theories simple enough for efficient theorem proving. Furthermore, we have grounded our proposal by giving it formal semantics based on circumscription.

There are many important avenues for extending this system. *Memory* and *state* can be added to the system easily, but they do not fit the current semantics. We plan to use belief update semantics to extend this framework and allow such modifications as defaults that change according to the beliefs of the robot and diagnosis of the robot behavior and whereabouts for determining its location. Also, I wish to create such reactive systems automatically from first-order theories that describe the intended behavior. We plan to pursue these goal in the near future.

In the immediate future I plan to add layers that create maps and layers that reason about and update explicit beliefs about the world. I am also working on incorporating vision sensory capabilities.

This work is a step towards our long-term goal of creating a general logic-based AI architecture that is efficient and scalable, and that supports reactive common-sense reasoning.

Chapter 7

Conclusions

7.1 Conclusions From This Work

There are two main conclusions from this work. First, we can use the structure of logical knowledge bases in ways similar to those used in other parts of AI. These should allow us to build large KBs and use them in everyday applications. Second, we can build agents that are based on logic and automated reasoners and that can act reactively in the world.

We can build large KBs using object-oriented design techniques without giving up the representation power of FOL and without adding complexity to our representations. Such KBs are easier to build, change and reuse, as was demonstrated in Chapter 3. They are also easier to reason with automatically, using the algorithms we developed in Chapter 4. Finally, they can easily be used in reactive architectures for agents.

Consequently, a viable way for building agent control architectures is to design components in logic, initially paying little attention to the complexity of solving the combined task or the efficient implementation of each of the components. The combination of those components in an AI architecture can be tested and applied using the simple techniques we presented in this thesis.

In the pursuit of human-level artificial intelligence, we can propose a method for the combination of different approaches to AI. This allows a more refined investigation than using a single representation and reasoning paradigm for an agent. One way to combine different approaches, advocated by this thesis, is to view these approaches as special cases of logic, determine the way that the combined system will work (each component viewed as its semantically-equivalent logical component) and specialize the components to achieve their function in the combined system using the original representation for each component.

7.2 Summary of Results

Logical theories for AI are coming back into favor, as we are beginning to find important practical uses in AI to which logic can be put. This phenomenon reverses a trend of the last ten years that resulted from computational intractability problems in reasoning with logic. This thesis has shown that recent methods used elsewhere in AI (namely, decomposition and structured reasoning) can be applied to logic with remarkable results for reasoning, knowledge representation and control of intelligent agents. It advances the use of logic in everyday applications in this way.

This work has shown how to automatically decompose logical theories into trees of smaller theories using network-flow techniques (Chapter 5). The algorithms I provided approximate the optimal by constant factors and are faster than previously available algorithms for equivalent tasks. I have also shown how to use these and man-made decompositions to speed up reasoning (Chapter 4). I have presented evidence that much of the commonsense knowledge that we may wish to use in practical applications has this kind of structure (Chapters 2, 3, 5).

The reasoning procedures I provided scale up reasoning from small theories, containing tens of axioms, to large theories, containing thousands of axioms, in a sound and complete way. These procedures are supported by computational complexity results and experiments (Sections 4.3.5, 4.4.2, 4.5). The reasoning task are typically performed using message-passing between the partitions. The reasoning that occurs in each of the partitions is done using standard reasoning techniques in logic.

This thesis also showed how to use decompositions of logical theories as a basis for artificial intelligence architectures. In particular, an architecture of this sort has been implemented for a mobile robot, and several experiments in an office environment have shown the feasibility of this approach (Chapter 6). I have also presented an adaptation of the object-oriented design technique to FOL, facilitating the creation of large theories that utilize and describe multiple domains. I have presented a simple adaptation of FOL and showed how design using the OOFOL methodology is performed more easily and efficiently (Chapter 2).

7.3 Future Directions and Open Problems

7.3.1 Using Commonsense Reasoning in AI Applications

This work gives rise to several related projects that are focused on representations of commonsense knowledge, reasoning with them, and building reactive high-level reasoners that use commonsense knowledge and ground this knowledge in the world (this has recently been called *connecting sensors*

and symbols). There are also interesting problems in the continued pursuit of to improve automated logical reasoning, AI architectures and mobile robots. I list some of these directions below.

Commonsense theories represent knowledge about the world. The CYC project and several recent DARPA-funded projects, such as HPKB and RKF, try to build large commonsense theories that are reliable, comprehensive and not brittle. This is a difficult task. We need methodologies and theories of knowledge-base change to allow the creation of large bases of commonsense knowledge.

One tool that is needed for building large KBs is a theory of elaboration tolerance (McCarthy, 1998). The theory should provide guidelines for writing axiom groups that are easier to change given distributions of predicted changes. The object-oriented methodology for logical KBs is a tool that should play an important role in this work.

Reactive systems that perform high-level reasoning can be created automatically from large theories describing intended behaviors. First, a decomposition of those theories produces a network of interacting sets of axioms. Then, some of the sets are assigned goals that they should prove (with some open variables) and default assumptions that they should assume (given no input messages from other sets of axioms). I have shown already that this kind of architecture can perform reactive assignments, such as navigation and control of a mobile robot in real-time.

It is easy to find applications that will utilize this kind of architecture for high-level reasoning. For example, if one robot follows another, and the latter goes out of sight, then the former can reason about the whereabouts of the former and locate it. It can also warn a human entering a room about dangers, if it concludes that the human does not know about those dangers, given what the robot knows about the knowledge of the human. The development of algorithms for the automatic creation of such reactive systems from specifications is important. It is also interesting to correlate the structure of such systems and the organization of human or mammal knowledge in the brain.

Another key problem in AI today is that using vision systems together with knowledge representation systems is difficult. It is not easy to instruct vision systems to obtain any particular information. For example, it is difficult to instruct a vision system to evaluate the existence of a particular relationship between objects. It is similarly difficult to use knowledge hidden in a knowledge representation system to help the vision system. Some very simple results have been obtained lately in this area, mostly in relating symbols in knowledge representation systems to features and objects detected by vision systems. Working in this direction, with the long-term goal of allowing natural-language description of pictures, is important. For example, an output for a picture might be: *a man sitting on a chair and writing a letter*.

One way to approach the problem of combining knowledge bases and vision systems follows

from this thesis. We can view both kinds of systems as knowledge systems that are connected to the world and use the world in changing their knowledge. We can model the interactions between these systems using OOFOL and find those messages that need to be sent between them using our MP algorithm or a variant of it that is more suitable for this application.

The design methodology for logical knowledge bases that we propose in this thesis complements the object-oriented design methodologies in Bayes nets, frame systems, and programming languages. The compatibility of these paradigms offers the opportunity to use many representation and reasoning methods in a single agent. Once the knowledge-structure of the agent is set, each of these systems can fit each of the objects in the network. Each object is matched with the system that most conveniently matches the object's tasks.

Finally, there are several simple observations coming from this work. The design methodology and partitioned reasoning presented in this thesis can be extended to modal logics and to agents that use multiple representation and reasoning methods. It would also be interesting to develop a theory of elaboration tolerance for commonsense theories and apply the distributions of knowledge-base change to the design of large object-oriented KBs. Lastly, representing planning problems using classical representations in logical AI and probabilistic AI allows decompositions and planning algorithms that have not been available so far.

7.3.2 Inference and Structure

This thesis has investigated the possibility of building or automatically partitioning KBs in a way that yields partitions of axioms that exhibit close relationships among the axioms inside each partition and only a loose relationship between axioms in different partitions. It developed algorithms that take advantage of such structures. However, this partitioning is sometimes not enough or not suitable for our KBs. In particular, if the tree of partitions generated by the structure available in these KBs has large connections between the partitions, then the simple algorithms we provided may not yield the anticipated performance improvement.

Instead of the structure used in this thesis, we can use different kinds of dependencies that the KBs allow us to use. For instance, we may *unroll* the FOL structure into one that puts different terms in different partitions, taking advantage of the unrolled structure to find partitions that are more loosely related than before. Also, we may sometimes want some back-and-forth messages between the partitions. Some proofs may be shorter or would be more intuitive if we allow such exchange of messages between adjacent partitions.

There may also be algorithms and assumptions that would allow us to reason with sets of partitions and associated interaction graphs that have loops. For example, if we know that a certain partition, is convex in some nonlogical symbols (i.e., for every disjunction $P \vee Q$ that is a valid consequence of this partition either P is a valid consequence or Q is a valid consequence), then we may allow the graph to have a loop with this partition, if the loop is the result of some of those symbols appearing on two edges in that loop connected to that partition. For example, if we know that, as a result of reasoning, we would find either a plan that would provide $subgoal_A$ or a plan that would provide $subgoal_B$, we may *split* on these subgoals without needing to consider the possibility that we would have to reason with the sentence $subgoal_B \vee subgoal_A$.

Finally, we may also find it useful to isolate a subproblem that has some characteristics (e.g., is the most constrained or the most likely to solve our problem) and then try to solve our original problem by solving this subproblem first. For example, we may use recent observations about the relationship between the ratio of number of clauses to the number of symbols in a theory and the probability that the theory is satisfiable (Mitchell et al., 1992). In this case, we may look for the most constrained subproblem (the one with the highest clause-to-variable ratio) and solve it first (Amir and McIlraith, 2001b).

7.3.3 Machine Learning Using Partitions

Automatic learning of knowledge bases of logical axioms can benefit from viewing them as made of interacting partitions, the way presented in this thesis. We can learn larger knowledge bases and higher-level concepts and axioms.

For example, an agent can use an object-oriented knowledge base as a skeleton for additional axioms that it would learn from its environment. In the context of learning axioms for a particular object (in our object-oriented view of logic), this process can be seen as analogous to *parameter learning* in Bayes nets. Concept learning can be used to learn high-level predicates and axioms, as seen in (Roth and tau Yih, 2001). Further refinements can be made per object (or partition, if those partitions were generated automatically) instead of given the entire knowledge base. A small fragment of the knowledge base can be used to evaluate a learned concept, instead of using the entire KB as in current methods. A save in computational effort when learning into the KB can also be made using our reasoning algorithms on the resulting structure instead of current straight-forward algorithms.

Learning the structure of the interactions between objects can be modeled along the lines of EM (*expectation maximization* algorithm) for structural learning of Bayes nets. Connections between

OOFOL objects can be added or removed using a hill-climbing strategy, resulting finally in a large KB.

Appendix A

Proofs

A.1 Proofs for Chapter 2: OOFOL

A.1.1 Theorem 2.5.2: Equivalent Semantics for Directional OOFOL

The proof relies on the following lemma. It says that we can weaken the circumscribed theory or the set of varied constant symbols without losing models.

Lemma A.1.1 *Let A be a FOL theory and P, Z, Z' be vectors of non-logical symbols, and assume P has only predicates. For a structure M, if $M \models Circ[A; P; Z, Z']$, then $M \models Circ[A; P; Z]$.*

PROOF Let $M \models Circ[A; P; Z, Z']$. Then M is a $\leq^{P;Z,Z'}$-minimal model satisfying A ($M \leq^{P;Z,Z'} M' \iff (|M| = |M'| \wedge \forall C \notin P \cup Z \cup Z', C^M = C^{M'} \wedge P^M \subseteq P^{M'})$). If $M' <^{P,Z;Z'} M$ such that $M' \models A$, then $|M'| = |M|$, $P^M \subsetneq P^{M'}$ and for every constant symbol C not in P, Z, Z', $C^M = C^{M'}$. Thus, in particular, for every constant symbol C not in P, Z, $C^M = C^{M'}$. Therefore $M' <^{P,Z} M$ and we have contradiction to M being $\leq^{P;Z,Z'}$-minimal model satisfying A. ■

The converse is not true, as it entails monotonicity of circumscription. For our specific case, though, we can get a version of the opposite direction. Let $\mathcal{A}(Obj) = \widetilde{\mathcal{A}}(Obj, Obj)$ and $\mathcal{L}(Obj) = \mathcal{L}(\mathcal{A}(Obj))$.

Our theorem can be restated as follows.

Theorem A.1.2 *Let T be a totally ordered OOFOL theory. $\mathcal{M}$ is a model of*

$$\bigwedge_{i \leq n} Circ[\widetilde{\mathcal{A}}(O_i, O_{i+1}); Ab_i; \mathcal{L}(O_{i+1})]$$

if and only if it is a model of

$$\bigwedge_{i\leq n} Circ[flat(T); Ab_i; Ab_{i+1}, ..., Ab_n, \mathcal{L}(O_{i+1}), ..., \mathcal{L}(O_n)].$$

PROOF The backward direction follows directly from lemma A.1.1 and noticing that

$$Circ[flat(T); Ab_i; \mathcal{L}(O_{i+1})] \equiv Circ[\widetilde{\mathcal{A}}(O_i, O_{i+1}); Ab_i; \mathcal{L}(O_{i+1})] \wedge flat(T)$$

(See formula (3.2) in (Lifschitz, 1993)).

For the forward direction, assume that

$$M \models \bigwedge_{j\leq n} Circ[\widetilde{\mathcal{A}}(O_j, O_{j+1}); Ab_j; \mathcal{L}(O_{j+1})], \text{ and}$$
$$M \not\models Circ[flat(T); Ab_i; Ab_{i+1}, ..., Ab_n, \mathcal{L}(O_{i+1}), ..., \mathcal{L}(O_n)]$$

for some $i \leq n$. Let i be the first (smallest) such index. For brevity, let us write Z for $Ab_{i+1}, ..., Ab_n$, $\mathcal{L}(Obj_{i+1}), ..., \mathcal{L}(Obj_n)$. $M \models flat(T)$, because $flat(T) = \bigcup_i \widetilde{\mathcal{A}}(O_i, O_{i+1})$. Thus, there is a model $M' <^{Ab_i;Z} M$ such that $M' \models flat(T)$.

Take M'' such that $|M''| = |M|$ (i.e., they have the same universe), the interpretation of all the symbols of $\mathcal{L}(T)$ other than $\mathcal{L}(O_{i+1}), Ab_i$ is identical to M and the interpretation of $\mathcal{L}(O_{i+1}), Ab_i$ is as in M' (this is sound since $|M'| = |M|$ by the conditions of $M' \leq^{Ab_i;Z} M$).

$M'' <^{Ab_i;Ab_{i+1},\mathcal{L}(O_{i+1})} M$, because $|M| = |M''|$, $C^M = C^{M''}$ for every constant symbol C not in $\mathcal{L}(O_{i+1})$ (since this is how we set the interpretation for these symbols in M'') and $Ab_i^{M''} \subsetneq Ab_i^M$ (since $M' <^{Ab_i;Z} M$). To see that $M'' \models \widetilde{\mathcal{A}}(O_i, O_{i+1})$, notice that M'' is identical to M' on $\mathcal{L}(\widetilde{\mathcal{A}}(O_i, O_{i+1}))$.

Thus, M is not a model of $Circ[\widetilde{\mathcal{A}}(O_i, O_{i+1}); Ab_i; \mathcal{L}(O_{i+1})]$, contradicting our assumption. ■

A.1.2 Theorem 2.5.3: Completeness Proof for Directional OOFOL

To prove the completeness theorem for MESSAGE-PASSING (Theorem 2.5.3), we need several lemmas.

Lemma A.1.3 *Let O_1, O_2 be two objects and $\mathcal{M}$ a model of $\widetilde{\mathcal{A}}(O_1, O_2)$. Assume that for every sentence $\phi \in \mathcal{L}_{O_1,O_2}$, $\mathcal{M} \models \phi$ iff $\mathcal{M} \models \phi'$, where ϕ' is the translation of ϕ from O_1 to O_2. Then, there is a model $\mathcal{N}$ such that for all $ab \in Ab_{O_1,O_2}$, $\mathcal{N} \models \forall x.\neg ab(x)$ and $\mathcal{N} \leq^{Ab_{O_1,O_2};\mathcal{L}(O_2)} \mathcal{M}$.*

PROOF Assume that $\mathcal{L}$ has a single sort, that the only link between O_1, O_2 is a single predicate P_1, P_2 of O_1, O_2, respectively, whose parameter is a single variable (we generalize over all of these assumptions at the end of the proof).

Let $||\mathcal{M}||$ denote the cardinality (finite or infinite) of the universe of $\mathcal{M}$ and, for a predicate P, let $|P|^{\mathcal{M}}$ denote the cardinality of $\{x \mid x \in |M|\ P^{\mathcal{M}}(x)\}$.

If $|P_1|^{\mathcal{M}} = |P_2|^{\mathcal{M}}$ and $|P_1^c|^{\mathcal{M}} = |P_2^c|^{\mathcal{M}}$ ($P^c(x) \iff \neg P(x)$), take f to be a bijection from $|\mathcal{M}|$ to itself such that f is a 1-1 and onto mapping of the elements of $P_1^{\mathcal{M}}$ on those of $P_2^{\mathcal{M}}$. Let $\mathcal{N}$ have the same universe and the same interpretation of $\mathcal{L}(O_1)$ as $\mathcal{M}$. For every relation R in $\mathcal{L}(O_2)$, let $R^{\mathcal{N}}(x_1, ..., x_k) \iff R^{\mathcal{M}}(f(x_1), ..., f(x_k))$. Finally, let $\forall x \in |\mathcal{N}|.\neg ab^{\mathcal{N}}(x)$. This model is easily seen to be the required one.

Otherwise, $|P_1|^{\mathcal{M}} \neq |P_2|^{\mathcal{M}}$ or $|P_1^c|^{\mathcal{M}} \neq |P_2^c|^{\mathcal{M}}$. If there is a model $\mathcal{M}'$ of O_2 such that $|\mathcal{M}| = |\mathcal{M}'|$ and the cardinalities of P_2, P_2^c match those of P_1, P_1^c as above, then we take $\mathcal{N}$ as in the previous case, setting $R^{\mathcal{N}}(x_1, ..., x_k) \iff R^{\mathcal{M}'}(f(x_1), ..., f(x_k))$, when f is a bijection as above between $|\mathcal{M}|$ and $|\mathcal{M}'|$ that is 1-1 and onto from $|P_1^{\mathcal{M}}|$ to $|P_2^{\mathcal{M}'}|$. Finally, let $\forall x \in |\mathcal{N}|.\neg ab^{\mathcal{N}}(x)$. This model is again easily seen to be the required one.

Otherwise, let $\kappa, \kappa', \kappa''$ stand for the cardinalities of $|\mathcal{M}|$, $(P_1)^{\mathcal{M}}$ and $(P_1^c)^{\mathcal{M}}$, respectively.

Assume that all of $\kappa, \kappa', \kappa''$ are finite. Then, since there is no model $\mathcal{M}'$ as above, there is a formula ξ_κ in $\mathcal{L}(O_2)$ that stands for "there are κ many elements", for each of $\kappa, \kappa', \kappa''$. Then, $O_2 \models \xi_\kappa \Rightarrow (\neg\xi_{\kappa'}(P_2) \vee \neg\xi_{\kappa''}(P_2^c))$. This formula is exactly such a ϕ' for which ϕ (the O_1-version of ϕ') satisfies $O_1 \not\models \phi$ (because $\mathcal{M}$ is a counter example of ϕ). Contradicting our assumption that in $\mathcal{M}$ every sentence $\phi \in \mathcal{L}_{O_1,O_2}$ is equivalent to ϕ'.

If κ is infinite, then that means that every infinite model of O_2 satisfies $\psi = \neg\xi_{\kappa'}(P_2) \vee \neg\xi_{\kappa'}(P_2^c)$, where ξ'_κ in $\mathcal{L}(O_2)$ stands for "there are κ' many elements such that ..." If all the finite models of O_2 satisfy $\neg\psi$ then this is a contradiction to the fact that FOL cannot express "there are finitely-many elements" (by the Löwenheim-Skolem Theorem (see p.141 in (Enderton, 1972))). Thus, there are finite cardinalities ν, ν', ν'' such that $O_2 \models \xi_{>\nu} \Rightarrow (\neg\xi_{>\nu'}(P_2) \vee \neg\xi_{\nu''}(P_2^c))$. But this is again a formula ϕ' for which ϕ satisfies $O_1 \not\models \phi$ ($\mathcal{M}$ is a counter example to ϕ). This is a contradiction to our assumption that in $\mathcal{M}$ every sentence $\phi \in \mathcal{L}_{O_1,O_2}$ is equivalent to ϕ'.

To generalize over the assumptions we made at the beginning of our proof, we need to first be able to extend the arity of the parameters of P_1, P_2 to n. This can be done using a new sort of elements that are vectors of such arity. Having the goal being stated in that sort and then "translated" to and from it to the different layers gives us the required reduction. To add more links (ab's between other nonlogical symbols), it is simple to consider the links as extending the arity of the single

formula P_1. Finally, to add constant symbols and functions, notice that they are all representable using relations in the language. ■

The following seemingly innocent lemma is central to our theory as it assures that every ab that can not be proved FALSE has a provable reason. It is also particularly difficult to prove because for an arbitrary circumscribed formula it may be the case that ab has no explicit definition coming out of the circumscription.

Lemma A.1.4 *If* $Circ[\widetilde{\mathcal{A}}(O_1, O_2); Ab_{O_1,O_2}; \mathcal{L}(O_2)] \not\models \forall x \neg ab(x)$ *for some* $ab \in Ab_{O_1,O_2}$*, then there is* $\phi \in \mathcal{L}_{O_1,O_2}$ *such that* $O_1 \not\models \phi$ *and* $O_2 \models \phi'$*, where* ϕ' *is the translation of* ϕ *from* O_1 *to* O_2*.*

Furthermore, if we let O_1' *be the result of adding all such formulae* ϕ*'s to the axioms of* O_1*, then* $Circ[\widetilde{\mathcal{A}}(O_1', O_2); Ab_{O_1,O_2}; \mathcal{L}(O_2)] \models \forall x \neg ab(x)$*.*

PROOF Assume that for every $\phi \in \mathcal{L}_{O_1,O_2}$, $O_2 \models \phi' \implies O_1 \models \phi$. Let us take a model $\mathcal{M}$ of the circumscription in which $ab(a)$ is true for some assignment a. We show that there is a model that is preferred over $\mathcal{M}$ (thus reaching a contradiction).

If, for every sentence $\phi \in \mathcal{L}_{O_1,O_2}$, $\mathcal{M} \models \phi$ iff $\mathcal{M} \models \phi'$ (ϕ' translated as above), then Lemma A.1.3 guarantees the existence of a model $\mathcal{N}$ of $\widetilde{\mathcal{A}}(O_1, O_2)$ such that $\mathcal{N} \leq^{Ab_{O_1,O_2};\mathcal{L}(O_2)} \mathcal{M}$. Contradiction.

Take $\Phi = \{\phi \in \mathcal{L}_{O_1,O_2} \mid \mathcal{M} \models \phi\}$. If Φ' (the translation of Φ from O_1 to O_2) is consistent with O_2, then there is a model $\mathcal{M}'$ in which every sentence $\phi \in \mathcal{L}_{O_1,O_2}$ satisfies $\mathcal{M}' \models \phi$ iff $\mathcal{M}' \models \phi'$. Take such a model $\mathcal{M}'$ so that it also has the same universe as $\mathcal{M}$, the same interpretation for $\mathcal{L}(O_1)$ and $\mathcal{M}' \models \forall x.ab(x)$ (this is possible because the interpretations for $\mathcal{L}(O_1)$ and $\mathcal{L}(O_2)$ are *independent* given $\forall x.ab(x)$, and if there is no such model, then we found a sentence ϕ such that $O_1 \not\models \phi$ and $O_2 \models \phi'$ (similar argument to the one given in the proof of Lemma A.1.3)). $\mathcal{M}'$ satisfies the conditions of Lemma A.1.3, and thus there is a model $\mathcal{N}$ that satisfies $\forall x.\neg ab(x)$, keeps the same interpretation for $\mathcal{L}(O_1)$ and has the same universe of elements as $\mathcal{M}$. Contradiction.

If Φ' is inconsistent with $\mathcal{A}(O_2)$, then there is a finite subset of it, $\hat{\Phi}'$, that is inconsistent with $\mathcal{A}(O_2)$ (by compactness of FOL). Let $\phi' = \bigwedge \hat{\Phi}'$. Then, $O_2 \models \neg\phi'$, and $\neg\phi$ is the formula promised by our lemma.

For the second part of the lemma, let O_1' be the result of adding all those ϕ to O_1. By the first part, we get that $Circ[\widetilde{\mathcal{A}}(O_1', O_2); Ab_{O_1',O_2}; \mathcal{L}(O_2)] \models \forall x \neg ab(x)$ because otherwise, there is ϕ that O_1' does not entail (which means O_1 does not entail it) but which O_2 does entail, contradicting the way we built O_1'. ■

The following lemma is a single step in our completeness proof.

Lemma A.1.5 *Let $T = \{O_1, O_2\}$ be a totally ordered OOFOL theory, and φ be a sentence in $\mathcal{L}(O_2)$. If $T \models \varphi$, then $O_2 \models \varphi$, or there is a sentence $\psi \in \mathcal{L}_{O_1,O_2}$ such that $O_1 \models \psi$ and $O_2 \models \psi' \Rightarrow \varphi$.*

The proof of this lemma relies on Craig's interpolation theorem (Theorem 4.2.5), which we repeat here.

Theorem A.1.6 (Craig's Interpolation Theorem (Craig, 1957a)) *If $\alpha \vdash \beta$, then there is a formula γ involving only symbols common to both α and β, such that $\alpha \vdash \gamma$ and $\gamma \vdash \beta$.*

PROOF Assume $T \models \varphi$. Then, $Circ[\widetilde{\mathcal{A}}(O_1, O_2); Ab_{O_1,O_2}; \mathcal{L}(O_2)] \models \varphi$. If $O_2 \models \varphi$ then we are done. Otherwise, there is a model M of O_2 such that $M \models \neg\varphi$.

Assume first that the circumscription entails $\forall x.\neg ab(x)$. We use Craig's Interpolation Theorem. Let $\alpha = \mathcal{A}(O_1)$ and $\beta = (\widetilde{\mathcal{A}}(O_1, O_2) \setminus \mathcal{A}(O_1) \cup \{\forall x.\neg ab(x) \mid ab \in Ab_{O_1,O_2}\}) \Rightarrow \varphi$. Since $\alpha \models \beta$, there is a formula ψ in the language $\mathcal{L}_{O_1,O_2}$ (which is the intersection of the two languages $\mathcal{L}(\alpha), \mathcal{L}(\beta)$), such that $\alpha \models \psi$ and $\psi \models \beta$. By the deduction theorem for FOL, we get that $O_1 \models \psi$ and $\psi' \wedge \mathcal{A}(O_2) \models \varphi$, where ψ' is the translation of ψ from O_1 to O_2.

Now we deal with the case that $\forall x.\neg ab(x)$ is not provable. Take Φ to be the set of all ϕ guaranteed by lemma A.1.4. Add Φ, the to O_1 to create O_1'. Using the first part, we know that there is a sentence ψ as required for O_1', O_2. Thus, $\{\psi\} \cup \mathcal{A}(O_2) \models \varphi$.

Take Ψ to be the set of sentences proved by O_1 in the language $\mathcal{L}_{O_1,O_2}$. $\Psi \cup \Phi' \models \psi$ because $\mathcal{A}(O_1) \cup \Phi' \models \psi$ and, by Craig's Interpolation Theorem, there is α in the language $\mathcal{L}_{O_1,O_2}$ such that $O_1 \models \alpha$ and $\alpha \models \Phi' \Rightarrow \psi$, and necessarily $\Psi \models \alpha$ (the last sentence conclusion was stated for the finite Φ'. For an infinite case, the compactness of FOL reduces our conclusion to the one given above).

Take Ψ' to be the translation of Ψ from O_1 to O_2. Then, $\Psi' \cup \mathcal{A}(O_2) \models \varphi$. Take ψ' to be the finite subset of Ψ' needed for the proof of φ. The corresponding $\psi \subset \Psi$ is the formula needed for our lemma. ■

The following lemma says that there is no inference back (from objects further in sequence to those before them). Let $\mathcal{L}_=$ be the language including no constant/function/relation symbols.

Lemma A.1.7 *Let $T = \{O_i\}_{i \leq m}$. Assume that all the circumscriptions done for T in Definition 2.4.1 have* smooth *preference relations (i.e., every model is either minimal or has a minimal model*

that is preferred over it). If $T \models \varphi$, $\varphi \in \mathcal{L}(O_k)$, *then*

$$T \setminus \{O_j | j \geq k\} \cup \{\psi | \psi \in \mathcal{L}_= \; T \models \psi\} \models \varphi$$

PROOF Assume $T' = T \setminus \{O_j \mid j > i\}$ and $T' \not\models \varphi$. Then, there is a minimal model $\mathcal{M} \models T'$ such that $\mathcal{M} \not\models \varphi$ ($\mathcal{M} \models \neg\varphi$). Since $T \models \varphi$, $\mathcal{M}$ is not minimal according to one of the circumscriptions, with P minimized varying Q.

Since we assume that $\leq^{P;Q}$ is smooth, there is $\mathcal{M}'$ minimal such that $\mathcal{M}' \leq^{P;Q} \mathcal{M}$. But P, Q are not in the language $\mathcal{L}(O_i)$, because this is a circumscription of some $j > i$. Thus, $\mathcal{M}'$ and $\mathcal{M}$ agree on all the symbols of $\mathcal{L}(O_i)$, and $\mathcal{M}' \models \neg\varphi$. Contradiction. ■

The proof of the theorem follows.

PROOF Assume first that $n = m$. We prove the theorem by induction on the number of objects in the theory. For the cases $n = 1, 2$ we have Lemma A.1.5 above. Assume that the theorem is proved for all sizes smaller than n, and we will prove it for n.

Assume $T \models \varphi$. The case of $Circ[\widetilde{\mathcal{A}}(O_{n-1}, O_n); Ab_{O_{n-1},O_n}; \mathcal{L}(O_n)] \models \varphi$, is dealt by Lemma A.1.5. For the other case ($Circ[\widetilde{\mathcal{A}}(O_{n-1}, O_n); Ab_{O_{n-1},O_n}; \mathcal{L}(O_n)] \not\models \varphi$), assume that the last circumscription entails $\forall x.\neg ab(x)$. Then,

$$\begin{array}{l}(\bigwedge_{i<n-1} Circ[\widetilde{\mathcal{A}}(O_i, O_{i+1}); Ab_{O_i,O_{i+1}}; \mathcal{L}(O_{i+1})]) \wedge \\ Circ[\widetilde{\mathcal{A}}(O_{n-1}, O_n); Ab_{O_{n-1},O_n}; \mathcal{L}(O_n)] \models \varphi\end{array}$$

Let $\Phi_{ab} = \{\forall x.\neg ab(x) \mid ab \in Ab_{O_{n-1},O_n}\}$. The above conjunctive formula is equivalent to

$$\begin{array}{l}(\bigwedge_{i<n-1} Circ[\widetilde{\mathcal{A}}(O_i, O_{i+1}); Ab_{O_i,O_{i+1}}; \mathcal{L}(O_{i+1})]) \wedge \\ (\widetilde{\mathcal{A}}(O_{n-1}, O_n) \cup \Phi_{ab}) \models \varphi\end{array}$$

Let $T' = \{O_i\}_{i<n-1} \cup \{O'_{n-1}\}$, where O'_{n-1} is the union of $\widetilde{\mathcal{A}}(O_{n-1}, O_n)$ and Φ_{ab}. By the induction hypothesis, there is the required chain of formulas $\varphi_a, ..., \varphi_{n-2}$ in the system T'. In particular, $T \models \varphi_{n-2}$ and

$$\{\varphi'_{n-2}\} \cup \widetilde{\mathcal{A}}(O_{n-1}, O_n) \cup \Phi_{ab} \models \varphi$$

Using Craig's Interpolation Theorem, there is ψ in $\mathcal{L}_{O_{n-1},O_n}$ such that $\varphi'_{n-2} \wedge \mathcal{A}(O_{n-1}) \models \psi$ and $\psi' \wedge \mathcal{A}(O_n) \models \varphi$. Putting $\varphi_{n-1} = \psi$ gives us the required induction step.

Now we deal with the case that $\forall x.\neg ab(x)$ is not provable for some $ab \in Ab_{O_{n-1},O_n}$. Take Φ to be the set of all ϕ guaranteed by Lemma A.1.4. Add Φ to O_{n-1} to create O'_{n-1}. Using the first part, we know that there is a sequence of sentences as required for the changed system T' (replacing O_{n-1} by O'_{n-1}).

Take φ'_{n-1} to be the sentence that is guaranteed for O'_{n-1}, O_n. Thus, $\{\varphi'_{n-1}\} \cup A(O_2) \models \varphi$. Take Ψ to be the set of sentences proved by O_{n-1} in the language $\mathcal{L}_{O_{n-1},O_n}$. $\Psi \cup \Phi' \models \varphi'_{n-1}$ because $\mathcal{A}(O_{n-1}) \cup \Phi' \models \varphi'_{n-1}$ (by Craig's Interpolation Theorem, there is α in the language $\mathcal{L}_{O_{n-1},O_n}$ such that $O_{n-1} \models \alpha$ and $\alpha \models \Phi' \Rightarrow \varphi'_{n-1}$, and necessarily $\Psi \models \alpha$).

Take Ψ' to be the translation of Ψ from O_{n-1} to O_n. Then, $\Psi' \cup \mathcal{A}(O_n) \models \varphi_n$. Take φ'_{n-1} to be the finite subset of Ψ' needed for the proof of φ. The corresponding $\varphi_{n-1} \subset \Psi$ is the required formula for our induction step.

Lemma A.1.7 reduces the case of $n \neq m$ to the previous case ($n = m$). We only need to notice that $\{\psi \in \mathcal{L}_= \mid T \models \psi\} = \{\psi \in \mathcal{L}_= \mid \exists i\ Obj_i \models \psi\}$. ■

A.2 Proofs for Chapter 3: Object-Oriented Situation Calculus

A.2.1 Proposition 3.4.1: Reiter's Frame Solution Keeps Links Thin

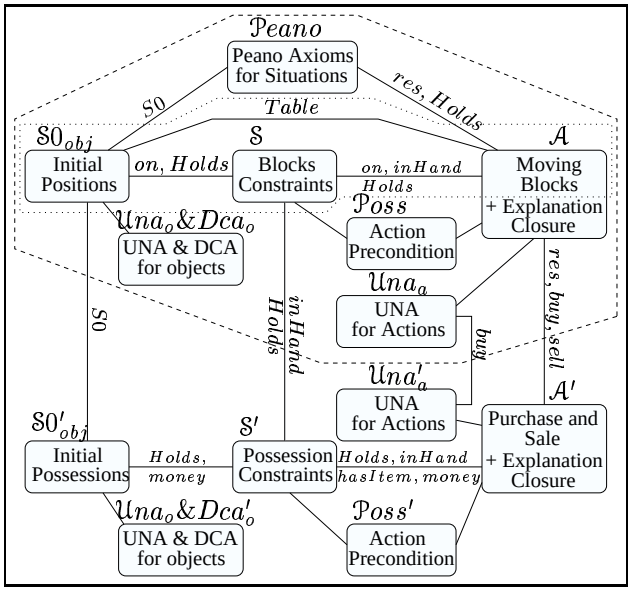

Figure A.1: Connecting two domain theories.

PROOF We use the structure depicted in Figure A.1 to explain the case for two domains. Call the two domains 1 and 2. We need to formulate $\mathcal{P}oss, \mathcal{U}na_a, \mathcal{U}na_o \& Dca_o$ such that they do not use any symbol that is in domain 2 and not in $\{f_i\}_{i\in I}, S0, res, Holds, \{a_j\}_{j\in J}$. This is enough, because $\mathcal{S}o_{obj}, \mathcal{S}, \mathcal{A}$ are in the original domain axiomatization, for which this is trivially true, and $\mathcal{P}eano$ uses only symbols $res, Holds, S0$ and the partial order $\prec$ on situations.

The axioms involving $Poss$ are always in the vocabulary of domain 1, following from our assumption that there are no qualification constraints. Also, we do not need to include DCA for actions, as there are no qualification constraints.

In each of $\mathcal{U}na_a, \mathcal{U}na'_a$ we use the formulation of UNAs given in (McCarthy, 1986b). We define a total order, $<$, on action prototypes (this can be extended to a total order on actions, but we are not concerned with that here). Set an action prototype, A_2 from $\{a_j\}_{j\in J}$, to be the $<$-smallest in domain 2. Set another action, A_1 from domain 1, to be the $<$-largest in domain 1. Adding the axiom $A_1 < A_2$ to Una_a completes the needed specification for UNA.

Finally, UNA and DCA for objects can be formulated to be domain-dependent and also situation dependent (e.g., adding $block(x)$ as a precondition for axioms in domain 1). This concludes part 1, as the arguments above can be easily generalized to more than two domain theories.

For part 2, notice that DCA and UNA for objects and UNA for actions do not need to be re-computed when we link domain 2 to domain 1. When there are more than two domains involved, an increasing order on added domains can be used to guide axiomatizing $<$ on actions so that there are no inconsistencies (inconsistency may arise if we sanction $a < b < c < a$). Notice that if two domains share object symbols, then we must use a scheme that is similar to the one used for actions (sharing objects but not object symbols does not give rise to this problem).

For part 3, notice that all domain theories have the same semantics to $Holds, res, S0$ (the graph of domain theories is connected by our assumption). If two domain theories do not share a fluent, then they do not share an action (we assume that domains originally do not share actions, and they eventually need to share an action only as a result of explanation closure). This means that symbols that should have matching semantics in both domains already have matching semantics by virtue of the graph being connected. Thus, such domain pairs do not need to be directly connected. ■

A.3 Proofs for Chapter 4: Reasoning

A.3.1 Theorem 4.2.4: FORWARD-M-P (MP) is Sound and Complete

First, notice that soundness is immediate because the only rules used in deriving consequences are those used in our chosen consequence-finding procedure (of which rules are sound). In all that follows, we assume $\mathcal{A}$ is finite. The infinite case follows by the compactness of FOL.

Lemma A.3.1 *Let $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$ be a partitioned theory. Let $\varphi \in \mathcal{L}(\mathcal{A}_2)$. If $\mathcal{A} \vdash \varphi$, then $\mathcal{A}_2 \vdash \varphi$ or there is a sentence $\psi \in \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ such that $\mathcal{A}_1 \vdash \psi$ and $\mathcal{A}_2 \vdash \psi \Rightarrow \varphi$.*

Proof of Lemma A.3.1. We use Craig's interpolation theorem (Theorem 4.2.5), taking $\alpha = \mathcal{A}_1$ and $\beta = \mathcal{A}_2 \Rightarrow \varphi$. Since $\alpha \vdash \beta$ (by the deduction theorem for FOL), there is a formula $\psi \in \mathcal{L}(\alpha) \cap \mathcal{L}(\beta)$ such that $\alpha \vdash \psi$ and $\psi \vdash \beta$. By the deduction theorem for FOL, we get that $\mathcal{A}_1 \vdash \psi$ and $\psi \wedge \mathcal{A}_2 \vdash \varphi$. Since $\psi \in \mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2)$ by the way we constructed α, β, we are done. ■

Proof of Theorem 4.2.4. we prove the theorem by induction on the number of partitions in the logical theory. For $|V| = 1$ (a single partition), $\mathcal{A} = \mathcal{A}_1$ and the proof is immediate, as $\mathcal{R}_1$ is refutation complete (every reasoner that is complete for $\mathcal{L}$-generation for some $\mathcal{L}$ is refutation complete) and we added $\neg Q$ to $\mathcal{A}_1$. Assume that we proved the theorem for $|V| \leq n-1$ and we prove the theorem for $|V| = n$.

In G, k has c neighbors, $i_1, ..., i_c$. $(k, i_1) \in E$ separates two parts of the tree G: G_1 (includes i_1) and G_2 (includes k). Let $\mathcal{B}_1, \mathcal{B}_2$ be the subtheories of $\mathcal{A}$ that correspond to G_1, G_2, respectively.

Notice that $Q \in \mathcal{L}(\mathcal{B}_2)$. By Lemma A.3.1, either $\mathcal{B}_2 \vdash Q$ or there is $\psi \in \mathcal{L}(\mathcal{B}_1) \cap \mathcal{L}(\mathcal{B}_2)$ such that $\mathcal{B}_1 \vdash \psi$ and $\mathcal{B}_2 \vdash \psi \Rightarrow Q$. If $\mathcal{B}_2 \vdash Q$, then we are done, by the induction hypothesis applied to the partitioning $\{\mathcal{A}_i \mid i \in V_2\}$ (V_2 includes the vertices of G_2) and G_2 (notice that $\prec'$ used for G_2, Q agrees with $\prec$ used for G).

Otherwise, let ψ be a sentence as above. $\bigcup_{(i_1,j)\in E, j\neq k} l(i_1, j) \supseteq L(\mathcal{B}_2 \cup \mathcal{A}_{i_1}) \cap L(\mathcal{B}_1 \setminus \mathcal{A}_{i_1})$ because the set of edges (i_1, j) separates two subgraphs corresponding to the theories $\mathcal{B}_1 \setminus \mathcal{A}_{i_1}$ and $\mathcal{B}_2 \cup \mathcal{A}_{i_1}$, and G is properly labeled for our partitioning. Thus, since $\psi \in \mathcal{L}(\mathcal{B}_1)$ we get that $\psi \in \mathcal{L}(\mathcal{A}_{i_1} \cup \bigcup_{(i_1,j)\in E, j\neq k} l(i_1, j))$. By the induction hypothesis for $G_1, \mathcal{B}_1$, at some point there will be a set of consequences of $\mathcal{A}_{i_1}$ that entail ψ (after some formulae were sent to it from the other partitions in G_1, B_1). Since G is properly labeled for $\mathcal{A}, G$, $\psi \in \mathcal{L}(l(k, i_1))$. Since $\mathcal{R}_i$ is complete for $\mathcal{L}(l(k, i_1))$-generation, at some point the set C of consequences of $\mathcal{A}_{i_1}$ that are in $\mathcal{L}(l(k, i))$ will entail ψ.

At this point, our algorithm will have sent the sentences in C to $\mathcal{A}_k$ because $C \subset \mathcal{L}(l(k, i_1))$. Since $B_2 \vdash \psi \Rightarrow Q$, then $B_2 \vdash C \Rightarrow Q$. By the induction hypothesis applied to G_2, B_2 ($C \Rightarrow Q \in \mathcal{L}(\mathcal{A}_k \cup_{(k,i)\in E} l(k, i))$) at some point $C \Rightarrow Q$ will be entailed from consequences of $\mathcal{A}_k$ (after some message passing). Thus, at some point we will find consequences of $\mathcal{A}_k$ that entail Q. This completes the induction step. ■

A.3.2 Corollary 4.2.8: MP and Intersection Graphs that are Trees

Proof of Corollary 4.2.8. All we have to prove is that the intersection graph G has a proper labeling. But if G is the intersection graph of the partitioning $\{\mathcal{A}_i\}_{i\leq n}$ then $l(i, j) = L(\mathcal{A}_i) \cap L(\mathcal{A}_j)$. If for $(i, j) \in E$ $L(l(i, j)) \not\supseteq L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$, with $\mathcal{B}_1, \mathcal{B}_2$ the theories on the two sides of (i, j) in the tree G, then there are $\mathcal{A}_x, \mathcal{A}_y$ in $\mathcal{B}_1, \mathcal{B}_2$, respectively, such that $(x, y) \in E$ and $x \neq i$ or $y \neq j$. Since G is connected (it is a single tree), this means there is a cycle in G, contradicting G being a tree. Thus $L(l(i, j)) \supseteq L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$ and G has a proper labeling. The proof follows from Theorem 4.2.4. ■

A.3.3 Corollary 4.2.9: MP with BREAK-CYCLES is Sound and Complete

Soundness is immediate, using the same argument as for Theorem 4.2.4. For completeness, first notice that the graph output by BREAK-CYCLES is always a tree, because BREAK-CYCLES will not terminate if there is still a cycle in G. Now, we need the following lemma.

Lemma A.3.2 *Let $G' = (V, E', l')$ be a tree resulting from applying BREAK-CYCLES to $G = (V, E, l)$ and $\{\mathcal{A}_i\}_{i\leq n}$. Then G' has a proper labeling for this partitioning.*

Proof of Lemma A.3.2. Assume there is a symbol p in $L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$ that is not in $l(i, j)$, and let $\mathcal{A}_x, \mathcal{A}_y$ be partitions on the two sides of (i, j) that include sentences with the symbol p. We will prove that throughout the run of the BREAK-CYCLES algorithm there is always a path in G' (we start with $G' = G$) between $\mathcal{A}_x, \mathcal{A}_y$ that has p showing on all the edge labels. Call such a path a *good path*.

Obviously we have a good path in G, because we have $(x, y) \in E$ and $p \in l(x, y)$ (because G is the intersection graph of $\mathcal{A}_1, ..., \mathcal{A}_n$). Let us stop the algorithm at the first step in which G' does not have a good path (assuming there is no such path, or otherwise we are done). In the last step we must have removed an arc (a, b) (which was on a good path) to cause G' to not have a good path.

Since $p \in l(a,b)$ and (a,b) is in a cycle $\langle (b,a_1),(a_1,a_2),...,(a_c,a),(a,b)\rangle$ (this is the only reason we removed (a,b)), we added $l(a,b)$ to the labels of the rest of this cycle. In particular, now the labels of $(b,a_1),(a_1,a_2),...,(a_c,a)$ include p. Replacing (a,b) in the previous good path by this sequence, we find a path in the new G' that satisfies our required property. This is a contradiction to having assumed that there is no such path at this step. Thus, there is no such p as mentioned above and $L(l(i,j)) \supseteq L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$. ■

Proof of Corollary 4.2.9. The proof follows immediately from Theorem 4.2.4 and Lemma A.3.2. ■

A.3.4 Theorem 4.2.10: BACKWARD-M-P (BMP) is Sound and Complete

Proof of Theorem 4.2.10. First, notice that procedure CHAINIFY returns a graph that is a chain with k in one of its ends because it uses a total order for the nodes of G with k the first element. The chain returned is properly labeled because steps 5 and 6 implement the proper labeling condition.

Next, we reduce steps done in BMP to corresponding ones done in MP and we get the soundness and completeness from Theorem 4.2.4. For a prover i to have a goal Q_i means that it needs to prove that the theory $\mathcal{A}_i \cup \{\neg Q_i\}$ is inconsistent. φ is a *subgoal* in a subgoal-disjunctive system if $\{\varphi\} \cup \mathcal{A}_i \vdash Q_i$. For a series of subgoals $\psi_1,...,\psi_r$ in partition $\mathcal{A}_i$, $\{\psi_1 \vee ... \vee \psi_r\} \cup \mathcal{A}_i \vdash Q_i$. Also, if $\mathcal{A}_j$ is the partition to whom $\mathcal{A}_i$ sends its subgoals, then Q_j, the goal of partition $\mathcal{A}_j$, is $\psi_1 \vee ... \vee \psi_r$ at this point in time.

Let φ be a subgoal of $\mathcal{A}_i$. This means that $\mathcal{A}_i \cup \{\neg Q_i\} \vdash \neg\varphi$. Thus, our BMP algorithm readily reduces to MP, as $\neg\varphi$ would be sent from $\mathcal{A}_i$ to $\mathcal{A}_j$ in MP while φ would be disjoin-ed with the goal of $\mathcal{A}_j$ in BMP, and both need to prove inconsistency of $\mathcal{A}_j \cup \{\neg Q_j \wedge \neg\varphi\}$, when Q_j is the goal of $\mathcal{A}_j$ before the arrival of φ. From the soundness and completeness of MP, we get soundness and completeness for BMP. ■

A.3.5 Theorem 4.3.2: RESOLUTION-M-P (RES-MP) is Sound and Complete

Theorem A.3.3 ((Lee, 1967)) *For every non-tautologous clause D following from a given clause set $\mathcal{A}$, a clause C is derivable by the resolution rule such that D is obtained from C by instantiation and addition of further literals (i.e., $C \subset$-subsumes D).*

Proof of Theorem 4.3.2. Soundness and completeness of the algorithm follow from that of FMP, if we show that RES-SEND (Implementation 4) adds enough sentences (implying completeness) to

$\mathcal{A}_i$ that are implied by φ (thus sound) in the restricted language $\mathcal{L}(l(i,j))$.

If we add all sentences φ that are submitted to RES-SEND to $\mathcal{A}_i$ without any translation, then our soundness and completeness result for FMP applies (this is the case where we add all the constant and function symbols to all $l(i,j)$).

We use Theorem 4.3.1 to prove that we add enough sentences to $\mathcal{A}_i$. Let φ_2 be a quantified formula that is the result of applying algorithm *U* to φ. Then, φ_2 results from a clause C generated in step 4 of algorithm *U* (respectively, Step 3 in RES-SEND). In algorithm *U*, for each variable x, the markings "$x \leftarrow \alpha_i$" in C are converted to a new variable that is existentially quantified immediately to the right of the quantification of the variables $y_1, ..., y_r$. φ_2 is a result of ordering the quantifiers in a consistent manner to this rule (this process is done in steps 5–6 of algorithm *U*).

Step 4 of RES-SEND performs the same kind of replacement that algorithm *U* performs, but uses new function symbols instead of new quantified variables. Since each new quantified variable in φ_2 is to the right of the variables on which it depends, and our new function uses exactly those variables as arguments, then Step 4 generates a clause C' from C that entails φ_2. Thus, the clauses added to $\mathcal{A}_i$ by RES-SEND entail all the clauses generated by unskolemizing φ using *U*. From Theorem 4.3.1, these clauses entail all the sentences in $\mathcal{L}(l(i,j))$ that are implied by φ.

To see that the result is still sound, notice that the set of clauses that we add to $\mathcal{A}_i$ has the same consequences as φ in $\mathcal{L}(l(i,j))$ (i.e., if we add those clauses to $\mathcal{A}_j$ we get a conservative extension of $\mathcal{A}_j$). ■

A.3.6 Lemma 4.3.6: Resolving on P is $\mathcal{L}_P$-Generation Complete

This proof uses the notion of *semantic trees*. For an exposition the reader is referred to (Chang and Lee, 1973). We bring only the basic definitions needed for the proof here.

The *atom set* of a set of formulae, $\mathcal{A}$ is the set of all atoms in $\mathcal{A}$.

Definition A.3.4 *Given a set S of clauses, let A be the atom set of S. A* semantic tree *for S is a (downward) tree T, where each link is attached with a finite set of atoms or negations of atoms from A in such a way that*

1. *For each node N, there are only finitely many immediate links $L_1, ..., L_n$ from N. Let Q_i be the conjunction of all the literals in the set attached to $L_i, i = 1, ..., n$. Then, $Q_1 \vee ... \vee Q_n$ is a valid propositional formula.*

2. *For each node N, let $I(N)$ be the union of all the sets attached to the links of the branch of T down to and including N. Then, $I(N)$ does not contain any complementary pair.*

Definition A.3.5 *Let $A = \{A_1, A_2, ...\}$ be the atom set of a set S of clauses. A semantic tree for S is said to be* complete *if and only if for every leaf node N of the semantic tree, $I(N)$ contains either A_i or $\neg A_i$, for $i = 1, 2, ...$*

Roughly speaking, a complete semantic tree corresponds to all possible *Herbrand interpretations*.

Definition A.3.6 *A node N is a* failure node *if $I(N)$ falsifies some ground instance of a clause in S, but $I(N')$ does not falsify any ground instance of a clause in S for every ancestor node N' of N. A semantic tree is said to be* closed *iff every branch of T terminates at a failure node. A node N of a closed semantic tree is called an* inference node *if all the immediate descendant nodes of N are failure nodes.*

The following is a version of Herbrand's Theorem (Herbrand, 1930) stated using semantic trees.

Theorem A.3.7 (Herbrand's Theorem (ver.1)) *A set S of clauses is unsatisfiable if and only if corresponding to every complete semantic tree of S, there is a finite closed semantic tree.*

We will also make use of the classic mention of Herbrand's Theorem.

Theorem A.3.8 (Herbrand's Theorem (ver.2)) *A set S of clauses is unsatisfiable if and only if there is a finite unsatisfiable set S' of ground instances of clauses of S.*

Finally, we also make use of the following lifting lemma (see (Chang and Lee, 1973) p.84).

Lemma A.3.9 (Lifting Lemma) *If C_1', C_2' are instances of C_1, C_2, respectively, and if C' is a resolvent of C_1', C_2', then there is a resolvent C of C_1, C_2 such that C' is an instance of C.*

Proof of Lemma 4.3.6. Take a formula $\varphi \in \mathcal{L}_P$ such that $\mathcal{A} \models \varphi$. We show that for $\mathcal{A}' = C_{\mathcal{R}}(\mathcal{A})$, $\mathcal{A}' \models \varphi$.

$\mathcal{A} \cup \{\neg\varphi\}$ is unsatisfiable, by our assumption. Let $\mathcal{A}_0$ be a finite unsatisfiable set of ground instances of $\mathcal{A} \cup \{\neg\varphi\}$, as guaranteed by Theorem A.3.8. Let T be a complete semantic tree of $\mathcal{A}_0$ such that the instances of P are the lowest steps in the tree. Since $\mathcal{A}_0$ is unsatisfiable, then by Theorem A.3.7 there is a finite closed semantic tree, T, corresponding to it. This tree is a semantic closed tree for $\mathcal{A} \cup \{\neg\varphi\}$ as well.

We show that this tree corresponds to a resolution deduction of $\{\}$ from $\mathcal{A} \cup \{\neg\varphi\}$ and modify the tree to achieve a resolution of $\{\}$ from $\mathcal{A}' \cup \{\neg\varphi\}$. First, notice that T has at least one inference node (a node under which both nodes are failure nodes), or otherwise we get an infinite branch in it.

Every inference node in T corresponds to a resolution step of two clauses that are falsified by this node's children. Let N be an inference node in T. Let N_1, N_2 be the failure nodes immediately below N. Since N_1, N_2 are failure nodes, there must be two ground instances C_1', C_2' of clauses C_1, C_2, respectively, such that C_1', C_2' are false in $I(N_1), I(N_2)$, respectively, but both C_1', C_2' are not falsified by $I(N)$. Let m be the ground atom that appears positively in $I(N_1)$ and negatively in $I(N_2)$. Then, without loss of generality, C_1' must contain $\neg m$, and C_2' must contain m. Resolving C_1', C_2' upon m we obtain the resolvent

$$C' = (C_1' \setminus \{\neg m\}) \cup (C_2' \setminus \{m\}).$$

C' must be false in $I(N)$ since both $(C_1' \setminus \{\neg m\}), (C_2' \setminus \{m\})$ are false in $I(N)$. By the lifting lemma there is a resolvent, C, of C_1, C_2 such that C' is a ground instance of C. Let T' be the closed semantic tree for $\mathcal{A} \cup \{\neg\varphi\} \cup \{C\}$ obtained from T by deleting any node or link that is below the first node where the resolvent C' is falsified. Clearly, $T' \subsetneq T$. If we continue this process iteratively, we eventually collapse the tree to a singleton, which is the empty set. This process then corresponds to a resolution proof of $\{\}$.

Coming back to our original semantic closed tree, T, we continue the process above iteratively until we remove from T all the inference nodes that resolve on P (we make sure to remove only those that resolve on P). This process results in a new closed semantic tree, T^*, for the theory including some of the resolvents of $\mathcal{A} \cup \{\neg\varphi\}$ on P together with $\mathcal{A}$. Since φ does not include P, φ did not participate in any of the resolutions. Also, T^* is closed before we get to nodes that are falsified by P. Thus, deleting the set of clauses that include a literal with P, we still have an inconsistent set of clauses. The new set of clauses is included in $\mathcal{A}' \cup \{\neg\varphi\}$. Thus, T^* corresponds to a proof of $\{\}$ from $\mathcal{A}' \cup \{\neg\varphi\}$, which concludes our proof. ■

A.3.7 Theorem 4.3.7: MP Can Simulate Variable Ordering (Most Times)

Proof of Theorem 4.3.7. 1. Let $\{p_i\}_{i \leq n}$ be the set of propositional symbols of $\mathcal{A}$, enumerated by the order $\leq_A$. For all $i \leq n$ let $\mathcal{A}_i = \{\varphi \in \mathcal{A} \mid \textit{highest_literal}(\varphi) = p_i\}$. Let the graph G have vertices $V = \{1, ..., n\}$, edges $E = \{(i, i-1) \mid i \leq n\}$ and labels $l(i, i-1) = \{p_1, ..., p_{i-1}\}$.

For partition $\mathcal{A}_i$ we take the reasoner $\mathcal{R}_{p_i}$ that resolves sentences only on p_i. By Lemma 4.3.6

we know that $\mathcal{R}_{p_i}$ is $\mathcal{L}_{p_i}$-generation complete. Also, for each $i \leq n$, $\mathcal{L}_{p_i} = \mathcal{L}(l(i, i-1))$. This can be seen by induction on i. When $i = n$, $\mathcal{L}(\mathcal{A}_i)$ includes propositions in $\{p_1, ..., p_i\}$, and $\mathcal{A}_i$ does not change as it never receives any messages. For $i < n$, $\mathcal{A}_i$ initially includes propositions in $\{p_1, ..., p_i\}$, and the messages $\mathcal{A}_i$ receives from $\mathcal{A}_{i+1}$ are also in this language, by induction. This proves that $\mathcal{L}_{p_i} = \mathcal{L}(l(i, i-1))$ and the set of reasoners $\mathcal{R}_{p_i}$ is generation-set complete for $\mathcal{A}, G$.

To see that MP does not perform more resolutions than directional resolution of $\mathcal{A}$ with order $\leq_A$, we only need to notice that every partition corresponds to a bucket in the directional resolution algorithm. Thus, each partition performs exactly those resolutions done by the corresponding bucket in the directional resolution algorithm.

2. Let $\{P_i\}_{i \leq n}$ be the set of predicate symbols of $\mathcal{A}$, enumerated by the order $\leq_A$. For all $i \leq n$ let $\mathcal{A}_i = \{\varphi \in \mathcal{A} \mid \textit{highest_predicate}(\varphi) = P_i\}$. Let the graph G have vertices $V = \{1, ..., n\}$, edges $E = \{(i, i-1) \mid i \leq n\}$ and labels $l(i, i-1) = \{P_1, ..., P_{i-1}, f_1, ..., f_s, c_1, ..., c_t\}$. ($f_1, ..., f_s$ are the function symbols in $\mathcal{L}(\mathcal{A})$ and $c_1, ..., c_t$ are the constant symbols in $\mathcal{L}(\mathcal{A})$).

For partition $\mathcal{A}_i$ we take the reasoner $\mathcal{R}_{P_i}$ that resolves sentences only on P_i. By Lemma 4.3.6 we know that $\mathcal{R}_{P_i}$ is $\mathcal{L}_{P_i}$-generation complete. Also, for each $i \leq n$, $\mathcal{L}_{P_i} = \mathcal{L}(l(i, i-1))$. This can be seen by induction on i. When $i = n$, $\mathcal{L}(\mathcal{A}_i)$ includes predicates in $\{P_1, ..., P_i\}$, and $\mathcal{A}_i$ does not change as it never receives any messages. For $i < n$, $\mathcal{A}_i$ initially includes propositions in $\{P_1, ..., P_i\}$, and the messages $\mathcal{A}_i$ receives from $\mathcal{A}_{i+1}$ are also in this language, by induction. This proves that $\mathcal{L}_{P_i} = \mathcal{L}(l(i, i-1))$ and the set of reasoners $\mathcal{R}_{P_i}$ is generation-set complete for $\mathcal{A}, G$.

To see that MP does not perform more resolutions than A-ordered resolution of $\mathcal{A}$ with order $\leq_A$, we only need to notice that every partition resolves only on the top predicate in each clause. This is the case because in every clause in partition $\mathcal{A}_i$ the highest predicate possible is P_i, and the reasoner resolves only on instances of P_i. Thus, each partition performs only resolutions that are allowed by A-ordered resolution.

3. Let $\{\{P^i_j\}_{j \leq l_i}\}_{i \leq n}$ be the set of predicate symbols of $\mathcal{A}$, aggregated to sets of predicates according to the index I (i.e., $I(P^i_j) = i$). For all $i \leq n$ let $\mathcal{A}_i = \{\varphi \in \mathcal{A} \mid I(\textit{highest_predicate}(\varphi)) = i\}$. Let the graph G have vertices $V = \{1, ..., n\}$, edges $E = \{(i, i-1) \mid i \leq n\}$ and labels $l(i, i-1) = \{P^1_1, ..., P^{i-1}_{l_{i-1}}, f_1, ..., f_s, c_1, ..., c_t\}$. ($f_1, ..., f_s$ are the function symbols in $\mathcal{L}(\mathcal{A})$ and $c_1, ..., c_t$ are the constant symbols in $\mathcal{L}(\mathcal{A})$).

For partition $\mathcal{A}_i$ we take the reasoner $\mathcal{R}_{P_i}$ that resolves sentences only on $P^i_1, ..., P^i_{l_i}$. Let $\mathbb{P}^i$ stand for $P^i_1, ..., P^i_{l_i}$. By Lemma 4.3.6 (applied recursively to the set of predicates $\mathbb{P}^i$), we know that $\mathcal{R}_{\mathbb{P}^i}$ is $\mathcal{L}_{\mathbb{P}^i}$-generation complete. Also, for each $i \leq n$, $\mathcal{L}_{\mathbb{P}^i} = \mathcal{L}(l(i, i-1))$. This can be seen by induction on i. When $i = n$, $\mathcal{L}(\mathcal{A}_i)$ includes predicates in $\{\mathbb{P}^1, ..., \mathbb{P}^i\}$, and $\mathcal{A}_i$ does not change

as it never receives any messages. For $i < n$, $\mathcal{A}_i$ initially includes propositions in $\{\mathbb{P}^1, ..., \mathbb{P}^i\}$, and the messages $\mathcal{A}_i$ receives from $\mathcal{A}_{i+1}$ are also in this language, by induction. This proves that $\mathcal{L}_{\mathbb{P}^i} = \mathcal{L}(l(i, i-1))$ and the set of reasoners $\mathcal{R}_{\mathbb{P}^i}$ is generation-set complete for $\mathcal{A}, G$.

To see that MP does not perform more resolutions than lock resolution of $\mathcal{A}$ with index I, we only need to notice that every partition resolves only on predicates with highest index in each clause. This is the case because in every clause in partition $\mathcal{A}_i$ the highest predicates possible are in $\mathbb{P}^i$, and the reasoner resolves only on instances of $\mathbb{P}^i$. Thus, each partition performs only resolutions that are allowed by lock resolution. ■

A.3.8 Theorem 4.3.8: Variable Ordering Can Simulate MP (Propositional, Sometimes)

Proof of Theorem 4.3.8. 1. Let $\{\mathcal{A}_i\}_{i \leq n}$ be a partitioning of the theory $\mathcal{A}$, let $G = (V, E, l)$ be a graph and let $k \leq n$. Let $\prec$ be the partial order derived in step 1 of MP. Let $\tilde{L}(i) = \bigcup_{(i,j) \in E, i \prec j} l(i, j)$ (this is the vocabulary mentioned in the partition itself together with the vocabulary mentioned on the links to that partition).

For every $p \in L(\mathcal{A})$, let $P(p)$ be the $\prec$-smallest partition, i, such that $p \in \tilde{L}(i)$. For every $p \in L(\mathcal{A})$ there is exactly one such partition because otherwise there are at least two that are not connected via another node, contradicting the fact that G is properly labeled (proper labeling guarantees that if a symbol appears in two partitions then it appears on the edges on the path between those partitions).

Define $<_0$, an order between symbols, by $p <_0 q$ iff $P(p) \prec P(q)$. Define $<_A$ to be a total order that agrees with $<_0$. $<_A$ is clearly a partial order, as it obeys reflexivity, transitivity and is strict (thus asymmetric). We show that $<_A$ is an order as required. Let $\{p_i\}_{i \leq m}$ be an enumeration of the propositional symbols of $\mathcal{A}$ according to $<_A$. Notice that if directional resolution with $<_A$ resolved two clauses upon p_i, then p_i is the highest symbol in both clauses.

Let C_1, C_2 be two clauses resolved on p in the directional resolution using the order $<_A$. Let i_1, i_2 be such that C_1 in partition i_1 and C_2 in partition i_2.

Assume that MP cannot resolve the two clauses. That means that they are in different partitions and that at least one of the clauses, say C_2, should be sent to another partition, $\mathcal{A}_j$, with $j \prec i_2$, in order for the two clauses to be resolved (j is the $\prec$-largest partition such that $j \prec i_1$ and $j \prec i_2$).

Since C_2 was not sent to partition j, its vocabulary is not in the label of one of the edges on the path between i_2 and p in G (by the definition of MP, we would have sent that clause if it was

expressed by the labels on each of the edges on that path).

Let q be a symbol that appears in C_2 but is not on one of the edges in the path between partitions i_2 and j. p is the $<_A$-largest symbol for both C_1 and C_2, meaning that p appears on the path between i_1, i_2 (thus it also appears on the path between i_2, j, which is a subset of the path between i_1, i_2) and that $q <_A p$ (because q does not appear on that path (thus, it is not p) and p is the $<_A$-largest in C_2).

Since q is not in $\tilde{L}(j)$ (otherwise it would be on the path between i_2, j) and $P(p) \preceq j$ (otherwise the path between j, i_2 would not include p) $P(p) \preceq j \prec P(q)$ implying that $p <_A q$. This contradicts the previous observation that $q <_A p$. Thus, q appears on the path between i_2, j and C_2 would have to be sent to partition j. The same holds for C_1. Thus, MP will resolve the two clauses.

2. The proof for A-ordering is identical to the one above, except that we treat the predicate symbols instead of the propositional symbols.

3. Let $\{\mathcal{A}_i\}_{i \leq n}$, $G = (V, E, l)$, k, $\prec$ and $\tilde{L}(i)$ be as for the proof of part 1.

For every predicate $p \in L(\mathcal{A})$, let $P(p)$ be the $\prec$-smallest partition, i, such that $p \in \tilde{L}(i)$. For every $p \in L(\mathcal{A})$ there is exactly one such partition because otherwise there are at least two that are not connected via another node, contradicting the fact that G is properly labeled (proper labeling guarantees that if a symbol appears in two partitions then it appears on the edges on the path between those partitions). For every atom a, $P(a) = P(p)$, if p is the predicate symbol of a.

Define $>_0$, a relation between atoms, by $a_1 >_0 a_2$ iff $P(a_1) \prec P(a_2)$. Define $>_A$ to be a total order that agrees with $>_0$. $>_A$ is clearly a partial order, as it obeys reflexivity, transitivity and is strict (thus asymmetric). Let $\{p_i\}_{i \leq m}$ be an enumeration of the predicate symbols of $\mathcal{A}$ according to $>_A$. Define I to be an index of the literals in $\mathcal{A}$ that agrees with this enumeration and has the same index for an atom and its negation.

We show that I is an index as required. Notice that if lock resolution with I resolved two clauses upon p_i, then p_i is the lowest literal in both clauses.

Let C_1, C_2 be two clauses that lock resolution can resolve together on predicate p. Let i_1, i_2 be such that C_1 in partition i_1 and C_2 in partition i_2.

Assume that MP cannot resolve the two clauses. That means that they are in different partitions and that at least one of the clauses, say C_2, should be sent to another partition, $\mathcal{A}_j$, with $j \prec i_2$, in order for the two clauses to be resolved (j is the $\prec$-largest partition such that $j \prec i_1$ and $j \prec i_2$).

Since C_2 was not sent to partition j, its vocabulary is not in the label of one of the edges on the path between i_2 and p in G (by the definition of MP, we would have sent that clause if it was expressed by the labels on each of the edges on that path).

Let q be a predicate symbol that appears in C_2 but is not on one of the edges in the path between partitions i_2 and j. p is the $>_A$-smallest symbol for both C_1 and C_2, meaning that p appears on the path between i_1, i_2 (thus it also appears on the path between i_2, j, which is a subset of the path between i_1, i_2) and that $q >_A p$ (because q does not appear on that path (thus, it is not p) and p is the $>_A$-largest in C_2).

Since q is not in $\tilde{L}(j)$ (otherwise it would be on the path between i_2, j) and $P(p) \preceq j$ (otherwise the path between j, i_2 would not include p) $P(p) \preceq j \prec P(q)$ implying that $p >_A q$. This contradicts the previous observation that $q >_A p$. Thus, q appears on the path between i_2, j and C_2 would have to be sent to partition j. The same holds for C_1. Thus, MP will resolve the two clauses. ■

A.3.9 Theorem 4.4.1: LINEAR-PART-SAT is Sound and Complete

Proof of Theorem 4.4.1. For each partition $\mathcal{A}_i, i \leq n$, lines 1 – 3 perform the equivalent of finding all the models of $\mathcal{A}_i$ and storing their truth assignments to the symbols of $\mathcal{L}(L(i))$ in T_i. ($L(i)$ specifies the columns, thus each row corresponds to a truth assignment.) This is equivalent to finding the implicates of the theory $\mathcal{A}_i$ in the sublanguage $\mathcal{L}(L(i))$. Thus, if A_i is the DNF of the set of implicates $(\alpha_1(p_{j_1}, ..., p_{j_{l_i}}) \vee ... \vee \alpha_{a_i}(p_{j_1}, ..., p_{j_{l_i}}))$, then T_i initially includes the set of models of A_i in the sublanguage $\mathcal{L}(L(i))$, namely, $[\![A_i]\!]_{\mathcal{L}(L(i))}$.

The *natural join* operation ($\bowtie$) then creates all the consistent combinations of models from $[\![A_i]\!]_{\mathcal{L}(L(i))}$ and $[\![A_j]\!]_{\mathcal{L}(L(j))}$. This set of consistent combinations is the set of models of $A_i \cup A_j$. Thus, $T_i \bowtie T_j \equiv [\![(A_i \cup A_j)]\!]_{\mathcal{L}(L(i) \cup L(j))}$.

Finally, the *projection* operation restricts the models to the sublanguage $\mathcal{L}(L(i))$, getting rid of duplicates in the sublanguage. This is equivalent to finding all the implicates of $A_i \wedge A_j$ in the sublanguage $\mathcal{L}(L(i))$. Thus, $\pi_{L(i)}(T_i \bowtie T_j) \equiv [\![\{\varphi \in \mathcal{L}(L(i)) \mid A_i \cup A_j \models \varphi\}]\!]_{\mathcal{L}(L(i))}$.

To see that the algorithm is sound and complete, notice that the it does the analogous operations to our forward message-passing algorithm MP (Figure 4.3). We break the cycles in G_0 (creating G) and perform forward reasoning as in MP, using the set of implicates instead of online reasoning in each partition: instruction 2b in MP is our projection ("$\mathcal{A}_i \models \varphi$ and $\varphi \in \mathcal{L}(l(i,j))$") and then join ("add φ to the set of axioms of $\mathcal{A}_j$"). Since $T_i \bowtie T_j \equiv [\![(A_i \cup A_j)]\!]_{\mathcal{L}(L(i) \cup L(j))}$, joining corresponds to sending all the messages together. Since $\pi_{L(i)}(T_i \bowtie T_j) \equiv [\![\{\varphi \in \mathcal{L}(L(i)) \mid A_i \cup A_j \models \varphi\}]\!]_{\mathcal{L}(L(i))}$, projection corresponds to sending only those sentences that are allowed by the labels.

By Corollary 4.2.9, LINEAR-PART-SAT is sound and complete for satisfiability of $\mathcal{A}$. ■

A.3.10 Lemma 4.4.2: Time Complexity of LINEAR-PART-SAT

Proof of Lemma 4.4.2. Let $\mathcal{A}$ be a partitioned propositional theory with n partitions. Let m be the total number of propositional symbols in $\mathcal{A}$, $L(i)$ the set of propositional symbols calculated in step 2 of LINEAR-PART-SAT, and m_i the number of propositional symbols mentioned in $\mathcal{A}_i \setminus L(i)$ ($i \leq n$). Let us examine procedure LINEAR-PART-SAT (Figure 4.14) step by step, computing the time needed for computation.

Computing the intersection graph takes $O(a * k^2)$ time, where k is the number of propositional symbol in each axiom (for 3SAT, that is 3), because we check and add k^2 edges to G_0 for each axiom.

BREAK-CYCLES' loop starts by finding a minimal-length cycle, which takes time $O(n)$ (BFS traversal of n vertices). Finding the optimal a in line 2 takes time $O((c * m) * c)$, where c is the length of the cycle found (union of two labels takes at most $O(m)$ time). Finally, since a tree always satisfies $|E| = |V| - 1$, breaking all the cycles will require us to remove $|E| - (|V| - 1)$ edges. Thus, the loop will run $|E| - (|V| - 1)$ times (assuming the graph G_0 is connected). An upper bound on this algorithm's performance is then $O(n^2 * (n^2 * m)) = O(n^4 * m)$ (because $c \leq n$ and and $|E| \leq |V|^2 = n^2$).

Step 2 of LINEAR-PART-SAT takes time $O(n * m)$, since there are a total of $n - 1$ edges in the graph G (G is a tree with n vertices) and every label is of length at most m.

Checking the truth assignments in step 3 takes time $\sum_{i=1}^{n} 2^{|L(i)|}$ per satisfiability check of $\mathcal{A}_i \cup A$, because there are $2^{|L(i)|}$ truth assignments for each $i \leq n$. Since $\mathcal{A}_i \cup A$ has only m_i free propositional variables, (A is an assignment of truth values to $|L(i)|$ variables), $\mathcal{A}_i \cup A$ is reducible (in time $O(|A|)$) to a theory of smaller size with only m_i propositional variables. If the time needed for a satisfiability check of a theory with m variables is $O(f_{SAT}(m))$, then the time for step 3 is

$$O(\sum_{i=1}^{n}(2^{|L(i)|} * f_{SAT}(m_i)))$$

Finding the relation $\prec$ takes $O(n)$ as it is easily generated by a BFS through the tree.

Instruction 5 performs a *projection* and *join*, which takes time $O(2^{|L(i)|})$ (the maximal size of the table). Since the number of iterations over $i \leq n$ and j being a child of i is $n - 1$ (there are only $n - 1$ edges), we get that the total time for this step is $O(\sum_{i=1}^{n} 2^{|L(i)|})$.

Summing up, the worst-case time used by the algorithm is

$$\begin{aligned}
&Time(n, m, m_1, ..., m_n, a, k, L(1), ..., L(n)) = \\
&\quad O(a * k^2 \ + \ n^4 * m \ + \ n * m \ + \\
&\qquad \textstyle\sum_{i=1}^{n}(2^{|L(i)|} * f_{SAT}(m_i)) \ + \ n \ + \ \sum_{i=1}^{n} 2^{|L(i)|}) = \\
&\quad O(a * k^2 \ + \ n^4 * m \ + \ \textstyle\sum_{i=1}^{n}(2^{|L(i)|} * f_{SAT}(m_i))).
\end{aligned}$$

We can reduce the second argument (in the last formula) from $n^4 * m$ to $n * m$, if the intersection graph G_0 is already a tree. ■

A.4 Proofs for Chapter 5: Decomposition

A.4.1 Proposition 5.4.1: Time Complexity for SPLIT

Proof of Proposition 5.4.1. Finding a minimum vertex separator R in G takes time $O(c * |V|^{\frac{3}{2}} * |E|)$. Finding a minimum separator that does not include s is equivalent to having s be the only source with which we check connectivity (in Even's algorithm). Thus, this can be done in time $O(|V|^{\frac{3}{2}} * |E|)$. Finding a minimum separator that separates s from t takes time $O(|V|^{\frac{1}{2}} * |E|)$. In the worst case, each time we look for a minimum s-separator ($t = nil$), we get a very small partition, and a very large one. Thus, we can apply this procedure $O(|V|)$ times. Summing up the time taken for each application of the procedure yields $O(|V| * |V|^{\frac{3}{2}} * |E| + c * |V|^{\frac{3}{2}} * |E|) = O(|V|^{\frac{5}{2}} * |E|)$. ■

A.4.2 Lemma 5.6.2: $\frac{2}{3}$-vtx-sep finds a $\frac{2}{3}$-separator.

Proof of Lemma 5.6.2. We prove the correctness of the algorithm first. Assume that the algorithm finds a separator X of S_1, S_2 in G'. X is also a separator of S_1, S_2 in G, by the way we constructed G' from G. Also, X separates $W^1 \setminus X$ and $W^2 \setminus X$ in G' because $W^1 \cup \{v_{W^1}\}$ and $W^2 \cup \{v_{W^2}\}$ are cliques in G' and X separates v_{W^1}, v_{W^2} (if X does not separate $W^1 \setminus X$ and $W^2 \setminus X$ in G', then there is a path between v_{W^1}, v_{W^2} that does not go through X). Finally, X is a $\frac{2}{3}$-vertex-separator of W because $|W^1|, |W^2| \geq \frac{|W|}{3}$, $W^1 \setminus X \subset S_1$ and $W^2 \setminus X \subseteq S_2$, so $|S_i \cap W| \leq \frac{2}{3}|W|$, for $i = 1, 2$. Notice that S_1, S_2 are never empty because $|X| \leq k$ and $|S_i| \geq |W^i| - |X| \geq 1$ for $i = 1, 2$ ($|W^i| \geq \lceil \frac{|W|}{3} \rceil = k + 1$ because $|W| = 3k + 2$).

For the reverse direction, assume that the treewidth of G is $k - 1$ and we show that the algorithm

will find a suitable separator. Assume first that there are two sets of vertices S_1, S_2 separated by X in G such that $S_1 \cup S_2 \cup X = V$ and $|S_i \cap W| \leq \frac{|W|}{2}$, for $i = 1, 2$. Let $W^i_{\text{n_sep}} = W \cap S_i$, for $i = 1, 2$. Let $W^i_{\text{sep}} \subseteq W \cap X$ such that $W^1_{\text{sep}} \cup W^2_{\text{sep}} = W \cap X$ and $|W^i_{\text{sep}} \cup W^i_{\text{n_sep}}| = \frac{|W|}{2}$. Let $W^i = W^i_{\text{sep}} \cup W^i_{\text{n_sep}}$, for $i = 1, 2$. Then, X separates $W^1 \setminus X, W^2 \setminus X$, as $W^i \setminus X = W^i_{\text{n_sep}}$, for $i = 1, 2$. Thus, running steps 2,3 in our algorithm using this selection of W^1, W^2 will find a separator of size $\leq |X| \leq k$. By the previous paragraph, this separator is a $\frac{2}{3}$-vertex-separator of W in G.

Now we show that if there are no such sets S_1, S_2, X, then our algorithm still finds a suitable separator. By Lemma 5.5.5, there are three sets, A, B, C, of vertices separated by X in G such that $|X| \leq k$ and $|W \cap C| \leq |W \cap B| \leq |W \cap A| \leq \frac{1}{2}|W|$. Let $S_1 = A$, $S_2 = B \cup C$. If $|S_2 \cap W| \leq \frac{|W|}{2}$, then the first selection case would cover this W (the previous paragraph). Thus, $|S_2 \cap W| > \frac{|W|}{2}$. Take $W^1 \subset (S_2 \cap W)$ of size $\frac{|W|}{2}$ and $W^2 \subset ((S_1 \cup X) \cap W) \setminus W^1$ of size $\frac{|W|}{3}$. The selection of W^2 is possible because $|(S_1 \cup X) \cap W| = |S_1 \cap W| + |X \cap W| \geq \frac{1}{3}|W \setminus X| + |X \cap W| = \frac{1}{3}|W|$. For this selection of W^1, W^2 our algorithm will find a separator of size $\leq |X| \leq k$ because X is already a separator of $W^1, W^2 \setminus X$. By the first paragraph in this proof, this separator is a $\frac{2}{3}$-vertex-separator of W in G.

Finally, each choice of W^1 takes $O(f(|V|, |E| + k^2, k))$ time to check, for $f(n, m, k)$ the time taken by a *min-(a, b)-vertex-separator* algorithm over a graph with n vertices, m edges and treewidth $k - 1$. There are $\binom{3k+2}{1.5k+1}$ ways to choose $1.5k + 1$ elements (W^1) from a set of $3k + 2$ elements (W). Also, there are $\binom{1.5k+1}{k+1}$ ways to choose $k + 1$ elements (W^2) from a set of $1.5k + 1$ elements ($W \setminus W^1$). Since $\binom{3k+2}{1.5k+1} = O(\frac{2^{3k}}{\sqrt{k}})$ and $\binom{1.5k+1}{k+1} = O(\frac{2^{1.3776k}}{\sqrt{k}})$ (using Stirling's approximation), we get the time bound of $O(\frac{2^{4.3776k}}{k} f(|V|, |E| + k^2, k))$. ■

A.4.3 Theorem 5.6.4: 2way-2/3-triang Finds a Factor-4 Approximation.

Proof of Theorem 5.6.4. Lemmas 5.6.1 and 5.6.2 prove the correctness. For the time bound, (Reed, 1992) showed that there are $O(|V|)$ recursive calls to such triangulation algorithms. Since each recursive step runs $\frac{2}{3}$-*vtx-sep* once and makes a clique of size $\leq 4k + 2$, we get that the combined procedure using (Even, 1979)'s algorithm for min-(a, b)-vertex-separator (time $O(|V|^{\frac{1}{2}}|E|)$) takes time $O(\frac{2^{4.38k}}{k}|V|^{\frac{1}{2}}(|E| + k^2)|V|)$. Using Proposition 5.6.3 we get the bound $O(\frac{2^{4.38k}}{k}|V|^{\frac{3}{2}}(|V|k + k^2)) = O(2^{4.38k}|V|^{\frac{5}{2}})$. Similarly, using the algorithm given by (Ford Jr. and Fulkerson, 1962) for finding a minimum (a, b)-vertex-separator in time $O(k(n + m))$ we get time $O(2^{4.38k}|V|^2 k)$. ■

A.4.4 Lemma 5.6.5: Existence of Two-Way $\frac{1}{2}$-Vertex-Separators of Size $k + \frac{1}{6}|W|$

Proof of Lemma 5.6.5. By Lemma 5.5.5 there are $A, B, C \subset V$ separated by X such that $A \cup B \cup C \cup X = V$, $|X| \leq k$ and $|W \cap C| \leq |W \cap B| \leq |W \cap A| \leq \frac{1}{2}|W|$. If $|(B \cup C) \cap W| \leq \frac{1}{2}|W|$, then $A, (B \cup C)$ and X satisfy our desired conditions.

Thus, assume that $|(B \cup C) \cap W| > \frac{1}{2}|W|$. Take $X_C \subset W \cap C$ of size $|(B \cup C) \cap W| - \frac{1}{2}|W|$. Then $|X_C| = |(B \cup C) \cap W| - \frac{1}{2}|W| \leq \frac{2}{3}|W| - \frac{1}{2}|W| = \frac{1}{6}|W|$. Let $X' = X \cup X_C$, $S_1 = A$ and $S_2 = (B \cup C) \setminus X_C$. This X', S_1, S_2 satisfy the desired conditions because $|S_2 \cap W| \leq \frac{1}{2}|W|$, $|S_1| \leq \frac{1}{2}|W|$, $|X'| \leq |X| + |X_C| \leq k + \frac{1}{6}|W|$ and X' separates S_1, S_2 (because X separates S_1, S_2). ■

A.4.5 Lemma 5.6.6: Correctness of 2way-half-triang

Proof of Lemma 5.6.6. If the algorithm outputs that the treewidth is more than $k - 1$, then it did not find a decomposition of W as needed. If the treewidth is at most $k - 1$, then Lemma 5.6.5 guarantees the existence of a two-way $\frac{1}{2}$-vertex-separator of W in G with size at most $k + \frac{1}{6}|W|$. Thus, this separator is of size at most $k + \frac{1}{6}|W| \leq k + \frac{1}{6}(3k + 2) = 1\frac{1}{2}k + \frac{1}{3}$ (and because the size cannot be fractional, it is at most $1\frac{1}{2}k$). If we did not find such a separator, then the treewidth is indeed at most $k - 1$.

The same argument used for the proof of Lemma 5.6.1 shows that the algorithm always terminates and, if it is successful, then it returns a graph that is triangulated.

We show that the clique number of this triangulation is at most $4\frac{1}{2}k+2$. First, notice that always $|W| \leq 3k + 2$. Initially, $|W| \leq 3k + 2$ by our assumption in the statement of the lemma. As the algorithm is called recursively, $|X| \leq 1\frac{1}{2}k$ and $|W_i| \leq \frac{1}{2}|W'| = 1\frac{1}{2}k + 1$. Thus, $|W_i \cup X| \leq 1\frac{1}{2}k + 1 + 1\frac{1}{2}k = 3k + 1$, which concludes the induction step (W in the recursive call to the algorithm is $W_i \cup X$).

Now, let M be a maximal clique. If M contains no vertex of $S_i \setminus W_i$, for $i = 1, 2$, then M contains only vertices of $W \cup X$. Thus, $|M| \leq 3k + 2 + 1\frac{1}{2}k = 4\frac{1}{2}k + 2$. On the other hand, if M contains a vertex of $S_i \setminus W_i$, then it does not contain any vertex of S_j, for $j \neq i$. This is because X vertex-separates S_1, S_2 (any two separated vertices cannot have an edge connecting them). Hence, M is a clique in the triangulation of $G[S_i \cup X]$. By induction we know that $|M| \leq 4\frac{1}{2}k + 2$. This proves the lemma. ■

A.4.6 Lemma 5.6.7: Time Complexity and Correctness of $\frac{1}{2}$-vtx-sep

Proof of Lemma 5.6.7. We prove the correctness of the algorithm. First, assume that the algorithm finds a separator X of S_1, S_2 in G'. X is also a separator of S_1, S_2 in G, by the way we constructed G' from G. Also, X separates $W^1 \setminus X$ and $W^2 \setminus X$ in G' because $W^1 \cup \{v_{W^1}\}$ and $W^2 \cup \{v_{W^2}\}$ are cliques in G' and X separates v_{W^1}, v_{W^2} (if X does not separate $W^1 \setminus X$ and $W^2 \setminus X$ in G', then there is a path between v_{W^1}, v_{W^2} that does not go through X). Finally, X is a $\frac{1}{2}$-vertex-separator of W because $|W^1| = |W^2| = \frac{|W|}{2}$, $W^1 \setminus X \subset S_1$ and $W^2 \setminus X \subseteq S_2$. S_1, S_2 are non-empty because $|S_1| \geq |W| - |X| - |W^2| \geq 3k + 2 - 1\frac{1}{2}k - (\frac{1}{2}k + 1) = 1$ (similarly for S_2).

Now, assume that there is a two-way $\frac{1}{2}$-vertex-separator X of W in G with $|X| \leq 1\frac{1}{2}k$. Let S_1, S_2 be two separated sets of vertices in G such that $S_1 \cup S_2 \cup X = V$ and $|S_i \cap W| \leq \frac{|W|}{2}$, for $i = 1, 2$. Let $W^i_{\text{n_sep}} = W \cap S_i$, for $i = 1, 2$. Let $W^i_{\text{sep}} \subseteq W \cap X$ such that $W^1_{\text{sep}} \cup W^2_{\text{sep}} = W \cap X$ and $|W^i_{\text{sep}} \cup W^i_{\text{n_sep}}| = \frac{|W|}{2}$. Let $W^i = W^i_{\text{sep}} \cup W^i_{\text{n_sep}}$, for $i = 1, 2$. Then, X separates $W^1 \setminus X, W^2 \setminus X$, as $W^i \setminus X = W^i_{\text{n_sep}}$, for $i = 1, 2$. Thus, running steps 2,3 in our algorithm using this selection of W^1, W^2 will find a separator of size $\leq |X| \leq 1\frac{1}{2}k$. By the previous paragraph, this separator is a $\frac{1}{2}$-vertex-separator of W in G.

Finally, each choice of W^1 takes $O(f(|V|, |E| + k^2, k))$ time to check, for $f(n, m, k)$ the time taken by a *min-(a, b)-vertex-separator* algorithm over a graph with n vertices, m edges and treewidth $k - 1$. There are $\binom{3k+2}{1\frac{1}{2}k+1}$ ways to choose $1\frac{1}{2}k + 1$ elements (W^1) from a set of $3k + 2$ elements (W). Since $\binom{2n}{n} = \frac{2^{2n}}{\sqrt{\pi n}}(1 + O(\frac{1}{n}))$, we get the time bound of $O(\frac{2^{3k}}{\sqrt{k}} f(|V|, |E| + k^2, k))$. ■

A.4.7 Lemma 5.7.1: Correctness and Time for α-sum-sep

Proof of Lemma 5.7.1. We prove the correctness of the algorithm first. Assume that the algorithm finds a α-sum-separator X of S_1, S_2, S_3 in G'. X is also a separator of S_1, S_2 in G, by the way we constructed G' from G. Also, X separates $W^i \setminus X$ and $W^j \setminus X$, $i \neq j \leq 3$, in G' because $W^i \cup \{v_{W^i}\}$ and $W^j \cup \{v_{W^j}\}$ are cliques in G' and X separates v_{W^i}, v_{W^j}.

To see that X is an α-sum-separator of W we examine two cases. In the first, $|W^1| \leq k$. Thus, $|X| \leq \alpha k$ (otherwise we return "*failure*"). $|(S_i \cap W) \cup X| \leq |W^i| + |X| \leq (1+\alpha)k$, for $i = 1, 2, 3$, because $S_i \cap W \subseteq W^i$. Thus, this is an α-sum-decomposition. In the second case, $|W^1| > k$. Thus, $|W^2 \cup W^3| < \alpha k$ because $|W| = (1 + \alpha)k$. Also, $|X| \leq k$ because it was returned by step 3 of algorithm $\frac{2}{3}$-vtx-sep (Figure 5.8). Thus, $|(S_i \cap W) \cup X| \leq |W^i| + |X| \leq (1 + \alpha)k$. Notice that $|S_i \cap W| \geq 1$, for at least two of $i = 1, 2, 3$ (i.e., X does not contain at least two of the W^i's). In the

first case this is because $|W^2 \cup W^3| = |W| - |W^1| \geq \alpha k + 1 > |X|$ (we set $|W| = (1+\alpha)k + 1$). In the second case this is because $|X| \leq k$, $|W^1| > k$ and $|W^2| = |W| - |W^1| \geq |W| - \frac{|W|}{2} > k$.

For the reverse direction, assume that the treewidth of G is $k-1$. We show that the algorithm finds a suitable separator. Let S_1, S_2, S_3 be three sets as guaranteed by Lemma 5.5.5, separated by X in G such that $S_1 \cup S_2 \cup S_3 \cup X = V$, $|S_3 \cap W| \leq |S_2 \cap W| \leq |S_1 \cap W| \leq \frac{|W|}{2}$ and $|X| \leq k$.

If $|W \cap S_1| \leq k$, then let $W^i_{\text{n_sep}} = W \cap S_i$, for $i = 1, 2, 3$. Let $W^i_{\text{sep}} \subseteq W \cap X$ such that $W^1_{\text{sep}} \cup W^2_{\text{sep}} \cup W^3_{\text{sep}} = W \cap X$ and $|W^i_{\text{sep}} \cup W^i_{\text{n_sep}}| \leq \frac{|W|}{2}$. Let $W^i = W^i_{\text{sep}} \cup W^i_{\text{n_sep}}$, for $i = 1, 2, 3$. Then, X separates $W^1 \setminus X, W^2 \setminus X, W^3 \setminus X$ because $W^i \setminus X = W^i_{\text{n_sep}}$, for $i = 1, 2, 3$. Thus, running steps 3,4 in our algorithm using this selection of W^1, W^2, W^3 will find a separator of size $\leq \alpha|X| \leq \alpha k$. By the first part of the proof, this separator is a α-sum-separator of W in G.

If $|W \cap S_1| > k$, then let $W^1_{\text{n_sep}} = W \cap S_1$ and $W^2_{\text{n_sep}} = W \cap (S_2 \cup S_3)$. Let $W^i_{\text{sep}} \subseteq W \cap X$, $i = 1, 2$ such that $W^1_{\text{sep}} \cup W^2_{\text{sep}} = W \cap X$ and $|W^i_{\text{sep}} \cup W^i_{\text{n_sep}}| \leq \frac{|W|}{2}$. Let $W^i = W^i_{\text{sep}} \cup W^i_{\text{n_sep}}$, for $i = 1, 2$. Then, X separates $W^1 \setminus X, W^2 \setminus X$ because $W^i \setminus X = W^i_{\text{n_sep}}$, for $i = 1, 2$. Thus, running steps 2,3 of algorithm $\frac{2}{3}$-vtx-sep (Figure 5.8) using this selection of W^1, W^2 will find a separator of size $\leq |X| \leq k$. By the first part of the proof, this separator is a α-sum-separator of W in G.

Finally, each choice of W^1 takes $O(f(|V|, |E| + k^2, k))$ time to check, for $f(n, m, k)$ the time taken by a α-approximating *3-way-vertex-separator* algorithm (or a minimum vertex separator algorithm, if it takes more time than the approximate 3-way-vertex-separator) over a graph with n vertices, m edges and treewidth $k-1$. There are at most $3^{|W|}$ ways to divide W into three sets. Since $|W| \leq (1+\alpha)k + 1$, we run a vertex separation algorithm at most $3^{(1+\alpha)k+1} = O(3^{2\frac{1}{3}k}) = O(2^{3.6982k})$ times, for $\alpha = \frac{4}{3}$. Thus, the total time is $O(2^{3.6982k} f(|V|, |E| + k^2, k))$. ■

A.4.8 Lemma 5.7.3: Correctness of lgk-triang

Proof of Lemma 5.7.3. As in the proofs for previous algorithms, this algorithm either outputs (correctly) that the treewidth exceeds k, or a triangulated graph. We show that the clique number of this triangulation is at most $\gamma klgk$.

First, notice that always $|W| \leq \gamma_1 klgk$. Initially, $|W| \leq \gamma_1$ by our assumption in the statement of the lemma. As the algorithm is called recursively, $|X| \leq \beta klgk$ and $|W_i| \leq \frac{2}{3}|W| \leq \frac{2}{3}\gamma_1 klgk$, by induction. Thus, $|W_i \cup X| \leq \frac{2}{3}\gamma_1 klgk + \beta klgk$. Since $\gamma_1 = 3\beta$ we get that $|W_i \cup X| \leq \gamma_1 klgk$ which concludes the induction step (W in the recursive call to the algorithm is $W_i \cup X$).

Now, let M be a maximal clique. If M contains no vertex of $S_i \setminus W_i$, for $i = 1, 2, 3$, then M contains only vertices of $W \cup X$. Thus, $|M| \leq \gamma_1 klgk + \beta klgk = \gamma klgk$. On the other hand, if

M contains a vertex of $S_i \setminus W_i$, then it does not contain any vertex of S_j, for $j \neq i$. This is because X vertex-separates S_1, S_2, S_3 (any two separated vertices cannot have an edge connecting them). Hence, M is a clique in the triangulation of $G[S_i \cup X]$. By induction we know that $|M| \leq \gamma klgk$. This proves the lemma. ■

A.5 Proofs for Chapter 6: LSA

A.5.1 Theorem 6.4.1 and Corollary 6.4.2: Interpolation for Circumscription

This section brings a few technical results that are important for establishing our proposed semantics for LSA.

Lemma A.5.1 *Let T_1, T_2 be two theories, with signatures in L_1, L_2, respectively. Let γ be the set of prime implicates of T_2 in $\mathcal{L}(L_1 \cap L_2)$. For every L_1-structure, $\mathcal{M}$, that satisfies $T_1 \cup \gamma$ there is a $(L_1 \cup L_2)$-structure, $\bar{\mathcal{M}}$, that is a model of $T_1 \cup T_2$ such that $\mathcal{M} \preceq \bar{\mathcal{M}} \restriction \mathcal{L}(L_1)$.*

PROOF Let $\mathcal{M}$ be a L_1-structure that is a model of $T_1 \cup \gamma$. Then $\mathcal{M} \models \gamma$. Notice that $\gamma \models Cn^{L_1}(T_2)$ because γ is the set of all prime implicates of T_2 in $L_1 \cap L_2$, so $\gamma \models Cn^{L_1 \cap L_2}(T_2)$ and $Cn^{L_1}(Cn^{L_1 \cap L_2}(T_2)) = Cn^{L_1}(T_2)$. Theorem B.1.11 with $L = L_1$, $L^+ = L_1 \cup L_2$ and $T = T_2$ implies that there is a $(L_1 \cup L_2)$-structure, $\bar{\mathcal{M}}$, that is a model of T_2 such that $\mathcal{M} \preceq \bar{\mathcal{M}} \restriction \mathcal{L}(L_1)$. ■

Lemma A.5.2 *If $\mathcal{M} \models Circ[T; P; Q]$ and $\mathcal{M} \models T \cup T'$, then $\mathcal{M} \models Circ[T \cup T'; P; Q]$.*

PROOF Let $\mathcal{M}$ be a model of $T \cup T'$ such that $\mathcal{M} \models Circ[T; P; Q]$. If there is $\mathcal{M}' <^{P,Q} \mathcal{M}$ such that $\mathcal{M}' \models T \cup T'$, then $\mathcal{M}' \models T$ and $\mathcal{M} \not\models Circ[T; P; Q]$. Contradiction. Thus, there is no such $\mathcal{M}'$ and $\mathcal{M} \models Circ[T \cup T'; P; Q]$. ■

Lemma A.5.3 *Let T be a theory and P, Q, Q' sequences of nonlogical symbols such that $Q' \subseteq Q$. If $\mathcal{M} \models Circ[T; P; Q]$, then $\mathcal{M} \models Circ[T; P; Q']$.*

PROOF If $\mathcal{M}' <^{P,Q'} \mathcal{M}$, then $\mathcal{M}' <^{P,Q} \mathcal{M}$. Thus if $\mathcal{M}'$ is a model of T, then $\mathcal{M} \not\models Circ[T; P; Q]$. From $\mathcal{M} \models Circ[T; P; Q]$, we get that $\mathcal{M}$ is also a $\leq^{P,Q'}$-minimal model of T. Thus, $\mathcal{M} \models Circ[T; P; Q']$. ■

Lemma A.5.4 *Let T be a theory and P, Q sequences of nonlogical symbols. Let $Q' \supseteq Q$ such that $(Q' \setminus Q) \cap L(T) = \emptyset$ Let $\mathcal{M}$ be a $(L(T) \cup Q')$-structure such that $\mathcal{M} \models Circ[T; P; Q]$. Then $\mathcal{M} \models Circ[T; P; Q']$.*

PROOF Assume that there is $\mathcal{M}' <^{P,Q'} \mathcal{M}$, and $\mathcal{M}' \models T$. Take M'' such that the interpretation of all the symbols in $L(T)$ is exactly the same as that of $\mathcal{M}'$ and such that the interpretation of all symbols in $Q' \setminus Q$ is exactly the same as that of $\mathcal{M}$. Then, $\mathcal{M}'' \models T$ and $\mathcal{M}'' <^{P,Q'} \mathcal{M}$. Thus, $\mathcal{M}'' <^{P,Q} \mathcal{M}$, since $\mathcal{M}'', \mathcal{M}$ agree on all the interpretation of all symbols in $Q' \setminus Q$. Thus, $\mathcal{M}$ is not a $\leq^{P,Q}$-minimal model of T, contradicting the assumptions of the lemma. Thus, there is no such $\mathcal{M}'$ and $\mathcal{M} \models Circ[T; P; Q']$. ■

Lemma A.5.5 *Let T_1, T_2 be two theories and P, Q two vectors of symbols from $L(T_1) \cup L(T_2)$ such that $P \subset L(T_1)$. Let γ be the set of prime implicates of T_2 in $L(T_1) \cap L(T_2)$. Then, for all $\varphi \in \mathcal{L}(T_1)$, if $Circ[T_1 \cup T_2; P; Q] \models \varphi$, then $Circ[T_1 \cup \gamma; P; Q] \models \varphi$.*

PROOF We show that for every model of $Circ[T_1 \cup \gamma; P; Q]$ there is a model of $Circ[T_1 \cup T_2; P; Q]$ whose reduct to $L(T_1)$ is an elementary extension of the reduct of the first model to $L(T_1)$.

Let $\mathcal{M}$ be a $L(T_1 \cup T_2)$-structure that is a model of $Circ[T_1 \cup \gamma; P; Q]$. Then, $\mathcal{M} \models T_1 \cup \gamma$. From Lemma A.5.1 we know that there is a $(L_1 \cup L_2)$-structure, $\bar{\mathcal{M}}$, that is a model of T_2 such that $\mathcal{M} \restriction \mathcal{L}(T_1) \preceq \bar{\mathcal{M}} \restriction \mathcal{L}(T_1)$.

Thus, $\bar{\mathcal{M}}$ is a $\leq^{P,Q}$-minimal model of $T_1 \cup \gamma$. To see this, assume otherwise. Then, there is a model $\mathcal{M}'$ for the signature $L(T_1 \cup T_2)$ such that $\mathcal{M}' <^{P,Q} \bar{\mathcal{M}}$ and $\mathcal{M}' \models T_1 \cup \gamma$. Take $\mathcal{M}''$ such that the interpretation of all the symbols in $L(T_1)$ is exactly the same as that of $\mathcal{M}'$ and such that the interpretation of all symbols in $L(T_2) \setminus L(T_1)$ is exactly the same as that of $\mathcal{M}$. Then, $\mathcal{M}'' \models T_1 \cup \gamma$ because $T_1 \cup \gamma \subseteq \mathcal{L}(T_1)$. Also, $\mathcal{M}'' <^{P,Q'} \mathcal{M}$, for $Q' = Q \cap L(T_1)$ because $P \subseteq L(T_1)$ and $\mathcal{M}, \bar{\mathcal{M}}$ agree on the interpretation of symbols in $L(T_1)$ ($\mathcal{M} \restriction \mathcal{L}(T_1) \preceq \bar{\mathcal{M}} \restriction \mathcal{L}(T_1)$). Thus, $\mathcal{M}'' <^{P,Q} \mathcal{M}$, since $\mathcal{M}'', \mathcal{M}$ agree on all the interpretation of all symbols in $L(T_2) \setminus L(T_1)$. This contradicts $\mathcal{M} \models Circ[T_1 \cup \gamma; P; Q]$, so $\bar{\mathcal{M}}$ is a $\leq^{P,Q}$-minimal model of $T_1 \cup \gamma$.

Thus, $\bar{\mathcal{M}} \models Circ[T_1 \cup \gamma; P; Q]$, and $\bar{\mathcal{M}} \models T_1 \cup T_2$. From Lemma A.5.2 we get that $\bar{\mathcal{M}} \models Circ[T_1 \cup T_2; P; Q]$.

Now, let $\varphi \in \mathcal{L}(T_1)$ such that $Circ[T_1 \cup T_2; P; Q] \models \varphi$. Then every model of $Circ[T_1 \cup T_2; P; Q]$ satisfies φ. Let $\mathcal{M}$ be a model of $Circ[T_1 \cup \gamma; P; Q]$ in the language $\mathcal{L}(T_1 \cup T_2)$. Then there is $\bar{\mathcal{M}}$ as above, i.e., $\bar{\mathcal{M}} \models Circ[T_1 \cup T_2; P; Q]$ and $\mathcal{M} \restriction \mathcal{L}(T_1) \preceq \bar{\mathcal{M}} \restriction \mathcal{L}(T_1)$. Thus,

$\bar{\mathcal{M}} \models \varphi$. Since $\mathcal{M} \restriction \mathcal{L}(T_1) \preceq \bar{\mathcal{M}} \restriction \mathcal{L}(T_1)$, $\mathcal{M} \models \varphi$. Thus every model of $Circ[T_1 \cup \gamma; P; Q]$ is a model of φ. ■

Lemma A.5.6 *Let T_1, T_2 be two theories, P, Q two vectors of symbols from $L(T_1) \cup L(T_2)$ such that $P \subset L(T_1)$ and $P \cup Q \supset L(T_2)$. Let γ be the set of prime implicates of T_2 in $L(T_1) \cap L(T_2)$. Then, for all $\varphi \in \mathcal{L}(T_1)$, if $Circ[T_1 \cup \gamma; P; Q] \models \varphi$, then $Circ[T_1 \cup T_2; P; Q] \models \varphi$.*

PROOF We show that for every model of $Circ[T_1 \cup T_2; P; Q]$ is also a model of $Circ[T_1 \cup \gamma; P; Q]$. Let $\mathcal{M}$ be a $L(T_1 \cup T_2)$-structure that is a model of $Circ[T_1 \cup T_2; P; Q]$. Then $\mathcal{M} \models T_1 \cup T_2$, implying that also $\mathcal{M} \models T_1 \cup \gamma$.

Assume that there is $\mathcal{M}' <^{P,Q} \mathcal{M}$ such that $\mathcal{M}' \models T_1 \cup \gamma$. From Lemma A.5.1, there is $\bar{\mathcal{M}}'$ such that $\bar{\mathcal{M}}' \models T_1 \cup T_2$ and $\mathcal{M}' \restriction \mathcal{L}(T_1) \preceq \bar{\mathcal{M}}' \restriction \mathcal{L}(T_1)$. Since $\mathcal{M}', \bar{\mathcal{M}}'$ agree on all the symbols of $L(T_1)$, we get that $\bar{\mathcal{M}}' \leq^{P,Q} \mathcal{M}'$ (because $P \cup Q \supseteq L(T_2)$). Finally, we get that $\bar{\mathcal{M}}' \leq^{P,Q} \mathcal{M}' <^{P,Q} \mathcal{M}$, contradicting the assumption of $\mathcal{M}$ being $\leq^{P,Q}$-minimal satisfying $T_1 \cup T_2$. Thus, $\mathcal{M}$ is a model of $Circ[T_1 \cup \gamma; P; Q]$.

Now, let $\varphi \in \mathcal{L}(T_1)$ such that $Circ[T_1 \cup \gamma; P; Q] \models \varphi$. Then every model of $Circ[T_1 \cup \gamma; P; Q]$ satisfies φ. Let $\mathcal{M}$ be a model of $Circ[T_1 \cup T_2; P; Q]$ in the language $\mathcal{L}(T_1 \cup T_2)$. Then, $\mathcal{M} \models Circ[T_1 \cup \gamma; P; Q]$ and $\mathcal{M} \models \varphi$. Thus, $Circ[T_1 \cup T_2; P; Q] \models \varphi$. ■

From Lemma A.5.5 and Lemma A.5.6 we get the following theorem (Theorem 6.4.1).

Theorem A.5.7 *Let T_1, T_2 be two theories, P, Q vectors of symbols in $L(T_1) \cup L(T_2)$ such that $P \subseteq L(T_1)$ and $P \cup Q \supset L(T_2)$. Let γ be the set of prime implicates of T_2 in $\mathcal{L}(L(T_1) \cap L(T_2))$. Then, for every $\varphi \in \mathcal{L}(T_1)$,*

$$Circ[T_1 \cup \gamma; P; Q] \models \varphi \iff Circ[T_1 \cup T_2; P; Q] \models \varphi$$

Finally, we conclude Corollary 6.4.2.

Corollary A.5.8 (Interpolation Theorem for Circumscription) *Let T be a theory, P, Q vectors of symbols in $L(T)$ such that $(P \cup Q) \supseteq L(T)$, $P \subseteq L(\varphi)$. Then, if $Circ[T; P; Q] \models \varphi$, then there is $\gamma \in \mathcal{L}(T) \cap \mathcal{L}(\varphi)$ such that*

$$T \models \gamma \quad \textit{and} \quad Circ[\gamma; P; Q] \models \varphi.$$

Furthermore, this γ can be the set of prime implicates of T in $L(T) \cap L(\varphi)$.

PROOF Let $T_1 = \emptyset$, $T_2 = T$, $L_1 = L(\varphi)$, $L_2 = L(T_2)$. Then, Theorem 6.4.1 guarantees that if $P \subseteq L_1$ then γ from that theorem satisfies $Circ[\gamma; P; Q] \models \psi \iff Circ[T; P; Q] \models \psi$ for every $\psi \in \mathcal{L}_1$. In particular, $Circ[\gamma; P; Q] \models \varphi$. ■

A.5.2 Theorem 6.4.4: LSA is Complete for *Sensors to Actions*

We prove our theorem by induction on the number of layers. By definition 6.4.3, if T has only one layer ($Layer_0$), then $T \models \varphi$ iff $Circ[Layer_0; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi$. For $\varphi_0 = \varphi$ and $k = 0$ we get that $Circ[Layer_k; \overrightarrow{C}_k; \overrightarrow{Z}_k] \models \varphi_k$. This proves the theorem for $n = 0$.

Assume that the theorem is correct for n and we prove it for $n + 1$. Let T' be the set of layers of T without $Layer_0$. Let $\varphi \in \mathcal{L}(Layer_0)$ such that $T \models \varphi$. By definition, $Circ[Layer_0 \cup Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi$.

Let γ be the set of prime implicates of $Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]$ in $L(T') \cap L(Layer_0)$. From the induction hypothesis we know that there are $k \geq 0$ and $\varphi_1, ...\varphi_k$ such that $\varphi_1 = \gamma$ and $Circ[Layer_k; \overrightarrow{C}_k; \overrightarrow{Z}_k] \models \varphi_k$, and for all i such that $1 \leq i < k$, $\varphi_i \in \mathcal{L}(Goal_i)$ and $Circ[Layer_i \cup \varphi_{i+1}; \overrightarrow{C}_i; \overrightarrow{Z}_i] \models \varphi_i$.

Recall that we assume that for every $i < n$, only predicates of $L(Goal_i)$ can appear in both $Layer_i, Layer_{i+1}$. From Theorem 6.4.1,

$$Circ[Layer_0 \cup \gamma; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi \iff Circ[Layer_0 \cup Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi$$

Thus, taking $\varphi_0 = \varphi$, and $\varphi_1 = \gamma$ provides the induction step, and $Circ[Layer_0 \cup \gamma; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi$. ■

A.5.3 Theorem 6.4.4: LSA is Sound for *Sensors to Actions*

We prove our theorem by induction on the number of layers. By definition 6.4.3, if T has only one layer ($Layer_0$), then $T \models \varphi$ iff $Circ[Layer_0; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi$. For $k = 0$ we get that $Circ[Layer_k; \overrightarrow{C}_k; \overrightarrow{Z}_k] \models \varphi_k$. This proves the theorem for $n = 0$.

Assume that the theorem is correct for n and we prove it for $n + 1$. Let T' be the set of layers of T without $Layer_0$. By definition, $Circ[Layer_0 \cup Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi_0$.

From the induction hypothesis we know that $T' \models \varphi_1$ because φ_1 is the set of prime implicates of $Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]$ in $L(T') \cap L(Layer_0)$. Recall that we assume that for every $i < n$,

only predicates of $L(Goal_i)$ can appear in both $Layer_i, Layer_{i+1}$. From Theorem 6.4.1,

$$Circ[Layer_0 \cup \varphi_1; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi_0 \iff Circ[Layer_0 \cup Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi_0$$

Thus, $Circ[Layer_0 \cup Circ[Layer_1 \cup ...; \overrightarrow{C}_1; \overrightarrow{Z}_1]; \overrightarrow{C}_0; \overrightarrow{Z}_0] \models \varphi_0$, and the induction step is complete. ■

Appendix B

More on Logical Systems Appearing in This Thesis

B.1 First-Order Logic and Propositional Logic

This section is a short reminder of logic and some of its less-known corridors that are useful in this thesis. It it intended to be used as reference rather than as a source for learning about this material. For a more deductive presentation and tutorials on the topics discussed here, one should consider reading one of (Shoenfield, 1967; Genesereth and Nilsson, 1987; Fitting, 1990; Gabbay et al., 1993a). We focus on First-Order Logic (FOL) and propositional logic.

B.1.1 Propositional Logic

Propositional logic is the most simple logic, and the least expressive one that we are going to present here. Like every logic it has syntax, semantics, and proof theory.

The *syntax* includes

language primitives **propositions** $p, q, r...$

connectives $\vee, \wedge, \neg$

formulae Every proposition is a formula. Every formula preceded by $\neg$ is a formula. Every two formulae connected by either $\wedge, \vee$ is a formula. This is a recursive definition.

The *semantics* for a theory (a set of formulae) is a set of *interpretations* that satisfy this theory. An interpretation for a propositional language is an assignment of truth values to the propositions

in the language. The truth values of the sentences are then defined in the intuitive way.

B.1.2 First Order Logic: Syntax and Semantics

First order logic is comprised of a language, a proof theory and semantics. While there can be many proof theories and semantics to a logic, when plainly referring to *the semantics* we refer to the Tarski Semantics, and when plainly referring to *the proof theory* or *the calculus* we refer to the Frege-Hilbert calculus. Both the syntax, the semantics and the proof theory will be detailed below.

The language of FOL

The language of first order logic has a set of object constant symbols, a set of relation constant symbols, a set of constant function symbols, and a set of connectives and parentheses as the building operators. We shall describe the language in the infix form, but there are equivalent prefix and postfix forms. The set of constants and their characteristics is called the *signature* of the specific instance of the language, and it is sometimes written like this:

$$\langle U; F_1, ..., F_n; R_1, ..., R_m; O_1, ..., O_k \rangle$$

Depending on our choice, we may include equality in the language or not, but most of the time it will not be mentioned in the signature of the language but rather in the specification of the language (i.e., "first order logic with equality").

Atomic Terms These are either object constant symbols or variable symbols.

Terms Any atomic term or the application of a function to a set of terms (depending on the arity of the function).

Atomic Formulae Application of relation symbols to terms (given the arity of the relation).

Literals Atomic formulae or their negation.

Formulae Atomic Formulae or the application of connectives (including quantifiers) to formulae.

Ground Formulae Formulae in which there is no variable mentioned.

Open Formulae A formula with all variables free (i.e. not quantified).

Closed Formulae (Sentences) Formulae with all variable instances quantified.

Prenex Normal Form The prenex normal form of a formula is an equivalent sentence produced from the original formula by making all the quantifiers have the largest scope. This produces a formula of the form $Q\varphi$, where Q is the quantification part and φ is a quantifier-free formula. φ is then called the *matrix* of $Q\varphi$.

After building the formulae of the language, we start building higher constructs, called *theories*. In this text we will say that every set of formulae is a theory and vice versa[1]. We will detail the properties of theories in section B.1.3 on the proof theory.

The Semantics of FOL

In general, a semantic interpretation of a logical theory is a structure on which the constant symbols used in the theory are mapped. In first order logic this *structure* (also called *interpretation*) is comprised of a set of elements called the *universe* of the structure, and a mapping from object constant symbols to elements of the universe, from relation constant symbols to relations over the universe (depending on the arity of the relation symbol) and from function constant symbols to functions over the universe.

Given a structure $\mathcal{I}$, every formula is given a truth value that is calculated using the mapping of the constants of the language to the universe. We write $\mathcal{I} \models \varphi$ for a formula φ if it is given a truth value of T. Otherwise we write $\mathcal{I} \not\models \varphi$. The definition of $\models$ (which is a relation between sets of formulae and a formula) is recursively defined as follows:

Atomic Term $A^{\mathcal{I}}$ is the defined mapping. For a variable, it will just vary over the set of elements in the universe.

A Term $(f(t_1, ..., t_n))^{\mathcal{I}} = f^{\mathcal{I}}(t_1^{\mathcal{I}}, ..., t_n^{\mathcal{I}})$.

An atomic formula $\mathcal{I} \models R(t_1, ..., t_n)$ iff for every variable assignment by objects from $\mathcal{I}$ $\sigma = [x_1/a_1, ..., x_n/a_n]$ (where $x_1, ..., x_n$ are the only variables free in $t_1, ..., t_n$), $R^{\mathcal{I}}(t_1^{\mathcal{I}}, ..., t_n^{\mathcal{I}})\sigma$.

The connective "$\wedge$" $\mathcal{I} \models \varphi \wedge \psi \iff \mathcal{I} \models \varphi$ and $\mathcal{I} \models \psi$.

The connective "$\neg$" $\mathcal{I} \models \neg\varphi$ iff for every assignment by objects from $\mathcal{I}$ $\sigma = [x_1/a_1, ..., x_n/a_n]$ (where $x_1, ..., x_n$ are the only variables free in φ), $\mathcal{I} \not\models \varphi$.

The quantifier "$\exists$" $\mathcal{I} \models \exists x.\varphi$ iff there is an object a in the universe such that $\mathcal{I} \models \varphi_{[x/a]}$.

[1] Some texts refer to *theories* as deductively closed sets of formulae. We prefer this notation here.

The symbols $\vee, \Rightarrow, \iff, \forall$ are defined in the usual way from the above connectives. The symbol $\Rightarrow$ is sometime called *material implication* as it has no special meaning besides being defined as $p \Rightarrow q \stackrel{def}{=} \neg p \vee q$. Other connectives exists in extended languages.

We say that $\mathcal{M}$ is a *model* of a theory Γ if $\forall \varphi \in \Gamma \; \mathcal{M} \models \varphi$, and we write it $\Gamma \models \mathcal{M}$ (and also $\mathcal{M} \models \Gamma$ when Γ is finite).

We say that a theory Γ *logically entails* a formula φ, and write it $\Gamma \models \varphi$ if for every model $\mathcal{M}$ of Γ, $\mathcal{M} \models \varphi$.

A theory T is said to be *complete* if for every sentence φ, either $T \models \varphi$ or $T \models \neg\varphi$. The following is an important result due to Gödel (Gödel, 1967).

Theorem B.1.1 (Incompleteness of Natural Numbers in FOL) *For the theory of Natural Numbers $\mathcal{N}$ with the function symbols $+, *$, for every theory $\mathcal{T} \supset \mathcal{N}$, either $\mathcal{T}$ is not complete or it is not recursively enumerable.*

B.1.3 Proof Theory of First-Order Logic

The relation $\vdash$ is a relation between sets of sentences and sentences (as is $\models$). The difference between the relations is that $\vdash$ is concerned with derivations, while $\models$ is concerned with semantics.

The classical mathematical logic calculus, invented by Gottlob Frege ((Frege, 1879)) and David Hilbert ((Hilbert and Bernays, 1934; Hilbert and Ackermann, 1950)), is called the Frege-Hilbert calculus. In it, there are a set of axioms and a set of rules. Many such combinations exist, in most of which, the set of rules is much smaller than the set of axioms, but not necessarily[2] A *derivation* of φ from Δ is a string of FOL formulae, each formula is either an axiom in Δ or is proved using a rule from previous formulae in the string. If there is such a string, then we write $\Delta \vdash \varphi$.

This calculus is mostly useful for meta-theorems, as it is very simple and close to the definition of First Order Logic. It is not suitable for automatic deduction, because its forward reasoning has no guiding heuristics, whereas its backward reasoning has a potentially infinite (or very large) branching factor (mostly because of the *modus-ponens* or the *cut* rules). Other calculi include Natural Deduction, Sequent Calculus, Tableaux, the Connection method and Resolution. Details, on these and other calculi (including example axioms-rules systems), can be found in (Gabbay et al., 1993a).

Theorem B.1.2 (Soundness of the Frege-Hilbert proof system) $\Gamma \vdash \varphi \implies \Gamma \models \varphi$, *where φ is a sentence.*

[2] E.g., the formalization in (Shoenfield, 1967).

Put other words, if there is no proof of FALSE from Γ, then there is a model for Γ.

Theorem B.1.3 (Completeness of the Frege-Hilbert proof theory) $\Gamma \models \varphi \implies \Gamma \vdash \varphi$. *where* φ *is a sentence.*

Put other words, if there is a model of Γ, then there is no proof of $FALSE$ from Γ.

Theorem B.1.4 (Compactness) $\Gamma \vdash FALSE$ *iff there is a finite* $\Gamma_0 \subset \Gamma$ *such that* $\Gamma_0 \vdash FALSE$.

A theory Δ is *consistent* if $\Delta \nvdash FALSE$ (i.e., a sentence and its negation). Equivalently, it is consistent iff there is an interpretation $\mathcal{M}$ such that $\mathcal{M} \models \Delta$.

The Herbrand Universe

The *Herbrand Universe* of a theory is the set of ground terms that can be produced using the constants mentioned in the theory (if there are no object constant symbols at all, a constant symbol is artificially added).

The *Herbrand Base* of a set of clauses (a clause is a disjunction of open literals), Δ, is the set of all ground instances of the clauses in Δ (this set is produced using the Herbrand Universe as the set of instantiators).

A *Herbrand Interpretation* of Δ is then the Interpretation built from the Herbrand Universe as its universe (terms are mapped to themselves) and ground atoms are assigned either true of false.

Mostly, when constructing the Herbrand Interpretation, we rely on having a model that will guide us in giving the right values to the ground atoms. The Herbrand Universe by itself is therefore not useful for proofs of completeness (where the existence of the model is what we try to prove), without using a previous completeness result. It does become useful when we wish to either choose a specific interpretation (e.g., the canonical model in logic and logic programs) or we have a previous completeness result (e.g., for a different language).

Theorem B.1.5 (Herbrand) *If a* finite *set of clauses* Δ *is unsatisfiable, then the Herbrand base of* Δ *is unsatisfiable.*

The contrapositive theorem holds as well (if the Herbrand base is unsatisfiable then the set of clauses is unsatisfiable).

Interpolation Theorems

Jon Doyle's work (Doyle, 1985) on implicit definability and Nonmonotonic Reasoning put some theorems under the spotlight as important for logical-AI research in general. Here we state some of them.

The following is Craig's Interpolation Theorem. It is key to several completeness theorems used in this thesis.

Theorem B.1.6 (Craig's Interpolation Theorem (Craig, 1957a)) *Let α, β be sentences such that $\alpha \vdash \beta$. Then there is a formula γ involving only symbols common to both α and β (and possibly identity), such that $\alpha \vdash \gamma$ and $\gamma \vdash \beta$.*

Two similar theorems have been proved by Lyndon and Robinson with somewhat different applications.

Theorem B.1.7 (Robinson's Consistency Theorem) *Let $\mathcal{L}_1, \mathcal{L}_2$ be two languages and let $\mathcal{L} = \mathcal{L}_1 \cap \mathcal{L}_2$. Suppose T is a complete theory in $\mathcal{L}$ and $T \subset T_1, T_2$ are consistent theories in $\mathcal{L}_1, \mathcal{L}_2$, respectively. Then, $T_1 \cup T_2$ is consistent in $\mathcal{L}_1 \cup \mathcal{L}_2$.*

Theorem B.1.8 (Lyndon's Interpolation Theorem (Lyndon, 1959)) *Suppose $\mathcal{L}$ has no function or object constant symbols. Let φ, ψ be sentences such that $\varphi \vdash \psi$. Then there is a sentence ϕ such that $\varphi \vdash \phi$ and $\phi \vdash \psi$ and every relation symbol (besides $=$) which occurs positively [negatively] in ϕ occurs positively [negatively] in both φ and ψ.*

A implicitly defines P iff when we pick some symbol Q not in $\mathcal{L}$, we can prove $\forall x(P(x) \equiv Q(x))$ from $A(P) \cup A(Q)$.

Theorem B.1.9 (Beth Definability Theorem (Beth, 1953)) *Let A be a theory. A implicitly defines P iff there is a formula ϕ involving only the symbols of A exclusive of P such that*

$$A \vdash \forall x(P(x) \equiv \phi(x))$$

I.e., in FOL, implicit definability is equivalent to explicit definability. This theorem is not true for Higher-Order logic.

B.1.4 Model Theory of First-Order Logic

Definition B.1.10 *Let A, B be L-structures, for FOL signature L and language $\mathcal{L}$. We say that B is an* elementary extension *of A (or A is an* elementary substructure *of B), written $A \preceq B$, if $U(A) \subset U(B)$ and for every $\varphi(\overrightarrow{x}) \in \mathcal{L}$ and sequence of elements $\overrightarrow{a}$ of $U(A)$, $A \models \varphi$ iff $B \models \varphi$.*

$f : A \to B$ is an *elementary embedding* if f is an injective (one-to-one) homomorphism from A to B and for every $\varphi(\overrightarrow{x}) \in \mathcal{L}$ and sequence of elements $\overrightarrow{a}$ of $U(A)$, $A \models \varphi(\overrightarrow{a})$ iff $B \models \varphi(f(\overrightarrow{a}))$.

For FOL signatures $L \subseteq L_1$, and for B an L_1-structure, we say that $B \restriction L$ is the *reduct* of B to L, the L-structure with the same universe of elements as B, and the same interpretation as B for those symbols from L_1 that are in L (there is no interpretation for symbols not in L).

Theorem B.1.11 (See (Hodges, 1997) p.148) *Let L, L^+ be FOL signatures with $L \subseteq L^+$ and T a theory in the language of L^+. Let A be an L-structure, and let $Cn^L(T)$ be the set of all consequences of T in the language of L. Then, $A \models Cn^L(T)$ if and only if for some model B of T, $A \preceq B \restriction L$ (A is an elementary substructure of the reduct of B to L).*

Theorem B.1.12 (Heir-coheir amalgams) *Let A, B, C be structures such that $A \preceq B$ and $A \preceq C$. Then there exist an elementary extension D of B and an elementary embedding $g : C \to D$ such that $g(C) \preceq D$.*

B.1.5 Resolution Theorem Proving

Resolution

Resolution is an analytic and negative calculus. It uses two rules: The modus-ponens (realized as the *resolution rule*), and the *instantiation rule* (realized using unification). The resolution rule for FOL is:

$$
\begin{array}{l}
Q_1, ..., Q_n,\ P(t_1, ..., t_l),\ Q_n + 1, ..., Q_m \\
K_1, ..., K_k,\ \neg P(s_1, ..., s_l),\ K_k + 1, ..., K_p \\
- - - -

would reach a contradiction (an empty clause would be the result of resolution steps). The resolution calculus is also sound. It is semi-decidable.

Theorem B.1.13 (Lifting Theorem) *If Δ' is a set of ground instances of clauses in Δ and there is a resolution deduction of a clause X' from Δ' then there is a resolution deduction of a clause X from Δ such that X' is a substitution instance of X (other words: If we have a resolution refutation of the ground version, there is a resolution refutation of the original (not grounded) version).*

Theorem B.1.14 (Completeness of Resolution) *If a set Δ of clauses is unsatisfiable, then there is a resolution deduction of the empty clause from Δ.*

Some modifications of resolution are

- Using macro resolution steps - resolving several clauses simultaneously (Hyper Resolution).
- Theory Resolution - uses specially tailored inference rules instead of axioms, for some predicates for which we have some special information.
- Using the *kind* of theory - for some kinds of theories there are special heuristics/rules that we can apply. Three kinds of theories have been emphasized so far for this role: Algorithmic (such as equality reasoning (paramodulation), and different unifications), Representational theories (using sorts in our language), and Compiled theories (translating an axiom into a resolution rule).

Resolution is used for both answering yes-no questions as well as fill-in-the-blank question (using an $Ans(answer)$ addition to the negated goal) and planning (such as in Green's planner (Green, 1969)).

Unification

The unification of two n-tuples $\langle t_1, ..., t_l \rangle, \langle s_1, ..., s_l \rangle$ is a substitution σ such that the two arguments are equal under that substitution. The *mgu* (most general unifier) is the substitution σ such that for any other unifier σ' there is a substitution δ such that $\delta\sigma = \sigma'$.

Theorem B.1.15 *If there is a unifier to $\langle t_1, ..., t_l \rangle, \langle s_1, ..., s_l \rangle$, then they have an mgu, and a unique one (up to variable renaming).*

An algorithm for finding the mgu goes as follows:

1. Assume the arguments for the procedure are t, s (strings).
2. If s is a variable, return $[s/t]$ (after occurs check).
3. If t is a variable, return $[t/s]$ (after occurs check).
4. If s or t are constants, return *FALSE*.
5. Run sequentially on t, and for each element find (recursively) the mgu of that element with the corresponding element in s. After each mgu is found, apply it to the entire set of s, t, and continue on (i.e., every later unification is influenced by an earlier unification).

Clausal Form

Translation of a sentence (in prenex normal form) to clausal form goes as follows:

1. Replace $\Longrightarrow$, $\Longleftrightarrow$ with the basic connectives ($\wedge, \vee, \neg$).
2. Negation In (negation only in front of atomic formulas).
3. Make all quantified variables names distinct (Standardize Variables).
4. Existentials out (Skolemization (with either constant object symbols, or constant function symbols)).
5. Universals out.
6. Disjunctions in.
7. Connectives out.
8. Rename Variables (so that the same variable does not show in two different clauses).

Resolution Strategies

Resolution strategies are algorithms or guidelines for the application of the resolution rule in automated deduction. There are many resolution strategies. First, we list some of the prominent ones that restrict the space of possible resolutions. These are called *Restriction Strategies*:

Unit Resolution Every resolution step uses at most one non-unit clause. This strategy is refutation complete for Horn Clauses, and clause sets for which refutation trees can be constructed (See 4.4,4.5 in (Gabbay et al., 1993a), Chapter 4., and (Wos et al. 1964)). This class of clause sets is called *Unit refutable clause sets*.

Input Resolution Here, every resolution uses at most one resolvent (the other/s are from the initial clause set). Refutation Complete for the class of Unit refutable.

Linear Resolution This strategy also permits parents where one of them is an ancestor of the other (on top of the pairs allowed in Input Resolution). It is Refutation Complete. (Loveland 1970).

Semantic Resolution One splits the set of clauses to two, given an interpretation: The first subset includes clauses that are satisfied by this interpretation, and the other subset includes those clauses that are not satisfied by the interpretation. We then disallow resolutions within the same set. It is refutation complete (Slagle, 1967).

Set-of-Support A special case of Semantic Resolution, where the first set is the hypothesis set, and the second is the clause of the negated conclusion (this set is called the *support*). Resolvents are added to the *support* set. This strategy is refutation complete (Wos et al., 1965).

Lock Resolution This strategy gives numbering to literals in clauses, so that an order is kept. The numbers are then inherited during the resolution process, and the resolution tries to keep the indexes of the resolved literals as low as possible. It is refutation complete (Boyer, 1971).

Clause-Ordered Resolution Every clause is considered an ordered set, and resolution is permitted only between the first elements of the sets. It is refutation complete for horn clauses, and can be made refutation complete for the general case if we resolve two clauses twice - once putting the remaining literals from the positive parent first, and another putting the remaining literals from the negative parent first.

Directed Resolution A restriction of Clause-Ordered resolution to Horn clauses in which the positive literal occurs either at the beginning or at the end of the clause. A *forward clause* is a clause where the positive literal shows at the end. It gives rise to *forward resolution* (positive conclusions are derived from positive data). A *backward clause* is a clause where the positive literal shows at the beginning. It gives rise to *backward resolution* (negative conclusions are derived from negative data).

Ordering Strategies put some ordering constraints on the clauses resolved. They are mostly fair (if not exhaustive) in the sense that they never postpone any step relative to infinitely many others.

Level Saturation Each clause is associated with its 'depth', i.e., initial clauses have depth 0, factors[3] are of the same depth as their parents, and resolvents are of depth $d + 1$, where d is the maximum of the depth of the parents. The resolution then prefers resolving clauses with lower depth (i.e., Breadth-First-Search style).

Unit Preference Same as *Level Saturation*, but with a constant c subtracted from the level of resolvents with a parent that is a unit clause (this way unit clauses occur c levels earlier).

Deletion Strategies These strategies reduce (and sometimes solve) the clause set under consideration according to some important rules:

Predicate Cancellation (also known as *pure predicate elimination*) Cancels pure literals (those that can not be resolved with anyone else), and then continue with another method (if you did not get any conclusion).

Tautology Elimination If there is a tautological clause (or clauses), we remove them.

Subsumption Elimination If a clause is subsumed by another clause[4], then the subsumed clause is removed.

Reasoning with equality

Equality can be dealt with by adding the appropriate equality axioms: Equivalence Relation axioms (Reflexivity, Symmetry, Transitivity), and Substitution (both for functions and relations). The clausal form of these axioms can be used as usual with Resolution. The resulting search space is intolerably large, even for small problems.

The same way that John Alan Robinson generalized the modus ponens rule to the resolution rule using unification, George A. Robinson and Lawrence Wos generalized the Leibniz principle ('replace equals by equals' rule) to the so called *paramodulation rule* (Robinson and Wos 1969).

[3] Factors are instantiations (replacement of variables by terms) of clauses.

[4] A clause Ψ is *subsumed* by a clause Φ iff there is a substitution σ such that $\Phi\sigma \subset \Psi$.

Definition B.1.16 *Let σ be the mgu of l and a term occurrence t in L. The Paramodulation rule is*

$$\begin{array}{l} L, K_1, ..., K_k \\ l = r, M_1, ..., M_m \\ - - - - - - - - - - - - - - - - - - - \\ \sigma L', \sigma K_1, ..., \sigma K_k, \sigma M_1, ..., \sigma M_m \end{array}$$

Where L' is obtained by replacing t in L by r.

Notice that in Paramodulation the clause containing the equation can also contain additional literals, and that the two terms involved in the replacement need only be unifiable.

The extension of the resolution calculus by the paramodulation rule and the *reflexivity axiom* $\forall x\ x = x$ is called the *RP-calculus* and is refutation complete for first-order predicate logic with equality.

EXAMPLE

$$\begin{array}{l} \{P(f(x)), Q(g(x))\} \\ \{f(a) = c, R(b)\} \\ - \\ \{P(c), Q(g(a)), R(b)\} \end{array}$$

B.2 Nonmonotonic Reasoning Systems in Sections 2.5,6.3.2,6.4

Since its debut, the nonmonotonic reasoning line of work has expanded and several textbooks now exist that give a fair view of nonmonotonic reasoning and its uses (e.g., (Brewka, 1991; Antoniou, 1997; Brewka et al., 1997; Gabbay et al., 1993c; Sandewall, 1994; Shanahan, 1997)). The motivations for nonmonotonic reasoning vary from formalizing Common Sense reasoning through Elaboration Tolerance and representation of uncertainty to Belief Revision. See (Shanahan, 1997; McCarthy, 1998; Pearl, 1990; Antoniou, 1997) for further details in these directions. In this section we bring only the background of two nonmonotonic systems that are used in this thesis: *negation-as-failure* and *circumscription*.

B.2.1 Negation As Failure and Closed World Assumptions

The notion of *Negation as Failure to Prove* (NAF) refers to any of a set of variants on the idea that if something is not proved, it can be assumed false. This set of variants includes *Naive NAF* (what

was originally called Negation as Failure) and the various forms of the Closed World Assumption (described below). Referred to by itself, NAF refers to what we call here Naive NAF.

Originally, the notion of NAF was studied by Clark (Clark, 1978), trying to formalize the primal form of non-monotonic reasoning that was put into extensive use in the form of logic programming languages such as PLANNER (Hewitt, 1972) and PROLOG (Cohen, 1988), (Sterling and Shapiro, 1994) (in fact, these programming languages used an approximation called "negation as finite failure").

Now days, the notion of *Naive Negation as Failure* is mostly used to denote the practical method that treats failure to prove a statement as proving the negation of the statement. I.e., using Negation As Failure for φ in T, we can assume $\neg\varphi$ if $T \nvdash \varphi$. If not used with caution, NAF can be harmful.

Example B.2.1 ((Reiter, 1978)) *Suppose that $T = Teach(pascal) \vee Teach(prolog)$. The naive NAF rule applied to T allows us to assume $\neg Teach(pascal)$ and $\neg Teach(prolog)$ and therefore leads to contradiction.*

Reiter (Reiter, 1978) studied the notion of (Closed World Assumption (see below)), motivated mostly by the theory of closure in Databases. Whereas Negation as Failure is defined for all sentences φ, Reiter (Reiter, 1978) defined a refined version called the *Closed World Assumption* (CWA) that works similarly, but restricts the NAF rule to apply only for ground atoms.

Definition B.2.2 (Closed World Assumption) *The* Closed World Assumption with respect to P *is the assumption that the positive information existing in our theory T about predicates from P is complete, i.e., no other positive information about P is available.*

More formally,

$$CWA(T) = T + \{\neg p \mid p \text{ is a ground atom such that } T \nvdash p\}$$

Theorem B.2.3 ((Reiter, 1978)) *For a theory T consisting of Horn clauses*[5] *the following holds:*

1. *If T is consistent, then $CWA(T)$ is consistent.*
2. *If T is consistent, then for any ground formula f we have $T \nvdash f \iff CWA(T) \vdash \neg f$.*

In (Van Emden and Kowalski, 1976) a semantical characterization of $CWA(T)$ was given in terms of minimal Herbrand models of T. Shepherdson (Shepherdson, 1984) generalizes Reiter's

[5]Horn clauses are disjunctive clauses with at most one positive literal.

consistency result to definite theories (a *definite theory* T is a theory for which for any ground disjunction $D = p_1 \vee ... \vee p_n$, $T \vdash D$ iff $T \vdash p_i$ for some i) and shows that for Horn theories the resolution used in PROLOG is sound with respect to $CWA(T)$.

Shepherdson (Shepherdson, 1984) compares the two notions of NAF and CWA.

Predicate Completion

Predicate Completion (Clark, 1978),(Kowalski, 1978) is a device for "closing off" a first order representation. This concept stems from the observation that frequently a world description provides sufficient, but not necessary, conditions on one or more of its predicates and hence is an incomplete description of that world.

After aggregating all the sufficient information for $P(x)$ in T into a formula $\forall x.\varphi(x) \Rightarrow P(x)$, Predicate Completion for P adds the *completion axiom* $\forall x.P(x) \Rightarrow \varphi(x)$ to T.

For Horn Theories, CWA defined above is enough to produce the needed Predicate Completion. For more complicated cases, Predicate Completion may not be that intuitive any more (for example, what should happen if there is no finite one prime-implicant as a sufficient condition for P?). Nevertheless, if our theory is equivalent to $\forall x.\mathcal{F}(x) \Rightarrow P(x)$, where the formula F does not contain an instance of P, Circumscription and some of the Negation as Failure variants that we describe below are capable of producing the right completion axiom.

Theorem B.2.4 ((Reiter, 1982),(Lifschitz, 1985)) *If $F(x)$ does not contain P, then the circumscription*

$$Circ[\forall x(F(x) \Rightarrow P(x)); P]$$

is equivalent to $\forall x(F(x) \equiv P(x))$.

Generalized Closed World Assumption

Minker (Minker, 1982) generalized the Closed World Assumption to be theories that are not Horn.

Definition B.2.5 (Generalized Closed World Assumption) *The Generalized Closed World Assumption applied to a theory T results in the closure $GCWA(T)$ of T defined as follows:*

$$GCWA(T) = T + \left\{ \neg p \;\middle|\; \begin{array}{l} p \text{ is a ground atom and there is no ground} \\ \text{disjunction } B \text{ of atoms such that } T \vdash p \vee B \\ \text{and } T \nvdash B \end{array} \right\}$$

GCWA allows us to construct query answering systems capable of handling common-sense situations as the one discussed in example B.2.1 on page 187. Such a system will give us a negative answer to a query $Teach(KR)$ but it will reply "maybe" to a query $Teach(pascal)$. Unlike the NAF, Minker's rule works only for literals. Yahya and Henschen (Yahya and Henschen, 1985) extended Minker's system and results to allow application of Minker's GCWA to arbitrary formulae.

Extended Closed World Assumption

Both CWA and GCWA do not allow us to prove any new positive facts.

Theorem B.2.6 ((Minker, 1982)) *For any positive ground formula* f

$$GCWA(T) \vdash f \iff T \vdash f$$

For example, if we wish to conclude $flies(Tweety)$ from $\{bird(Tweety),\ bird(x) \wedge \neg ab(x) \Rightarrow flies(x)\}$, neither of them is helpful. An extension of Minker's GCWA is *Careful Closed World Assumption* (CCWA) proposed by Gelfond and Przymusinska (Gelfond and Przymusinska, 1986) to counter this problem. Later, Gelfond, Przymusinska and Przymusinska and Przymusinski (Gelfond et al., 1986),(Gelfond et al., 1989) devised a generalization of CCWA that allows formulas other than ground literals to be "closed" (CCWA allowed application of the closure to ground literals only). This version is called Extended Closed World Assumption (ECWA) and we bring it here.

Given a list R of predicates, R^+ [R^-] will denote the set of all positive [negative] ground literals with predicate symbols in R. In the following, T is a theory and P, Z are sets of constant symbols (P is a set of *predicate* constant symbols) as in Circumscription (section B.2.2). Let Q be the list of all remaining predicate symbols from T.

Definition B.2.7 *Let* K *be an arbitrary sentence* not *involving predicate symbols from* Z. *We will say that* K *is* free for negation in (T, P, Z) *if there exists* no *disjunction* φ *of ground literals from* $P^+ \cup Q^+ \cup Q^-$, *such that*

1. $T \vdash K \vee \varphi$
2. $T \nvdash \varphi$

Roughly speaking, a sentence is free for negation if it will not be forced to be positive after we make some closed world assumption about P.

Definition B.2.8 (Extended Closed World Assumption) *An* Extended Closed World Assumption *with respect to P and Z applied to the theory T results in the closure*

$$ECWA(T; P; Z) = T + \{\neg K \mid K \text{ is free for negation in } T\}$$

Two of the most common nonmonotonic assumptions used in commonsense theories.

DCA (Domain Closure Axioms)

$$\forall x.(x = t_1 \vee ... \vee x = t_n)$$

This axiom entails that the domain is always finite.

UNA (Unique Names Axioms)

$$\forall i \neq j.(t_i \neq t_j)$$

This axiom entails that all of the symbols refer to distinct objects in the domain.

Together with the DCA, UNA implies the exact cardinality of the domain.

Theorem B.2.9 *Assume that T is a first-order theory over a language $\mathcal{L}$ with a finite number of individual constant symbols and no function symbols. The theory T is also assumed to include equality axioms and the Domain Closure Assumption (DCA) and the Unique Name Assumption (UNA) axioms for the constant symbols in $\mathcal{L}$.*

Then, for any sentence $\varphi \in \mathcal{L}$,

$$Circ[T; P; Z] \models \varphi \iff ECWA(T; P; Z) \vdash \varphi$$

Other Negation-As-Failure Notions

A further extension of ECWA that we do not consider here for lack of space is Iterated Closed World Assumption (Gelfond et al., 1989) (ICWA). Roughly speaking, it applies ECWA to the less prioritized predicates first ("varying" those predicates with even lower priority), and then applies ECWA in a recursive fashion on its way up to the predicate with the highest priority. Thus, it covers the case of priorities between predicates and a correspondence to Prioritized Circumscription, similar to the one mentioned in the last theorem, is shown in (Gelfond et al., 1989).

Figure B.1 on the next page gives a summary of the development of the systems we described above.

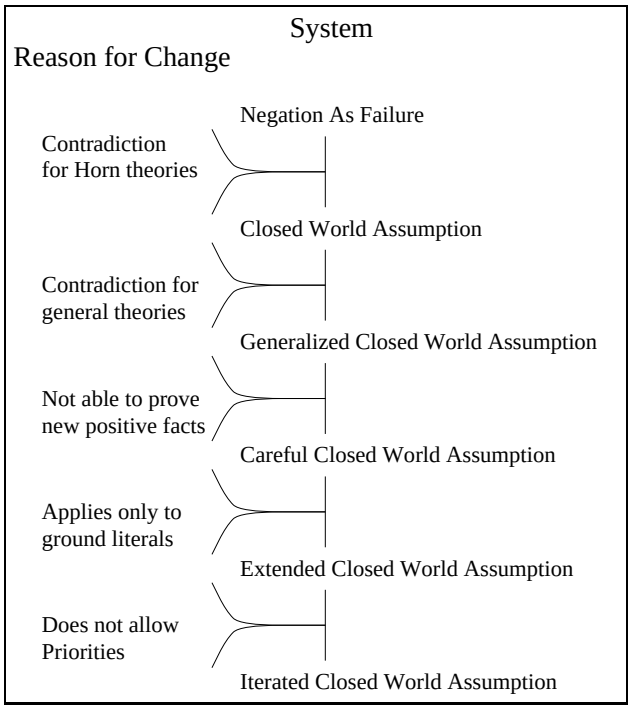

Figure B.1: Progress through the different variants of Negation as Failure

Other works on Negation as Failure that we do not consider here are works on delivering a variant of Negation As Failure that is "safe" (as demonstrated above) but also has nice computational properties (the versions described above (especially ECWA and its variants) are not easily computable). The work of Geffner (Geffner, 1991) tries to define such a notion, and discusses different variants.

Others that studied variants of NAF include Naqvi (Naqvi, 1986) and Rajasekar et al (Rajasekar et al., 1989).

B.2.2 Circumscription

McCarthy's Circumscription (McCarthy, 1980) is one of the first and major nonmonotonic reasoning tools. The major idea behind the family of circumscription formulas is that nonmonotonicity of reasoning can be achieved by making the reasoning in two steps: first, add some new facts to your original theory (facts that will depend on your original theory) and only then make deductions using these new facts and your original theory. These *new facts* are frequently of a second-order logic nature (for the first-order original theory case).

Nonmonotonicity is achieved here by the differences between the added sentences to two different theories. This family of nonmonotonic operators is of particular interest in that it does not introduce a new logic but rather use classical second-order logic (or first-order in some cases) to account for the nonmonotonic entailment.

One way to define an entailment relation using this family is, $\Gamma \mid\sim_{C(P,Z)} \varphi$ iff $\Gamma \wedge \Psi(\Gamma, P, Z) \models \varphi$, for $\Psi(\Gamma, P, Z)$ a second-order formula (given a first-order theory Γ) depending on the form of circumscription chosen, and P, Z vectors of predicate or function constant symbols.

McCarthy's Circumscription

McCarthy's Circumscription formula (McCarthy, 1980)

$$Circ[A(P,Z); P; Z] \;=\; A(P,Z) \wedge \forall p, z\, [A(p,z) \Rightarrow \neg(p < P)] \tag{B.1}$$

says that in the theory A, with parameter relations and function sequences P, Z, P is a minimal element such that $A(P,Z)$ is still consistent, when we are allowed to vary Z in order to allow P to become smaller.

Notice that $p < P$ stands for the conjunction of $\forall \hat{x}.(p(\hat{x}) \Rightarrow P(\hat{x}))$ (p is no larger than P) and $\exists \hat{x}.(\neg p(\hat{x}) \wedge P(\hat{x}))$ (p is strictly smaller than P).

Take for example the following simple theory:

$$T \;\equiv\; block(B_1) \wedge block(B_2)$$

Then, the circumscription of $block$ in T, varying nothing, is $Circ[T; block;] = T \wedge \forall p\, [T_{[block/p]} \Rightarrow \neg(p < block)]$ which is equivalent to

$$Circ[T; block;] \equiv \forall x\, (block(x) \Leftrightarrow (x = B_1 \vee x = B_2))$$

By minimizing $block$ we concluded that there are no other blocks in the world other than those mentioned in the original theory T.

Historical notes The formula presented above is in fact the result of a progression of papers.

In (McCarthy, 1977) a formula called *domain circumscription* was presented (there called *minimal entailment*) with the intuition that we try to make the domains of our models as small as

possible:

$$Axiom(\Phi) \wedge A^{\Phi} \wedge \forall x.\Phi(x)$$

$Axiom(\Phi)$ stands for the conjunction of all $\Phi(a)$'s, for every object constant symbol a together with $\forall x.(\Phi(x) \Rightarrow \Phi(f(x)))$, for every function symbol f (and similarly for predicate symbols). A^{Φ} is the relativization of A to the domain defined by Φ, written as the axiom that is achieved from A by rewriting "$\forall x$" into "$\forall x.\Phi(x) \Rightarrow$" and "$\exists x$" into "$\exists x.\Phi(x) \wedge$".

As a second step, (McCarthy, 1980) introduced a more intuitive formula, similar to the one we mentioned at the beginning of this section, in which there were no varied constant symbols mentioned:

$$A(\Phi) \wedge [(\Phi \leq P) \Rightarrow (\Phi = P)]$$

Notice that this is in fact a schema (on the formula Φ) while (B.1) is a second-order formula. Properties of the first-order schema were studied in (Perlis and Minker, 1986). In (McCarthy, 1980) it was also shown that domain circumscription is reducible (under a simple translation) to predicate circumscription.

In response to some criticism given by (Etherington et al., 1985) and others, a slightly different formula was given by McCarthy in (McCarthy, 1986b), a formula which he calls *formula circumscription*, which he distinguishes from the less-general previous formula which he calls *predicate circumscription* in two ways: the new one is a second order formula, and the minimized formula is an arbitrary wff whereas the original version minimizes a single predicate. For P being a tuple of some predicate symbols, the *formula circumscription* of the wff $E(P, x)$ in $A(P)$ is

$$A(P) \wedge \forall P'.[A(P') \wedge [\forall x.E(P', x) \Rightarrow E(P, x)] \Rightarrow [\forall x.E(P', x) \equiv E(P, x)]]. \qquad \text{(B.2)}$$

This formula, although not mentioning the varied predicates, is in fact equivalent to predicate circumscription with varied constant symbols, which is the formula (B.1) in the opening of this section. Because of this equivalence, we will continue considering only (B.1) in the rest of this thesis.

Parallel and Prioritized Circumscription

Prioritized Circumscription was introduced by Lifschitz and McCarthy in (Lifschitz, 1985) and (McCarthy, 1986b) to be a simple generalization over the Circumscription idea. Instead of minimizing only one specific predicate, we may want to minimize several predicates, allowing for some priorities among them.

The example below describes an inheritance hierarchy in which every bird is a thing and every penguin is a bird. Things normally do not fly, birds normally do and penguins normally do not.

$$\left.\begin{array}{l} thing \wedge \neg Ab_1 \Rightarrow \neg fly \\ bird \wedge \neg Ab_2 \Rightarrow fly \\ penguin \wedge \neg Ab_3 \Rightarrow \neg fly \end{array}\right\} \text{Deliberated properties}$$
$$\left.\begin{array}{l} bird \Rightarrow thing \\ penguin \Rightarrow bird \end{array}\right\} \text{Hierarchy definition} \tag{B.3}$$

In this theory we would like to say that all the abnormalities Ab_1, Ab_2, Ab_3 are normally FALSE, but that Ab_3 takes priority over Ab_2 and Ab_2 takes priority over Ab_1. This is done by lexicographically saying that, for $ab = \langle ab_1, ab_2, ab_3 \rangle$ and $Ab = \langle Ab_1, Ab_2, Ab_3 \rangle$,

$$\begin{array}{lll} ab \leq Ab \equiv [& (ab_3 \leq Ab_3) \wedge & \\ & (ab_3 = Ab_3 \Rightarrow [& (ab_2 \leq Ab_2) \wedge \\ & & (ab_2 = Ab_2 \Rightarrow [(ab_1 \leq Ab_1)])])] \end{array}$$

The use of this order $ab \leq Ab \wedge ab \neq Ab$ instead of $p < P$ in our circumscription formula (B.1) then yields the intended solution (i.e., that a penguin does not fly).

A generalization of this idea is the following (here we follow the account given in (Lifschitz, 1993)). Let $\mathcal{P}$ be a tuple of predicate constants that is broken into disjoint sets $\mathcal{P}_1, ..., \mathcal{P}_n$. The predicates in each $\mathcal{P}_i$ have equal priority and are therefore minimized concurrently. They also all have more priority than those in $\mathcal{P}_j$ for $j > i$. This is formulated by the following order relations:

$$\begin{array}{rcl} \langle P_1, ..., P_k \rangle & \leq & \langle Q_1, ..., Q_k \rangle \equiv P_1 \leq Q_1 \wedge ... \wedge P_k \leq Q_k \\ \mathcal{P} \preccurlyeq \mathcal{Q} & \equiv & \bigwedge_{i=1}^{n} \left(\bigwedge_{j=1}^{i-1} (\mathcal{P}_j = \mathcal{Q}_j) \Rightarrow (\mathcal{P}_i \leq \mathcal{Q}_i) \right) \\ \mathcal{P} \prec \mathcal{Q} & \equiv & (\mathcal{P} \preccurlyeq \mathcal{Q}) \wedge \neg (\mathcal{P} = \mathcal{Q}) \end{array} \tag{B.4}$$

The *prioritized circumscription* $Circ[A; \mathcal{P}_1, ..., \mathcal{P}_n; Z]$ is then

$$A(\mathcal{P}, Z) \wedge \forall pz[A(p, z) \Rightarrow \neg (p \prec \mathcal{P})] \tag{B.5}$$

Fortunately, there is a simple theorem that relates prioritized circumscription with predicate circumscription.

Theorem B.2.10 *The circumscription $Circ[A; \mathcal{P}_1, ..., \mathcal{P}_n; Z]$ is equivalent to*

$$\bigwedge_{i=1}^{n} Circ[A; \mathcal{P}_i; \mathcal{P}_{i+1}, ..., \mathcal{P}_n, Z].$$

Finally, the reader should notice that prioritized circumscription does not treat the cases in which we would like to put priorities that are not in the order format mentioned above. For example, arbitrary partial orders are not treated (see also (Amir, 1998)). A generalization of prioritization was suggested by Grosof (Grosof, 1991) to the case of any well-founded strict partial order, yielding a revised version of Prioritized Circumscription.

Semantics for Circumscription

Given by Lifschitz (Lifschitz, 1985), McCarthy (McCarthy, 1986b) and Etherington (Etherington, 1986), the preferential semantics for Circumscription was the first such semantics discovered.

Definition B.2.11 ((Lifschitz, 1985)) *For any two models $\mathcal{M}, \mathcal{N}$ of a theory T we write $\mathcal{M} \leq_{P,Z} \mathcal{N}$ if the models $\mathcal{M}, \mathcal{N}$ differ only in how they interpret predicates from P and Z and if the extension of every predicate from P in M is a subset of its extension in $\mathcal{N}$.*

The relation $\leq_{P,Z}$ is a pre-order and hence we can talk about minimal models $\mathcal{M}$ with respect to $\leq_{P,Z}$ in the class of all models of T. More precisely, we say that a model $\mathcal{M}$ of T is (P, Z)*-minimal* if there is no model $\mathcal{N}$ such that $\mathcal{N} <_{P,Z} \mathcal{M}$.

Theorem B.2.12 ((Lifschitz, 1985): Semantics for Circumscription) *A structure $\mathcal{M}$ is a model of $Circ[T; P; Z]$ iff $\mathcal{M}$ is a (P, Z)-minimal model of T.*

A similar theorem is given with respect to the semantics for Prioritized Circumscription.

Theorem B.2.13 ((Lifschitz, 1985): Semantics for Prioritized Circumscription) *A structure $\mathcal{M}$ is a model of the prioritized circumscription $Circ[T; P_1 > ... > P_n; Z]$ iff $\mathcal{M}$ is a model of T such that for every $i \leq n$, $\mathcal{M}$ is $(P_i, \{P_{i+1}, ..., P_n, Z\})$-minimal model of T.*

Computing Circumscription

Roughly speaking, there are two kinds of computation systems for circumscription. The first is an equivalent transformation system of circumscription into first-order sentences ((Doherty et al.,

1997), (Gabbay and Ohlbach, 1992), (Kolaitis and Papadimitriou, 1990), (Lifschitz, 1985), (Rabinov, 1989), (Cadoli et al., 1992)) and logic programs ((Gelfond and Lifschitz, 1989), (Przymusinski, 1989)). The second is a deductive computation system for circumscription ((Ginsberg, 1989), (Przymusinski, 1989)).

The following theorem allows us to remove the varied constant symbols from the circumscription formula, in price of incrementing the order of the formula (i.e., the formula may become second-order if it was originally first-order).

Theorem B.2.14 ((Lifschitz, 1993)) *Let A be a set of sentences in $\mathcal{L}$, P a sequence of predicate constant symbols in $\mathcal{L}$ and Z a sequence of constant symbols in $\mathcal{L}$. Then,*

$$Circ[A(P,Z);P;Z] \equiv Circ[\exists z.A(P,z);P;]$$

Another interesting theorem allows us to reduce the circumscription formula to a slightly less complicated circumscription formula.

Theorem B.2.15 $Circ[A \wedge B;P;Z] \equiv Circ[A;P;Z] \wedge B$ *if P, Z do not appear in B.*

Other works on reducing Circumscription to FOL include the works of (Lifschitz, 1986b) and (Mott, 1987) on the satisfiability of circumscription and Cadoli et al (Cadoli and Schaerf, 1992) on approximate inference for circumscription and default logic. Also, Ginsberg (Ginsberg, 1989) used an ATMS to supply a circumscriptive theorem prover. Finally, Cadoli (Cadoli, 1992) wrote about the complexity of model checking for circumscriptive formulae.

Summary

Diagram B.2 on the facing page describes the development of Circumscription variants we discussed above and the reasons behind them.

B.2.3 Progression of Nonmonotonic Systems

Diagram B.3 on page 198 is an illustration of the development of Nonmonotonic systems. Circumscription, Nonmonotonic Logic and Default Logic are shown as *attractors* in a space of nonmonotonic systems.

Some comparison works between the different nonmonotonic systems include (Gelfond and Lifschitz, 1989; Minker et al., 1991) (Circumscription and Logic Programs), (Gelfond et al., 1989) (Circumscription, Prioritized Circ, CWA, GCWA, ECWA, Iterated CWA), (Lifschitz, 1989; Michael

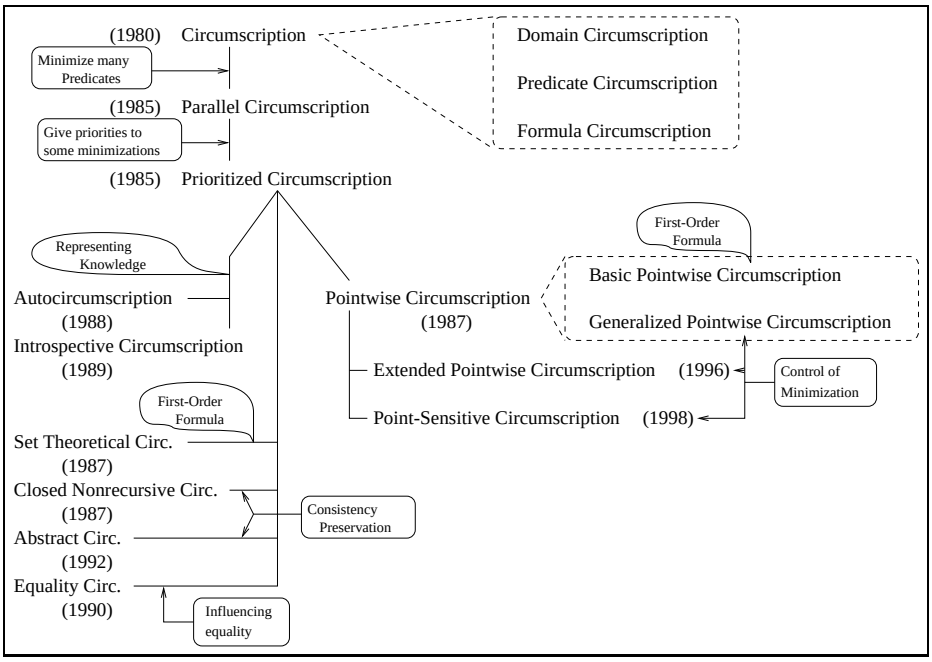

Figure B.2: Circumscription Systems

Gelfond, Vladimir Lifschitz and Schwarz, 1994) (Introspective Circumscription and Autoepistemic Logic), (Shepherdson, 1984) (Closed-World Assumption (CWA) and Negation-As-Failure (NAF)), (Bondarenko et al., 1997) (Logic Programming, Autoepistemic Logic (AEL) and Default Logic (DL)) and (Yuan and You, 1993) (Autocircumscription and Logic Programs).

This account does not take into consideration works done on the definition of nonmonotonic entailment relations. The reader interested in those can look at (Makinson, 1993).

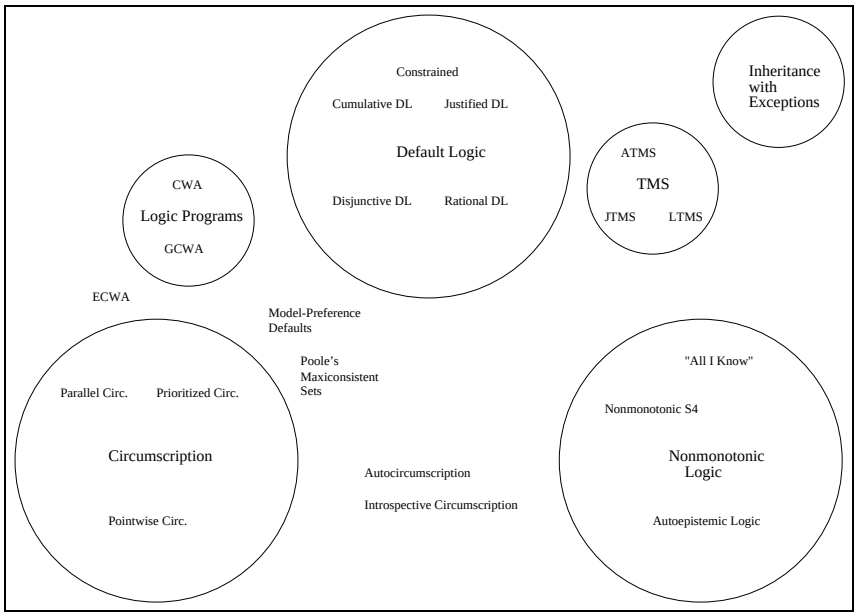

Figure B.3: Nonmonotonic Reasoning Systems.

Appendix C

More on Reactive Systems and Robot Control

This chapter presents some agent architectures, robot control issues and reactive system architectures that can be related to the LSA architecture presented in Chapter 6. This is not meant to be a comprehensive list or even a good summary of related work in those fields. Instead, it is intended to give some more general background for the work reported in Chapter 6. The discussion provided in this chapter is not required to understand the material in Chapter 6, and is provided here only for a more complete picture.

C.1 AI Architectures

Typically, an *AI architecture* combines a set of individual AI techniques into a larger system in a principled way. The set of components used and the way in which those components interact is the main contribution of an architecture. Sometimes an architecture would be designed for a set of systems that solve a family of problems (e.g., diagnosis of patients by a medical doctor). Other systems are intended to achieve certain features of human intelligence.

This section discusses three examples of AI architectures that influenced this thesis.

C.1.1 SOAR

SOAR (Laird et al., 1986) is a cognitive architecture built by Newell, Laird and Rosenbloom (all at CMU at the time). The original system appeared in 1982, and development has continued to

the present. SOAR was built borrowing some ideas from GPS (Ernst and Newell, 1969) and on a production system shell called OPS5 (Newell, 1973).

SOAR uses Chunking/EBL (Explanation-Based Learning) for its deductive learning, and is rule-based. It also uses several levels of memory such as long-term, short-term and chunked-rules.

In any situation, SOAR makes a selection in one of four areas: The goal space (which goal to attack next), the problem space (choosing a problem space for this goal involves, for example, choosing plan space versus state space in planning), the state space (deciding what is a state in that problem space), and the state to continue from (choosing the operator to use on that state). Making the choice in each stage relies on a heuristic that is explicitly represented in the long-term memory by ways of preference of one choice over another. If there is no such preference a new goal of finding such a preference is added to the list of active goals and the process begins again.

Each time a set of rules is applicable to a state, all the rules are fired at once, and the results are collected and applied to the current situation. Another way to deal with several applicable rules is by *conflict resolution* trying to estimate which rule is the most *profitable* in the current situation. After a goal is achieved, chunking of the operators and rules used is done, and the chunked rules are stored in the chunked-rules memory.

The system is supplied with several sets of rules and goals that serve as the defaults for the system. Each set of rules leads to a different search behavior. These defaults are there just in case the user did not supply enough information for a more developed search technique or some domain-specific knowledge to apply.

C.1.2 CYC

In development by Lenat, Guha and others since 1982 (Lenat and Guha, 1990), CYC is an architecture that tries to create a *critical mass* of reusable knowledge for natural language understanding and learning. The critical mass here is the set of knowledge needed for the learning agent to be able to acquire knowledge by itself from literature written in natural language.

The architecture embodies two levels of knowledge description: the logical level, where all the knowledge is written in a superset of FOL, and a heuristic level, where some bodies of knowledge are more efficiently reasoned about. Only the first one is actually available to the knowledge-enterer, and the program is in charge of translating from one representation to another. The system can reason in both languages, and prefers specific reasoning methods for certain bodies of knowledge to more general deduction.

Overall, CYC has a large database of concepts and axioms, put in hierarchies similar to Frame

systems or semantic nets, and many ontologies are represented. This structure is realized using contexts: the axioms are put in micro-theories that are associated with contexts, and the contexts are related to each other by various links (e.g., one context may specialize another one). When the system is asked to answer queries several contexts may be used (currently much of this is controlled by the user submitting the query). The logic used in CYC has 5-value truth assignments, using abnormality theory and a reduction of circumscription (to be more tractable in price of less precision) to account for defaults. The five values are TRUE, default-true, unknown, default-false, FALSE.

More recently (Lenat, 1995), the CYC project has extended the system to have contexts that lie in a Cartesian space of ten features. This allows a knowledge engineer to specify the context in which the axioms are entered more easily and with less regard to the relationship to other contexts and axioms in the system. The system now has more than 1.5 million axioms, and it is hoped that it will reach more than 10 million in five or ten years.

C.1.3 Blackboard Systems

Blackboard architectures (Nii, 1986) use a central data structure called the *blackboard* and a set of *knowledge sources*. Every *knowledge source* has access to the blackboard, and no communication is possible between the knowledge sources.

This architecture is very useful when dealing with problems to which there is no exact algorithmic solution, and on which we know several separate things that can give us some clues on the way to the solution, but they all interact. Mostly the blackboard will have several levels, each dealing with a different aspect of the problem and data at hand.

Two examples can explain the virtues of the architecture and the different paradigms dominating it: HEARSAY-II and HASP.

HEARSAY-II

HEARSAY-II (Erman et al., 1980) is a system for understanding spoken language, built at CMU as part of the ARPA SUR project. The blackboard in HEARSAY-II is divided into 7 levels for different aspects of the spoken language, some of which are: phonemes, syllables, words, and phrases. Each of the knowledge sources receives its input from one level, and emits output on some level (possibly the same one). Correspondingly, each knowledge source applies its knowledge to different aspects of the input, such as the way phonemes add to create syllables and words.

The blackboard includes *islands*, each corresponding to a partial parse of the input. At each

stage in the run of the system over some input, a knowledge source is selected by a central component. This chooser decided on the selected source according to the proposed improvement in the board. This improvement is specified in a rule-based component of each knowledge source. The selected knowledge source expands one of of the islands in an island-driving-search pattern. One of these islands will eventually expand over the rest, eliminating unused fragments on the way, and the solution will be that expanded island.

HEARSAY-II was successfully applied to a travel agent domain and had a successful performance.

HASP

HASP (Nii et al., 1982) uses a different paradigm than HEARSAY-II. Instead of maintaining islands, it maintained a *current best hypothesis* (CBH) on the blackboard, allowing any new information to lead to a change in that hypothesis. The domain for HASP was the detection of submarines (the submarines were called *platforms*) using inputs from instruments and intelligence.

The different sources of input emphasized the way blackboards facilitate opportunistic reasoning, since some knowledge sources used knowledge in the upper levels (the intelligence sources) to create knowledge in lower levels (expected patterns in time X) and vice versa.

An interesting aspect of HASP is that it could explain its choice for CBH. Another important aspect is its ability to put predictions on the blackboard, predictions that if falsified would cause the system to look for a new hypothesis.

C.2 Reactive Systems

Reactive systems deal with situations in which there is a need to reason in a time-dependent fashion. The various issues include architectures, algorithms for time-dependent planning, and adjustments of techniques in AI to robots engaged in real-time activities.

Other systems that are relevant to this subject but are not discussed here are Agre and Chapman's Pengi (Agre and Chapman, 1987) and Drummond's *plan nets* and *situated control rules*. We mention the work of Kaelbling and Rosenschein (e.g., (Rosenshein and Kaelbling, 1995) in Section 6.7.

C.2.1 Boddy & Dean – Time-Dependent Planning

Expectation-driven iterative refinement is a method introduced by Boddy and Dean in (Boddy and Dean, 1989). It uses some expectation over the improvement of extra deliberation time, t, to decide whether that extra time will be used for extra planning. It uses an anytime algorithm, and relies on past experience to give expectations about its improvement over time.

An example of an anytime algorithm is the following algorithm for path planning. The simplest plan that we can supply to the robot is an iterative refinement of the space into finer and finer resolution. At first, we supply the robot with a very high-level plan, or a plan that has a relatively poor chance of succeeding (e.g., use a very coarse cell decomposition (see Section C.3.2)). After more time is invested in the planning, we plan with a more refined space, providing the robot with a lower level plan or a detailed plan that has better chances of success, or possibly a proven route to the goal (e.g., use a very detailed cell decomposition).

Different anytime algorithms and different tasks have different *performance profiles* (relating the time spent on solving the problem and the average quality of the generated plan). When such profiles are available, they can be used to decide on the amount of time that should be spent planning (or solving a particular task) before the plan (or solution) is used. For example, if we know that a plan of a certain quality is going to take time t_{exec} to execute, and it would take time t_{plan} to plan, then the total execution time for this plan would be $t_{plan} + t_{exec}$.

Deliberation scheduling deals with the problem of multiple anytime algorithms and a goal of minimizing the total amount of time for the combined plan. For each anytime algorithm we compute or receive its performance profile in advance. We use this profile to allocate the time to the different algorithms. (Boddy and Dean, 1989) developed this approach in the context of path planning and tour improvement.

Expectation-driven iterative refinement is also suitable for problems where we can decompose the problem into several subproblems, on which we can deliberate separately. These cases are particularly attractive for this approach because anytime algorithms and their accurate performance profiles are simpler to obtain for small and isolated problems.

C.2.2 Brooks' Layered Control System

A different framework for reactive systems comes from a more pragmatic point of view. Instead of analyzing properties of anytime algorithms, and trying to work with anytime planners, a *layered*

control system (Brooks, 1986; Brooks, 1991) deals with the architecture underlying a robot (somewhat in the flavor of blackboard systems), where there is no actual planning process involved. The system works according to predetermined finite automata, and it acts in accordance with its sensors. This architecture is called a *subsumption architecture* because each succeeding layer subsumes the operations of the preceding ones. Notice that again, some planning work can be done in advance (before the system is set loose), but now no reasoning is done while the robot acts, just pure reactions (possibly in accordance with the pre-planned plan) to sensors.

The architecture is designed to accommodate the following characteristics of robot motion planning: multiple goals (avoid obstacles, reach target), multiple sensors (sonars, cameras), robustness (to noise and failure of sensors and manipulators, and to changes in the environment), and extensibility (adding more sensors/effectors).

The subsumption architecture is built with layers of competence, where each has its own tasks knowledge and capabilities. Each layer is more simple than the previous paradigm (perception $\rightarrow$ modeling $\rightarrow$ planning $\rightarrow$ task execution $\rightarrow$ motor control), because now each layer just keeps its own simple reactive nature. For example, the *avoid objects* layer reacts to the proximity of objects instead of doing a planning process for a certain path that will avoid those obstacles. The control is achieved by letting the higher levels override lower level commands, but otherwise letting the lower level perform their duties. This way, multiple goals, multiple sensors, robustness and extensibility are achieved easily.

Brooks (Brooks, 1990a) suggested a robot control specification language that enables us to treat each module as a finite state machine with four state types: output (do something), side effect (internal variable change), conditional (according to which a different state is entered), and event dispatch (wait for something from a set (*case*) to happen, and then switch to the right state).

In this language, an output can be used to inhibit some outputs of lower levels, suppress an input to a lower level, and apply some commands to some manipulators (e.g., turn, forward, etc.).

One of the main results from the architecture is that there is no need for a central control module for the mobile robot; the control is divided among the different layers.

C.3 Robot Motion Planning

In this section we review some of the traditional methods used in robot motion planning. Little of this theory has been used explicitly in the creation of our robot, LiSA (described in Chapter 6), but some of it has been used implicitly in decisions made about the content of the different layers. The

reader is referred to (Latombe, 1991; Khatib, 1992) for a more detailed description of basic concepts involved with robot motion. For an architectural-level view, the reader is referred to (Kortenkamp et al., 1998).

C.3.1 What Is a Robot?

Typically, mobile robots are modeled using the concept of a linkage. A *linkage* is a combination of links connected with joints. Each link can be thought of as a one dimensional segment. The simple linkage is a sequence of such links. A linkage can be on a mobile robot, or may have one joint in a static location.

The *work space* of the robot is the space spanned by the robot's links and joints. A simple representation of this space is as a Cartesian space that is a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$. This representation of the space is a Cartesian product. A representation that lends itself to easier manipulation is the configuration space.

The *configuration space* of a robot is a high-dimensional space in which the robot's configuration is represented as a single dot. A position in the configuration space of a robot with l links is described as the parameters describing every link (e.g., an angle). If we are dealing with a mobile robot that can translate freely and can rotate in all directions, the robot has a configuration space of dimension 3, if the robot is on the plane $\mathbb{R}^2$. The configuration space is of dimension 6, if the robot is in $\mathbb{R}^3$.

The configuration space makes it simple to carry on planning, as the robot is represented as a point, and some relatively simple algorithms for path planning can be applied.

The number of *degrees of freedom* n_{dof} determines the dimensionality of the configuration space, in an immobile robot. The Gr obler formula describes this number as

$$n_{dof} \geq n_0 * (n_{link} - 1) - (n_0 - 1) * n_{joint}$$

Typically, computation time for the different algorithms for motion planning is PSpace-Hard in the dimension of the configuration space.

C.3.2 Basic Planning Methods

The basic problem in robot motion planning consists of a given initial location/configuration of the robot, a goal configuration, and a set of obstacles and their positions and dimensions. Given these,

the task is to plan a path in free space from the initial configuration to the goal. The robot is assumed to be free flying (no constraints on its configuration, velocity, and acceleration).

First, the *free space* is defined. This is done by projecting the obstacles in the physical space into the configuration space of the robot. In the basic problem, the free space is just the complement of the space taken by the $\mathcal{C}$-obstacles.

Several common planning methods include:

Complete Algorithms

We say that a planning algorithm is *complete* when it finds a plan whenever there is one, and declares failure otherwise.

Roadmap Road-map techniques build a graph in the free space of the robot configuration space, and find a path in that graph using a search method of any kind. Two example roadmap methods are:

Retraction In 2 and 3 dimensions this is called the *Voronoi diagram*. It is constructed using the centers of maximal balls, which touch at least two distinct points in the border of the free space. The balls are maximal in the sense that any increase in the radius of the ball will make it overlap a C-obstacle (an obstacle in the configuration space).

Visibility Graph The first road-map method, introduced by Nils Nilsson in (Nilsson, 1969) for Shakey. It is based on connecting all the vertices of the polygonal C-obstacles (and the source and destination configurations of the robot), such that these edges are free paths.

Exact cell decomposition The free space is dissected into regions using the vertices of the (polygonal) C-obstacles (useful mostly in 2-dimensional spaces), and a *channel* is then searched using the adjacency relation between the regions. An exact path is then constructed in the regions of the channel. This method is complete, but it creates semi-free paths rather than free paths: some regions may be not completely free of obstacles. If the path produced is only semi-free, other methods are used to transform it into a free path, if possible.

Probabilistic Approaches

Probabilistic methods for path planning are not complete, but they are significantly faster than complete methods, and therefore are applicable to higher dimensionality configuration spaces. Instead

of computing the representation of the free space, a sample of milestones is picked, and checked for being free of obstacles. Those that are left are used to create paths (we connect those that we can easily check for closeness). This resulting graph is called a *probabilistic road-map*. Under some assumptions (ϵ-goodness) we can estimate the number of milestones needed to be examined to be $O(1/\epsilon * log(1/(\epsilon * \alpha)))$ to answer correctly with probability $1 - \alpha$.

Incomplete Algorithms

Some methods are not complete, but can be augmented by a strong search mechanism that ensures asymptotic completeness. We say that the a planner is *asymptotically complete* if it is guaranteed to find a path (if one exists), provided that we take some sufficiently large asymptotic constants for our algorithm. Here are two examples of incomplete methods for path planning.

Hierarchical cell decomposition We create a grid of the configuration space, and map every cell to either of $\{$free, occupied, shared$\}$. Then, we try to find a *channel* using the free cells. If we did not succeed, and there are still some *shared* cells, we divide these cells to smaller cells, and repeat the process with those smaller cells.

Potential Field This method combines an attracting field (attraction to the goal) with a repulsive field (repealed by the C-obstacles), to generate a heuristic function for a search in the graphs created by other methods (such as cell decomposition, with its variants).

C.3.3 Computational Complexity

Almost every planning task is PSPACE-hard in the dimension of the configuration space ($dim(C)$). Free-path planning (for a polyhedral robot, polyhedral obstacles, and given two configurations (start and goal)), planar-linkage ($\mathbb{R}^2$, no obstacles, revolute joints, 1-dim links, the problem of getting to a point in configuration space), and planar-arm ($\mathbb{R}^2$, obstacles, revolute joints, 1-dimensional links, and two given configurations), all are PSPACE-hard($dim(C)$).

If we hold $dim(C)$ fixed, there is an upper bound on the complexity of free-path planning, which is $O(poly(n + d))$ where n, d are the number of constraints on the free space, and the degree of these constraints, respectively.

C.3.4 Reducing the Complexity

Several simplifying heuristics are typically used to reduce the complexity of path planning. A simple one is to project or slice the configuration space, and then plan in the slice (i.e., removing some dimensions from the configuration space, simply by fixing some coordinates). A commonly used heuristic in this class is replacing a joint with the span of the space the link can take (e.g., a disc with the link as radius). Another possibility is simplifying the structure of the robot and obstacles (i.e., reducing the number of constraints and their degree).

A different method that simplifies the planning problem is to split the planning into several stages. This is useful in cases where there are some concentrated cluttered areas that can be identified. We plan the movements with some care in the cluttered areas, while using a more coarse approach in the more open areas.

Bibliography

Abrahamson, K. A. and Fellows, M. R. (1993). Finite automata, bounded treewidth and well-quasiordering. In *Graph structure theory*, number 147 in Contemporary Mathematics, pages 539–564. American Mathematical Society. Proc. Conf. on Graph Minors (Seattle, 1991).

Agre, P. E. and Chapman, D. (1987). Pengi: An implementation of a theory of activity. In *Proc. National Conference on Artificial Intelligence (AAAI '87)*, pages 268–272. AAAI.

Ahuja, R. K. and Orlin, J. B. (1991). Distance directed augmenting path algorithms for maximum flow and parametric maximum flow problems. *Naval Res. Log. Quart.*, 38:413–430.

Akman, V. and Surav, M. (1996). Steps toward formalizing context. *AI Magazine*, 17(3):55–72.

Amir, E. (1997). Machinery for elaborating action – Preliminary Report. In NRAC'97 (workshop on nonmonotonic reasoning about actions in IJCAI'97) and in http://www-formal.stanford.edu/eyal/nmr/machinery-ea.ps.

Amir, E. (1998). Nonmonotonic Reasoning as SAT Search. Unpublished note. Available at http://www-formal.stanford.edu/eyal/nmr/nmr-search.ps.

Amir, E. (1999). Elaboration tolerance of logical theories. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-99); Proceedings of the 11th Conference on Innovative Applications of Artificial Intelligence*, pages 940–940, Menlo Park, Cal. AAAI/MIT Press.

Amir, E. (2000a). (De)Composition of situation calculus theories. In *Proc. National Conference on Artificial Intelligence (AAAI '00)*, pages 456–463. AAAI Press/MIT Press.

Amir, E. (2000b). Towards a formalization of elaboration tolerance: Adding and removing axioms. In *Frontiers of Belief Revision*, pages 147–162. Kluwer Academic Publishers. Also in SARA-98 and in http://www-formal.stanford.edu/eyal/papers/et-def-syntactic.ps.

Amir, E. (2001). Efficient approximation for triangulation of minimum treewidth. In *Proc. Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI '01)*, pages 7–15. Morgan Kaufmann.

Amir, E. and Maynard-Reid II, P. (1999). Logic-based subsumption architecture. In *Proc. Sixteenth International Joint Conference on Artificial Intelligence (IJCAI '99)*, pages 147–152.

Amir, E. and Maynard-Reid II, P. (2001a). LiSA: A robot driven by logical subsumption. In working notes of the CommonSense'01 symposium.

Amir, E. and Maynard-Reid II, P. (2001b). LSA version 2.0. On the LSA project web page: http://www-formal.stanford.edu/eyal/lsa.

Amir, E. and McIlraith, S. (2000). Paritition-based logical reasoning. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conference (KR '2000)*, pages 389–400. Morgan Kaufmann.

Amir, E. and McIlraith, S. (2001a). Partition-based logical reasoning for first-order and propositional theories. *Submitted for publication.*

Amir, E. and McIlraith, S. (2001b). Solving satisfiability using decomposition and the most constrained subproblem. In Kautz, H. and Selman, B., editors, *Electronic Notes in Discrete Mathematics*, volume 9. Elsevier Science Publishers. Proceedings of LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT 2001).

Anderson, D. and Werthimer, D. (2001). home: Search for extraterrestrial intelligence at home.

Anderson, R. and Bledsoe, W. W. (1970). A linear format for resolution with merging and a new technique for establishing completeness. *Journal of the ACM*, 17:525–534.

Antoniou, G. (1997). *Nonmonotonic Reasoning*. MIT Press, Cambridge, Massachusetts.

Arkin, R. C. (1998). *Behavior-based robotics*. MIT Press.

Arnborg, S., Corneil, D. G., and Proskurowski, A. (1987). Complexity of finding embebdings in a K-tree. *SIAM J. Alg. and Discr. Meth.*, 8:277–284.

Arnborg, S., Lagergren, J., and Seese, D. (1991). Problems easy for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340.

Baader, F. and Schulz, K. U. (1992). Unification in the union of disjoint equational theories: Combining decision procedures. In *Proceedings of the 11th International conference on automated deduction*, volume 607 of *LNAI*, pages 50–65. Springer-Verlag.

Baral, C. and Tran, S. C. (1998). Relating theories of actions and reactive control. *Electronic Transactions on Artificial Intelligence (http://www.etaij.org)*, 2(3-4):211–271.

Becker, A., Bar-Yehuda, R., and Geiger, D. (2000). Randomized algorithms for the loop cutset problem. *Journal of Artificial Intelligence Research*, 12:219–234.

Becker, A. and Geiger, D. (1994). Approximate algorithms for the loop cutset problem. In *Proc. Tenth Conference on Uncertainty in Artificial Intelligence (UAI '94)*, pages 60–68. Morgan Kaufmann.

Becker, A. and Geiger, D. (1996). A sufficiently fast algorithm for finding close to optimal junction trees. In *Proc. Twelfth Conference on Uncertainty in Artificial Intelligence (UAI '96)*, pages 81–89. Morgan Kaufmann.

Beth, E. W. (1953). On padoa's method in the theory of definition. *Indag. Math.*, 15:330–339.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford.

Bledsoe, W. W. and Ballantyne, A. M. (1978). Unskolemizing. Technical Report Memo ATP-41, Mathematics Department, University of Texas, Austin.

Boddy, M. and Dean, T. (1989). Solving time-dependent planning problems. *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 2:979–984.

Bodlaender, H. L. (1996). A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317.

Bodlaender, H. L. (1997). Treewidth: Algorithmic techniques and results. In Prívara, I. and Ruzicka, P., editors, *Mathematical Foundations of Computer Science 1997*, volume 1295 of *lncs*, pages 19–36. Springer.

Bodlaender, H. L. (2000). Personal communication.

Bodlaender, H. L., Gilbert, J. R., Hafsteinsson, H., and Kloks, T. (1995). Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255.

Bodlaender, H. L. and Kloks, T. (1991). Better algorithms for the pathwidth and treewidth of graphs. In *Automata, Languages and Programming, 18th International Colloquium*, volume 510 of *LNCS*, pages 544–555. Springer-Verlag.

Bonacina, M. P. and Hsiang, J. (1994). Parallelization of deduction strategies: an analytical study. *Journal of Automated Reasoning*, 13:1–33.

Bonacina, M. P. and Hsiang, J. (1996). On the representation of dynamic search spaces in theorem proving. In Yang, C.-S., editor, *Proceedings of the International Computer Symposium*, pages 85–94.

Bondarenko, A., Dung, P., Kowalski, R., and Toni, F. (1997). An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93(1-2):63–101.

Bonzon, P. E. (1997). A reflective proof system for reasoning in contexts. In *Proc. National Conference on Artificial Intelligence (AAAI '97)*, pages 398–403.

Boppana, R. B. and Sipser, M. (1990). The complexity of finite functions. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science*, volume 1: Algorithms and Complexity, pages 757–804. Elsevier/MIT Press.

Borgida, A. (1996). On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1-2):353–367.

Bossu, G. and Siegel, P. (1985). Saturation, nonmonotonic reasoning and the closed world assumption. *Artificial Intelligence*, 25(1):13–65.

Boutilier, C., Reiter, R., Soutchanski, M., and Thrun, S. (2000). Decision-theoretic, high-level agent programming in the situation calculus. In *Proc. National Conference on Artificial Intelligence (AAAI '00)*, pages 355–362. AAAI Press.

Boyer, R. S. (1971). *Locking: a restriction of resolution*. PhD thesis, Mathematics Department, University of Texas, Austin.

Brachman, R. and Levesque, H. (1985). *Readings in Knowledge Representation*. Morgan Kaufmann.

Brachman, R. J., Fikes, R. E., and Levesque, H. J. (1983). KRYPTON: A functional approach to knowledge representation. *Computer*, 16(10):67–73.

Brewka, G. (1991). *Nonmonotonic Reasoning: Logical Foundations of Common Sense*. Cambridge University Press.

Brewka, G., Dix, J., and Konolige, K. (1997). *Nonmonotonic Reasoning: An Overview*, volume 73 of *CSLI Lecture Notes*. CSLI Publications.

Broersma, H., Kloks, T., Kratsch, D., and Müller, H. (1998). A generalization of AT-free graphs and a generic algorithm for solving treewidth, minimum fill-in and vertex ranking. In *WG: Graph-Theoretic Concepts in Computer Science, International Workshop WG*, pages 88–99. LNCS 1517.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23.

Brooks, R. A. (1990a). The behavior language; user's guide. A.I. Memo 1227, Massachusetts Institute of Technology.

Brooks, R. A. (1990b). Elephants don't play chess. *Journal of robotics and autonomous systems(1-2)*, 6:3–15.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence, January 1991(1-3)*, 47:139–159.

Brooks, R. A. and Flynn, A. M. (1989). Robot beings. In *Proceedings of the IEEE/RSJ Int'l Conference on Intelligent Robotics and Systems (IROS-89)*, pages 2–10.

Brooks, R. A. and Stein, L. A. (1994). Building brains for bodies. *Autonomous Robots*, 1(1):7–25.

Buvač, S. (1996). Quantificational logic of context. In *Proc. National Conference on Artificial Intelligence (AAAI '96)*.

Cadoli, M. (1992). The complexity of model checking for circumscriptive formulae. *Information Processing Letters*, 44(3):113–118.

Cadoli, M., Eiter, T., and Gottlob, G. (1992). An efficient method for eliminating varying predicates from a circumscription. *Artificial Intelligence*, 54(3):397–410.

Cadoli, M. and Schaerf, M. (1992). Approximate inference in default logic and circumscription. In Neumann, B., editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 319–323, Vienna, Austria. John Wiley & Sons.

Carbone, A. (1997). Interpolants, cut elimination and flow graphs for the propositional calculus. *Annals of Pure and Applied Logic*, 83(3):249–299.

Chadha, R. and Plaisted, D. A. (1993). Finding logical consequences using unskolemization. In *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems (ISMIS'93)*, volume 689 of *Lecture Notes in AI*, pages 255–264. Springer-Verlag.

Chang, C.-L. and Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., and Rice, J. P. (1998). Okbc: A programmatic foundation for knowledge base interoperability. In *Proc. National Conference on Artificial Intelligence (AAAI '98)*, pages 600–607. AAAI Press.

Chekassky, B. V. and Goldberg, A. V. (1997). On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410.

Choueiry, B. Y. and Walsh, T., editors (2000). *Abstraction, reformulation and approximation, Proceedings of the 4th international symposium (SARA'2000)*, volume 1864 of *LNAI*. Springer.

Churchland, P. S. and Sejnowski, T. J. (1992). *The computational brain*. MIT Press.

Clark, K. L. (1978). Negation as failure. In Gallaire, H. and Minker, J., editors, *Logic and Databases*, pages 293–322, New York. Plenum Press.

Clark, P. and Porter, B. (1997). Building concept representations from reusable components. In *Proc. National Conference on Artificial Intelligence (AAAI '97)*, pages 369–376.

Cohen, J. (1988). A view of the origins and development of PROLOG. *Communications of the Association for Computing Machinery*, 31:26–36. Cohen, J.

Cohen, P., Chaudhri, V., Pease, A., and Schrag, R. (1999). Does prior knowledge facilitate the development of knowledge-based systems. In *Proc. National Conference on Artificial Intelligence (AAAI '99)*, pages 221–226. AAAI Press/MIT Press.

Cohen, P., Schrag, R., Jones, E., Pease, A., Lin, A., Starr, B., Gunning, D., and Burke, M. (1998). The darpa high-performance knowledge bases project. *AI Magazine*, 19(4):25–49.

Cohn, A. G. (1997). Qualitative spatial representation and reasoning techniques. In Brewka, G., Habel, C., and Nebel, B., editors, *KI-97, Advances in Artificial Intelligence*, pages 1–30. Springer-Verlag, Berlin.

Conry, S. E., McIntosh, D. J., and Meyer, R. A. (1990). DARES: a distributed automated reasoning system. In *Proc. National Conference on Artificial Intelligence (AAAI '90)*, pages 78–85. AAAI Press/MIT Press.

Cook, S. A. and Mitchell, D. G. (1997). Finding hard instances of the satisfiability problem: a survey. In *Dimacs Series in Discrete Mathamatics and Theoretical Computer Science*, volume 35, Providence, RI, USA. American Mathematical Society.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1989). *Introduction to Algorithms*. McGraw Hill.

Costello, T. (1997). Beyond minimizing change. In *Proceedings of AAAI-97*.

Cowen, R. and Wyatt, K. (1993). BREAKUP: A preprocessing algorithm for satisfiability testing of CNF formulas. *Notre Dame Journal of Formal Logic*, 34(4):602–606.

Cox, P. and Pietrzykowski, T. (1984). A complete nonredundant algorithm for reversed skolemization. *theoretical computer science*, 28:239–261.

Craig, W. (1957a). Linear reasoning. a new form of the herbrand-gentzen theorem. *Journal of Symbolic Logic*, 22:250–268.

Craig, W. (1957b). Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic*, 22:269–285.

Cunningham, W. H. (1991). The optimal multiterminal cut problem. In *DIMACS Series in Disc. Math. and Theor. Comput. Sci.*, volume 5, pages 105–120. American Mathematical Society.

Dantzig, G. B. and Fulkerson, D. R. (1956). On the max-flow min-cut theorem of networks. In Kuhn, H. W. and Tucker, A. W., editors, *Linear Inequalities and Related Systems*, Annals of Mathematics Study, vol. 38, pages 215–221, Princeton Univ. Press, Princeton, NJ.

Darwiche, A. (1996). Utilizing knowledge-based semantics in graph-based algorithms. In *Proc. National Conference on Artificial Intelligence (AAAI '96)*, pages 607–613. Morgan Kaufmann.

Darwin, I. F. (2001). *Java Cookbook: Solutions and examples for Java developers*. O'Reilly Java Series. O'Reilly & Associates.

Davis, E. (1990). *Representation of commonsense knowledge*. Morgan Kaufmann.

Davis, E. (1998). The naive physics perplex. *AI Magazine*, 19(4):51–79.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem proving. *Communications of the ACM*, 5:394–397.

Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215.

de Kleer, J. (1977). Multiple representations of knowledge in a mechanics problem-solver. In *Proc. Fifth International Joint Conference on Artificial Intelligence (IJCAI '77)*, pages 299–304.

de Kleer, J. (1992). An improved incremental algorithm for computing prime implicants. In *Proc. National Conference on Artificial Intelligence (AAAI '92)*, pages 780–785.

de Kleer, J. and Brown, J. S. (1985). A qualitative physics based on confluences. In Hobbs, J. R. and Moore, R. C., editors, *Formal Theories of the Commonsense World*, pages 109–183. Ablex, Norwood, NJ.

Dechter, R. (1990). Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312.

Dechter, R. (1996). Bucket elimination: A unifying framework for probabilistic inference. In *Proc. Twelfth Conference on Uncertainty in Artificial Intelligence (UAI '96)*, pages 211–219.

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1–2):41–85.

Dechter, R. and Pearl, J. (1989). Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366.

Dechter, R. and Rish, I. (1994). Directional resolution: The davis-putnam procedure, revisited. In *Principles of Knowledge Representation and Reasoning: Proc. Fourth International Conference (KR '94)*, pages 134–145. Morgan Kaufmann.

del Val, A. (1999). A new method for consequence finding and compilation in restricted language. In *Proc. National Conference on Artificial Intelligence (AAAI '99)*, pages 259–264. AAAI Press/MIT Press.

Denzinger, J. (1995). Knowledge-based distributed search using teamwork. In Lesser, V., editor, *Proceedings of the First International Conference on Multi–Agent Systems*, pages 81–88, San Francisco, CA. MIT Press.

Denzinger, J. and Dahn, I. (1998). Cooperating theorem provers. In Bibel, W. and Schmitt, P., editors, *Automated Deduction. A basis for applications.*, volume 2, chapter 14, pages 383–416. Kluwer.

Denzinger, J. and Fuchs, D. (1999). Cooperation of heterogeneous provers. In Thomas, D., editor, *Proc. Sixteenth International Joint Conference on Artificial Intelligence (IJCAI '99)*, pages 10–15. Morgan Kaufmann.

Denzinger, J., Fuchs, M., and Fuchs, M. (1997). High performance ATP systems by combining several AI methods. In *Proc. Fifteenth International Joint Conference on Artificial Intelligence (IJCAI '97)*, pages 102–107. Morgan Kaufmann.

Dietrich, S., Urban, S., Harrison, J., and Karadimice, A. (1992). A DOOD RANCH at ASU: Integrating Active, Deductive and Object-Oriented Databases. *IEEE Quarterly Bulletin on Data Engineering, Special Issue on Active Databases*, 15(1-4):40–44.

Dinic, E. A. (1970). Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Mathematics Doklady*, 11:1277–1280.

Doherty, P., Gustafsson, J., Karlsson, L., and Kvarnström, J. (1998). Temporal action logics (TAL) language specification and tutorial. *Electronic Transactions on Artificial Intelligence (http://www.etaij.org)*, 3:nr 15.

Doherty, P., Lukaszewicz, W., and Szalas, A. (1997). Computing circumscription revisited: A reduction algorithm. *Journal of Automated Reasoning*, 18(3):297–336.

Donini, F. M., Lenzerini, M., Nardi, D., and Nutt, W. (1991). The complexity of concept lanauges. In *KR91 Proceedings*.

Doyle, J. (1985). Circumscription and implicit definability. *Journal of Automated Reasoning*, 1(4):391–405.

Eisinger, N. and Ohlbach, H. J. (1993). Deduction systems based on resolution. In Gabbay, D., Hogger, C., and J.A.Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 1: Logical Foundations*. Oxford University Press.

Enderton, H. B. (1972). *A Mathematical Introduction to Logic*. Academic Press.

Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, D. R. (1980). The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys*, 12(2):213–253.

Ernst, G. and Newell, A. (1969). *GPS: A Case Study in Generality and Problem Solving*. Academic Press, New York.

Ertel, W. (1992). OR-parallel theorem proving with random competition. In *Proc. LPAR'92*, volume 624 of *LNAI*, pages 226–237.

Etherington, D. (1986). *Reasoning with incomplete Information*. PhD thesis, University of British Columbia.

Etherington, D. W., Mercer, R. E., and Reiter, R. (1985). On the Adequacy of Predicate Circumscription for Closed World Reasoning. *Computational Intelligence*, 1.

Even, S. (1975). An algorithm for determining whether the connectivity of a graph is at least k. *SIAM Journal on Computing*, 4(3):393–396.

Even, S. (1979). *Graph Algorithms*. Computer Science Press.

Even, S. and Tarjan, R. E. (1975). Network flow and testing graph connectivity. *SIAM Journal on Computing*, 4(4):507–518.

Feige, U. and Krauthgamer, R. (2000). A polylogarithmic approximation of the minimum bisection. In *Proc. 41st IEEE Symp. on Foundations of Computer Science (FOCS'00)*. IEEE Press.

Fikes, R. (1997). Personal communication.

Fikes, R. and Farquhar, A. (1999). Large-scale repositories of highly expressive reusable knowledge. *IEEE Intelligent Systems*, 14(2).

Fikes, R., Farquhar, A., and Rice, J. (1997). Tools for assembling modular ontologies in ontolingua. In *Proc. National Conference on Artificial Intelligence (AAAI '97)*, pages 436–441. AAAI Press.

Finger, J. J. (1987). Exploiting constraints in design synthesis. Technical Report STAN-CS-88-1204, Department of Computer Science, Stanford University, Stanford, CA.

Finger, J. J. and Genesereth, M. R. (1985). Residue: a deductive approach to design synthesis. Technical Report STAN-CS-85-1035, Stanford University, Computer Science Department, Stanford, CA.

Fitting, M. (1990). *First-Order Logic and Automated Theorem Proving*. Springer, New York.

Forbus, K. (1996). Qualitative reasoning. In *CRC Handbook of Computer Science and Engineering*. CRC.

Ford Jr., L. R. and Fulkerson, D. R. (1962). *Flows in networks*. Princeton University Press.

Frege, G. (1879). *Begriffsschrift, eine der Arithmetischen Nachgebildete Formelsprache des Reinen Denkens*. Halle. English translation in (J. van Heijenoort, 1967), pp. 1–82.

Fridman-Noy, N. and Hafner, C. D. (1997). The state of the art in ontology design. *AI Magazine*, 18(3):53–74.

Gabbay, D., Hogger, C., and J.A.Robinson, editors (1993a). *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 1: Logical Foundations*. Oxford University Press, Great Britain.

Gabbay, D., Hogger, C., and J.A.Robinson, editors (1993b). *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 2: Deduction Methodologies*. Oxford University Press, Great Britain.

Gabbay, D., Hogger, C., and J.A.Robinson, editors (1993c). *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*. Oxford University Press, Great Britain.

Gabbay, D., Hogger, C., and J.A.Robinson, editors (1993d). *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 4: Epistemic and Temporal Reasoning*. Oxford University Press, Great Britain.

Gabbay, D., Hogger, C., and J.A.Robinson, editors (1993e). *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 5: Logic Programming*. Oxford University Press, Great Britain.

Gabbay, D. M. and Ohlbach, H. J. (1992). Quantifier elimination in second–order predicate logic. In Nebel, B., Rich, C., and Swartout, W., editors, *Principles of Knowledge Representation and Reasoning (KR92)*, pages 425–435. Morgan Kaufmann.

Gallo, G. and Urbani, G. (1989). Algorithms for testing the satisfiability of propositional formulae. *Journal of Logic Programming*, 7:45–61.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Massachusetts.

Garg, N., Vazirani, V. V., and Yannakakis, M. (1994). Multiway cuts in directed and node weighted graphs. In *Automata, Languages and Programming, 21st ICALP*, volume 820 of *Lecture Notes in Computer Science*, pages 487–498. Springer-Verlag.

Geffner, H. (1991). Beyond negation as failure. In Allen, J., Fikes, R., and Sandewall, E., editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 218–229, San Mateo, CA, USA. Morgan Kaufmann Publishers.

Gelfond, M. and Lifschitz, V. (1989). Compiling Circumscriptive Theories into Logic Programs. In Reinfrank, de Kleer, Ginsberg, and Sandewall, editors, *Non-Monotonic Reasoning*, pages 74–99. Lecture Notes in Artificial Intelligence 346, Springer-Verlag.

Gelfond, M. and Lifschitz, V. (1993). Representing Actions and Change by Logic Programs. *Journal of Logic Programming*, 17:301–322.

Gelfond, M. and Lifschitz, V. (1998). Action languages. *Electronic Transactions on Artificial Intelligence (http://www.etaij.org)*, 3:nr 16.

Gelfond, M. and Przymusinska, H. (1986). Negation as failure: Careful closure procedure. *Artificial Intelligence*, 30:273–287.

Gelfond, M., Przymusinska, H., and Przymusinski, T. C. (1986). The extended closed world assumption and its relation to parallel circumscription. *Proc. Fifth ACM SIGACT-SIGMOD Symposium on Principle of Database Systems*, pages 133–139.

Gelfond, M., Przymusinska, H., and Przymusinski, T. C. (1989). On the relationship between circumscription and negation as failure. *Artificial Intelligence*, 38(1):75–94.

Genesereth, M. R. and Nilsson, N. J. (1987). *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers Inc.

Ghidini, C. and Serafini, L. (1998). Distributed first order logic. In *Proceedings of Frontiers of Combining Systems (FroCoS'98)*, Amsterdam. Also IRST-Technical Report #9804-02.

Giacomo, G. D., Lespérance, Y., and Levesque, H. J. (1997). Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *Proc. Fifteenth International Joint Conference on Artificial Intelligence (IJCAI '97)*, pages 1221–1226. Morgan Kaufmann.

Giacomo, G. D., Reiter, R., and Soutchanski, M. (1998). Execution monitoring of high-level robot programs. In Cohn, A., Schubert, L., and Shapiro, S. C., editors, *Proceedings of the 6th International Conference on Knowledge Representation and Reasoning (KR-98)*, pages 453–464. Morgan Kaufmann.

Ginsberg, M. L. (1989). A circumscriptive theorem prover. *Artificial Intelligence*, 39(2):209–230.

Ginsberg, M. L. (1993). Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46.

Giunchiglia, E., Kartha, G. N., and Lifschitz, V. (1997). Representing Action: Indeterminacy and Ramifications. *Artificial Intelligence*, 95(2):409–438.

Giunchiglia, E. and Traverso, P. (1995). A multi-context architecture for formalizing complex reasoning. *International Journal of Intelligent Systems*, 10:501–539. Also, IRST Tech. Report #9307-26.

Giunchiglia, F. (1994). GETFOL manual - GETFOL version 2.0. Technical Report DIST-TR-92-0010, DIST - University of Genoa. Available at http://ftp.mrg.dist.unige.it/pub/mrg-ftp/92-0010.ps.gz.

Giunchiglia, F. and Ghidini, C. (1998). Local models semantics, or contextual reasoning = locality + compatibility. In *Principles of Knowledge Representation and Reasoning: Proc. Sixth Int'l Conference (KR '98)*.

Giunchiglia, F. and Serafini, L. (1994). Multilanguage hierarchical logics, or: how we can do without modal logics. *Artificial Intelligence*, 65(1):29–70.

Giunchiglia, F. and Walsh, T. (1992). A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–389.

Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198.

Gödel, K. (1967). On formally undecidable propositions of principia mathematica and related systems. In Heijenoort, J. V., editor, *From Frege to Gödel: A Source Book in Mathematical Logic*, pages 592–617. Harvard University Press. A translation of (Gödel, 1931).

Goldberg, A. V. and Tarjan, R. E. (1988). A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940.

Gottlob, G., Leone, N., and Scarcello, F. (1999). A comparison of structural CSP decomposition methods. In *Proc. Sixteenth International Joint Conference on Artificial Intelligence (IJCAI '99)*, pages 394–399. Morgan Kaufmann.

Green, C. (1969). Theorem-proving by resolution as a basis for question-answering systems. In Meltzer, B., Michie, D., and Swann, M., editors, *Machine Intelligence 4*, pages 183–205. Edinburgh University Press, Edinburgh, Scotland.

Grosof, B. N. (1991). Generalizing prioritization. In Allen, J., Fikes, R., and Sandewall, E., editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 289–300, San Mateo, CA, USA. Morgan Kaufmann Publishers.

Grosskreutz, H. and Lakemeyer, G. (2000). cc-golog: Towards more realistic logic-based robot controllers. In *AAAI/IAAI*, pages 476–482.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220.

Guha, R. V. (1991). *Contexts: A Formalization and Some Applications*. PhD thesis, Stanford University. Also published as technical report STAN-CS-91-1399-Thesis, and MCC Technical Report Number ACT-CYC-423-91.

Guha, R. V. and Lenat, D. B. (1990). Cyc: A midterm report. *AI Magazine*, 11(3):33–59.

Gustafsson, J. and Doherty, P. (1996). Embracing occlusion in specifying the indirect effects of actions. In *Proc. 5th Intl Conf. on Knowledge Representation and Reasoning (KR '96)*, pages 87–98.

Gustafsson, J. and Kvarnstrom, J. (2001). Elaboration tolerance through object-orientation. In working notes of the CommonSense'01 symposium.

Haas, A. R. (1987). The case for domain-specific frame axioms. In *proceedings of the 1987 workshop on the frame problem*, pages 343–348.

Haken, A. (1985). The intractability of resolution. *theoretical computer science*, 39:297–308.

Hayes, P. J. (1979). The naive physics manifesto. In Michie, D., editor, *Expert Systems in the Micro-Electronic Age*, pages 242–270. Edinburgh University Press.

Hayes, P. J. (1985a). Naive physics I: Ontology for liquids. In Hobbes, J. R. and Moore, R. C., editors, *Formal Theories of the Commonsense World*, pages 71–107. Ablex Publishiing Corporation.

Hayes, P. J. (1985b). The second naive physics manifesto. In J.R., H. and R.C., M., editors, *Formal Theories of the Commonsense World*, pages 1–36. Ablex.

Herbrand, J. (1930). Recherches sur la théorie de la démonstration. *Travaux de la Société des Sciences et de Lettres de Varsovie, Classe III Sci. Math. Phys.*, 33.

Hewitt, C. F. (1972). Description and theoretical analysis (using schemata) of planner: a language for proving theorems and manipulting models in a robot. Technical Report TR-258, MIT AI Laboratory.

Hilbert, D. and Ackermann, W. (1928). *Grundzüge der theoretischen Logik*. Springer.

Hilbert, D. and Ackermann, W. (1950). *Principles of Mathematical Logic*. Chelsea, New York. A translation into English of (Hilbert and Ackermann, 1928).

Hilbert, D. and Bernays, P. (1934). *Grundlagen der Mathematik*. Springer, Berlin.

Hodges, W. (1997). *A shorter model theory*. Cambridge University Press.

Horswill, I. (1993). Polly: A vision-based artificial agent. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 824–829, Menlo Park, CA, USA. AAAI Press.

Huang, G. (1995). Constructing craig interpolation formulas. In *Conference on computing and combinatorics*, pages 181–190.

Inoue, K. (1991). Consequence-finding based on ordered linear resolution. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 158–164. Morgan Kaufmann.

Inoue, K. (1992). Linear resolution for consequence finding. *Artificial Intelligence*, 56(2-3):301–353.

J. van Heijenoort, E., editor (1967). *From Frege to Gödel, a Source Book in Mathematical Logic*. Harvard University Press, Cambridge, Massachusetts.

Jackson, P. (1992). Computing prime implicants incrementally. In *Proceedings, 11th International Conference on Automated Deduction (CADE)*, LNCS 607, pages 253–267.

Jensen, F. V., Lauritzen, S. L., and Olesen, K. G. (1990). Bayesian updating in recursive graphical models by local computation. *Computational Statistics Quarterly*, 4:269–282.

Kaelbling, L. P. (1987). REX: A symbolic language for the design and parallel implementation of embedded systems. In *Proceedings of the AIAA conference on computers in aerospace*, pages 255–260.

Kaelbling, L. P. (1990). An architecture for intelligent reactive systems. In Allen, J., Hendler, J., and Tate, A., editors, *Readings in Planning*, pages 713–728. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

Kaelbling, L. P. and Rosenschein, S. J. (1990). Action and planning in embedded agents. *Robotics and Autonomous Systems(1–2), June 1990*, 6:35–48.

Kamps, J. (1999). On criteria for formal theory building: Applying logic and automated reasoning tools to the social sciences. In Hendler, J. and Subramanian, D., editors, *Proc. National Conference on Artificial Intelligence (AAAI '99)*, pages 285–290. AAAI Press.

Kamps, J. and P'eli, G. (1995). Qualitative reasoning beyond the physics domain: The density dependence theory of organizational ecology. In *Proceedings of QR-95*, pages 114–121.

Kean, A. and Tsiknis, G. (1990). An incremental method for generating prime implicants/implicates. *Journal of Symbolic Computation*, 9:185–206.

Khatib, O. (1992). Inertial properties in robotic manipulation. Technical Report STAN-CS-92-1457, Stanford University.

Kifer, M., Lausen, G., and Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843.

Kjaerulff, U. (1993). *Aspects of efficiency imporvement in bayesian networks*. PhD thesis, Aalborg University, Department of Mathematics and Computer Science, Fredrik Bajers Vej 7E, DK-9220 Aalborg, Denmark.

Klein, P., Agrawal, A., Ravi, R., and Rao, S. (1990). Approximation through multicommodity flow. In *Proc. 31st IEEE Symp. on Foundations of Computer Science (FOCS'90)*, pages 726–739. IEEE Press.

Klein, P., Plotkin, S. A., and Rao, S. (1993). Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 682–690.

Kloks, T. (1994). *Treewidth: computations and approximations*, volume 842 of *LNCS*. Springer-Verlag.

Kohlas, J., Haenni, R., and Moral, S. (1999). Propositional information systems. *Journal of Logic and Computation*, 9(5):651–681.

Kolaitis, P. G. and Papadimitriou, C. H. (1990). Some computational aspects of circumscription. *Journal of the ACM*, 37(1):1–14.

Koller, D. and Pfeffer, A. (1997). Object-oriented Bayesian networks. In *Proc. of the 13th Conf. on Uncertainty in Artificial Intelligence (UAI-97)*, pages 302–313. Morgan Kaufmann.

Kortenkamp, D., Bonasso, R. P., and Murphy, R., editors (1998). *Artificial Intelligence and Mobile Robots*. AAAI Press/MIT Press.

Kowalski, R. A. (1978). Logic for data description. In Minker, H. G. J., editor, *Logic and Data Bases*, pages 77–102. Plenum Press, New York.

Kowalski, R. A. and Hayes, P. J. (1969). Semantic trees in automatic theorem-proving. *Machine Intelligence*, 4:87–101.

Krajiček, J. (1997). Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *Journal of Symbolic Logic*, 62(2):457–486.

Kraus, S., Lehmann, D., and Magidor, M. (1990). Nonmonotonic Reasoning, Preferential Models and Cumulative Logics. *Artificial Intelligence*, 44(1):167–207.

Lagergren, J. (1990). Efficient parallel algorithms for tree-decomposition and related problems. In *Proc. 31st IEEE Symp. on Foundations of Computer Science (FOCS'90)*, pages 173–182. IEEE Press.

Lagergren, J. and Arnborg, S. (1991). Finding minimal forbidden minors using a finite congruence. In *Proc. 18th Int. Coll. Automata, Languages and Programming*, number 510 in Lecture Notes in Computer Science, pages 532–543. Springer-Verlag.

Laird, J. E., Rosenbloom, P. S., and Newell, A. (1986). Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46.

Lakemeyer, G. (1999). On sensing and off-line interpreting in GOLOG. In Levesque, H. and Pirri, F., editors, *Logical Foundations for Cognitive Agents*. Springer.

Latombe, J.-C. (1991). *Robot Motion Planning*. Kluwer Academic Publishers.

Lauritzen, S. L. and Spiegelhalter, D. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society B*, 50(2):157–224.

Lausen, G. and Vossen, G. (1998). *Models and Languages of Object-Oriented Databases*. Addison-Wesley.

Lee, R. C.-T. (1967). *A Completeness Theorem and a Computer Program for Finding Theorems Derivable from Given Axioms*. PhD thesis, University of California, Berkeley.

Leighton, T., Plotkin, F. M. S., Stein, C., Tardos, E., and Tragoudas, S. (1995). Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50(2):228–243.

Leighton, T. and Rao, S. (1988). An approximate max-flow min-cut theorem for uniform multi-commodity flow problems with applications to approximation algorithms. In *Proc. 29th IEEE Symp. on Foundations of Computer Science (FOCS'88)*, pages 422–431.

Leighton, T. and Rao, S. (1999). Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832.

Lenat, D. B. (1995). Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38.

Lenat, D. B. and Guha, R. V. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley.

Levesque and Brachman (1985). A fundamental tradeoff in knowledge representation and reasoning. In Brachman, R. and Levesque, H., editors, *Readings in Knowledge Representation*, pages 41–70. Morgan Kaufmann. Revised from a paper that appeared with the same title in Proc. Fourth Conference of the Canadian Society for Computational Studies of Intelligence.

Levesque, H., Pirri, F., and Reiter, R. (1998). Foundations for the situation calculus. *Electronic Transactions on Artificial Intelligence (http://www.etaij.org)*, 3:nr 18.

Levesque, H., Reiter, R., Lesprance, Y., Lin, F., and Scherl, R. (1997). Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84.

Lifschitz, V. (1985). Computing circumscription. In *Proc. of IJCAI-85*, pages 121–127.

Lifschitz, V. (1986a). On formalizing contexts. Unpublished manuscript.

Lifschitz, V. (1986b). On the Satisfiability of Circumscription. *Artificial Intelligence*, 28:17–27.

Lifschitz, V. (1989). Between circumscription and autoepistemic logic. In Brachman, H. J. L. R. J. and Reiter, R., editors, *Proc. First International Conference on Principles of Knowledge Representation and Reasoning (KR '89)*, pages 235–244, Toronto, Canada. Morgan Kaufmann.

Lifschitz, V. (1993). Circumscription. In Gabbay, D., Hogger, C., and Robinson, J., editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*. Oxford University Press.

Lifschitz, V. (1998). Cracking an egg: an exercise in formalizing commonsense reasoning. In working notes of the CommonSense'98 symposium.

Lifschitz, V. (2000). Missionaries and cannibals in the causal calculator. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conference (KR '2000)*, pages 85–96. Morgan Kaufmann.

Lin, F. (1997). Applications of the situation calculus to formalizing control and strategic information: the prolog cut operator. In *Proc. Fifteenth International Joint Conference on Artificial Intelligence (IJCAI '97)*, pages 1412–1418.

Lin, F. (1999). Search algorithms in the situation calculus. In *Logical Foundations of Cognitive Agents: contributions in honor of Ray Reiter*, pages 213–233. Springer.

Lin, F. (2000). On strongest necessary and weakest sufficient conditions. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conference (KR '2000)*, pages 167–175.

Lin, F. and Reiter, R. (1994). State constraints revisited. *Journal of Logic and Computation*, 4(5):655–678.

Lloyd, J. W. and Topor, R. W. (1985). A basis for deductive database systems. *Journal of Logic Programming*, 2:93–109.

Loveland, D. W. (1969). A simplified format for the model elimination procedure. *Journal of the ACM*, 16(3):349–363. Reprinted in: (Siekmann and Wrightson, 1983).

Loveland, D. W. (1970). A linear format for resolution. In *Proc. IRIA symposium on automatic demonstration*, volume 125 of *Lecture notes in mathematics*, pages 147–162. Springer-Verlag.

Loveland, D. W. (1978). *Automated theorem proving: a logical basis*. Fundamental studies in computer science. North-Holland.

Luckham, D. (1970). Refinement theorem in resolution theory. In *Proc. IRIA symposium on automatic demonstration*, volume 125 of *Lecture notes in mathematics*, pages 163–190. Springer-Verlag.

Lyndon, R. C. (1959). An interpolation theorem in the predicate calculus. *Pacific Journal of Mathematics*, 9(1):129–142.

Maier, D. (1986). A logic for objects. In Minker, J., editor, *Workshop on Foundations of Deductive Databases and Logic Programming*, pages 6–26.

Makinson, D. (1993). General patterns in nonmonotonic reasoning. In Gabbay, D., C.J.Hogger, and J.A.Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*. Oxford University Press.

Marquis, P. (1995). Knowledge compilation using theory prime implicates. In *IJCAI-95*, pages 837–843.

Marquis, P. (2000). Consequence finding algorithms. In Gabbay, D. and Smets, P., editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 5: Algorithms for defeasible and uncertain reasoning. Kluwer.

Matarić, M. J. (1992). Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312.

McCarthy, J. (1958a). Programs with Common Sense. In *Mechanisation of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory*, pages 77–84, London, U.K. Her Majesty's Stationery Office. Reprinted in McC90.

McCarthy, J. (1958b). Programs with common sense. In *Mechanisation of Thought Processes, Proceedings of the Symposium of the National Physics Laboratory*, pages 77–84, London, U.K. Her Majesty's Stationery Office. Reprinted in McC90.

McCarthy, J. (1977). Epistemological problems in artificial intelligence. In *Proc. 5th International Conference on Artificial Intelligence*, pages 1038–1044.

McCarthy, J. (1980). Circumscription—A Form of Non-Monotonic Reasoning. *Artificial Intelligence*, 13:27–39.

McCarthy, J. (1986a). Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–118.

McCarthy, J. (1986b). Applications of Circumscription to Formalizing Common Sense Knowledge. *Artificial Intelligence*, 28:89–116.

McCarthy, J. (1993). Notes on Formalizing Context. In *Proc. Thirteenth International Joint Conference on Artificial Intelligence (IJCAI '93)*. Available on http://www-formal.stanford.edu/jmc/.

McCarthy, J. (1998). Elaboration Tolerance. In CommonSense '98 and in http://www-formal.stanford.edu /jmc/elaboration.html.

McCarthy, J. and Buvač, S. (1998). Formalizing Context (Expanded Notes). In Aliseda, A., Glabbeek, R. v., and Westerståhl, D., editors, *Computing Natural Language*, volume 81 of

CSLI Lecture Notes, pages 13–50. Center for the Study of Language and Information, Stanford University, Stanford, California.

McCarthy, J. and Costello, T. (1998). Combining Narratives. In *Principles of Knowledge Representation and Reasoning: Proc. Sixth Int'l Conference (KR '98)*. Morgan Kaufmann.

McCarthy, J. and Hayes, P. J. (1969). Some Philosophical Problems from the Standpoint of Artificial Intelligence. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press.

McIlraith, S. (1998). Logic-based abductive inference. Technical Report KSL-98-19, Knowledge Systems Lab, Department of Computer Science, Stanford University.

McIlraith, S. (2000). Integrating actions and state constraints: a closed-form solution to the ramification problem (sometimes). *Artificial Intelligence*, 16(1–2):87–121.

McIlraith, S. and Amir, E. (2001). Theorem proving with structured theories. In *Proc. Seventeenth International Joint Conference on Artificial Intelligence (IJCAI '01)*, pages 624–631. Morgan Kaufmann.

Meyer, A. R. (1980). A note on the length of Craig's interpolants. Technical Memo MIT/LCS/TM-183, Massachusetts Institute of Technology, Laboratory for Computer Science.

Meyer, B. (1997). *Object-Oriented Software Construction*. Prentice-Hall, 2nd edition.

Michael Gelfond, Vladimir Lifschitz, H. P. and Schwarz, G. (1994). Autoepistemic logic and introspective circumscription. In Fagin, R., editor, *Proceedings of the 5th Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 197–207, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Miller, R. and Shanahan, M. (1999). The event calculus in classical logic – alternative axiomatizations. *Electronic Transactions on Artificial Intelligence (http://www.etaij.org)*, 4:nr 16. under review.

Minicozzi, E. and Reiter, R. (1972). A note on linear resolution strategies in consequence-finding. *Artificial Intelligence*, 3:175–180.

Minker, J. (1982). On indefinite databases and the closed world assumption. In Loveland, D. W., editor, *6th Conference on Automated Deduction*, volume 138 of *Lecture Notes in Computer Science*, pages 292–308. Springer-Verlag, New York.

Minker, J., Lobo, J., and Rajasekar, A. (1991). Circumscription and Disjunctive Logic Programming. In Lifschitz, V., editor, *Artificial Intelligence and Mathematical Theory of Computation. Papers in Honor of John McCarthy*, pages 281–304. Academic Press.

Minsky, M. (1975). A framework for representing knowledge. Winston.

Minsky, M. (1985). *The Society of Mind*. Simon and Schuster.

Mitchell, D., Selman, B., and Levesque, H. (1992). Hard and easy distributions of sat problems. In *Proc. National Conference on Artificial Intelligence (AAAI '92)*, pages 459–465, San Jose, California. AAAI Press. Selman, B.Levesque, H.Mitchell, D.

Morgenstern, L. (1996). Inheriting well-formed formulae in a formula-agumented semantic network. In Aiello, L. C., Doyle, J., and Shapiro, S., editors, *Proc. 5th Intl Conf. on Knowledge Representation and Reasoning (KR '96)*, pages 268–279, San Francisco. Morgan Kaufmann.

Morgenstern, L. (1998). Inheritance comes of age: Applying nonmonotonic techniques to problems in industry. *Artificial Intelligence*, 103(1–2):237–271.

Morgenstern, L. (2001). Mid-sized axiomatizations of commonsense problems: A case study in egg cracking. *Studia Logica*, 67.

Moss, C. (1994). *Prolog++, The Power of Object-Oriented and Logic Programming*. Addison-Wesley.

Mott, P. L. (1987). A theorem on the consistency of circumscription. *Artificial Intelligence*, 31(1):87–98.

Naqvi, S. A. (1986). Some extensions to the closed world assumption in databases. In Ausiello, G. and Atzeni, P., editors, *Proceedings of the International Conference on Database Theory*, pages 341–348, Rome, Italy. Springer-Verlag.

Nayak, P. and Levy, A. (1995). A semantic theory of abstractions. In *Proc. Fourteenth International Joint Conference on Artificial Intelligence (IJCAI '95)*, pages 196–202. Morgan Kaufmann.

Nelson, G. and Oppen, D. C. (1979). Simplification by cooperating decision procedures. *ACM Trans. on Programming Languages and Systems*, 1(2):245–257.

Newell, A. (1973). *Production System: Models of Control Structure*, pages 463–526. Academic Press, New York.

Ngair, T.-H. (1993). A new algorithm for incremental prime implicate generation. In *Proc. Thirteenth International Joint Conference on Artificial Intelligence (IJCAI '93)*.

Nii, H. P., Feigenbaum, E. A., Anton, J. J., and Rockmore, A. J. (1982). Signal to symbol transformation: HASP/SIAP case study. *Artificial Intelligence Magazine*, 3:23–35.

Nii, P. (1986). Blackboard systems. *AI Magazine*, 7(2).

Nilsson, N. J. (1969). A mobile automaton: An application of artificial intelligence techniques. In *Proc. First International Joint Conference on Artificial Intelligence (IJCAI '69)*, pages 509–520.

Nilsson, N. J. (1984). Shakey the robot. Technical Report 323, SRI International, Menlo Park, California.

Park, T. J. and Gelder, A. V. (1996). Partitioning methods for satisfiability testing on large formulas. In *Proceedings of the Thirteenth International Conference on Automated Deduction (CADE-13)*, pages 748–762. Springer-Verlag.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann.

Pearl, J. (1990). Reasoning under uncertainty. *Annual Review of Computer Science*, 4:37–72.

Pednault, E. P. D. (1989). ADL: exploring the middle ground between STRIPS and the situation calculus. In *Proc. First International Conference on Principles of Knowledge Representation and Reasoning (KR '89)*, pages 324–332.

Perlis, D. and Minker, J. (1986). Completeness Results for Circumscription. *Artificial Intelligence*, 28:29–42.

Pinto, J. (1998). Concurrent actions and interacting effects. In *Principles of Knowledge Representation and Reasoning: Proc. Sixth Int'l Conference (KR '98)*. Morgan Kaufmann.

Plaisted, D. A. (1994). The search efficiency of theorem proving strategies. In *Proceedings of the Twelfth International Conference on Automated Deduction (CADE-12)*, pages 57–71.

Pradhan, M., Provan, G., Middleton, B., and Henrion, M. (1994). Knowledge engineering for large belief networks. In *Proc. Tenth Conference on Uncertainty in Artificial Intelligence (UAI '94)*, pages 484–490. Morgan Kaufmann.

Przymusinski, T. C. (1989). An algorithm to compute circumscription. *Artificial Intelligence*, 38(1):49–73.

Rabinov, A. (1989). A generalization of collapsible cases of circumscription. *Artificial Intelligence*, 38(1):111–117.

Rajasekar, A., Lobo, J., and Minker, J. (1989). Weak generalized closed world assumption. *Journal of Automated Reasoning*, 5(3):293–308.

Reed, B. A. (1992). Finding approximate separators and computing tree width quickly. In *Proc. 24th ACM Symp. on Theory of Computing*, pages 221–228. ACM Press.

Reiter, R. (1971). Two results on ordering for resolution with merging and linear format. In *Journal of the ACM*, volume 18, pages 630–646.

Reiter, R. (1978). On closed world data bases. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases*, pages 55–76. Plenum, New York / London.

Reiter, R. (1982). Circumscription implies predicate completion (sometimes). *Proc. National Conference on Artificial Intelligence (AAAI '82)*, pages 418–420.

Reiter, R. (1991). The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Lifschitz, V., editor, *Artificial Intelligence and Mathematical Theory of Computation (Papers in Honor of John McCarthy)*, pages 359–380. Academic Press.

Reiter, R. (1996). Natural actions, concurrency and continuous time in the situation calculus. In *Principles of Knowledge Representation and Reasoning, Proceedings of the fifth International Conference (KR '96)*. Morgan Kaumann.

Reiter, R. (1998). Sequential, temporal GOLOG. In *Principles of Knowledge Representation and Reasoning: Proc. Sixth Int'l Conference (KR '98)*. Morgan Kaumann.

Reiter, R. (2001). *Knowledge In Action: Logical Foundations for Describing and Implementing Dynamical Systems*. MIT Press.

Reynolds, J. C. (1965). Unpublished seminar note, stanford university, stanford, ca. Referenced in (Slagle, 1967).

Ringeissen, C. (1996). Cooperation of decision procedures for the satisfiability problem. In Baader, F. and Schulz, K., editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 121–140. Kluwer Academic Publishers.

Rish, I. and Dechter, R. (2000). Resolution versus search: two strategies for SAT. *Journal of Automated Reasoning*, 24(1-2):225–275.

Robertson, N. and Seymour, P. D. (1986). Graph minors. II: algorithmic aspects of treewidth. *Journal of Algorithms*, 7:309–322.

Robertson, N. and Seymour, P. D. (1995). Graph minors XIII. the disjoint paths problem. *J. Comb. Theory, Series B*, 63:65–110.

Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41.

Rose, D. J. (1974). Triangulated graphs and the elimination process. *J. of Discrete Mathematics*, 7:317–322.

Rosenshein, S. J. and Kaelbling, L. P. (1995). A situated view of representation and control. *Artificial Intelligence*, 73(1–2):149–173.

Roth, D. and tau Yih, W. (2001). Relational learning via propositional algorithms: An information extraction case study. In *Proc. Seventeenth International Joint Conference on Artificial Intelligence (IJCAI '01)*, pages 1257–1263. Morgan Kaufmann.

Sandewall, E. (1994). *Features and Fluents*. Oxford University Press.

Sandewall, E. and Shoham, Y. (1994). Nonmonotonic Temporal Reasoning. In Gabbay, D., C.J.Hogger, and J.A.Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 4: Epistemic and Temporal Reasoning*. Oxford University Press.

Schiermeyer, I. (1996). Pure literal look ahead: an O($1,497^n$) 3-satisfiability algorithm (extended abstract). Technical report, University of Köln. Workshop on the Satisfiability Problem, Siena April 29-May 3.

Schubert, L. K. (1990). Monotonic solution of the frame problem in the situation calculus. In Kryburg, H., Loui, R., and Carlson, G., editors, *knowledge representation and defeasible reasoning*, pages 23–67. Kluwer.

Selman, B. and Kautz, H. (1993). Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *IJCAI-93*.

Selman, B., Kautz, H. A., and Cohen, B. (1994). Noise strategies for local search. In *Proc. 12th National Conference on Artificial Intelligence, AAAI'94, Seattle/WA, USA*, pages 337–343.

Selman, B., Levesque, H. J., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In Rosenbloom, P. and Szolovits, P., editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California. American Association for Artificial Intelligence, AAAI Press.

Selman, B., Mitchell, D., and Levesque, H. (1997). Generating hard satisfiability problems. *Artificial Intelligence*, pages 17–29. Selman, B.Mitchell, D.Levesque, H.

Shanahan, M. (1997). *Solving the Frame Problem, a mathematical investigation of the common sense law of inertia*. MIT press, Cambridge, MA.

Shanahan, M. (1998a). A logical account of the common sense informatic situation for a mobile robot. *Electronic Transactions on Artificial Intelligence (http://www.etaij.org)*, 2:69–104.

Shanahan, M. (2000). Reinventing shakey. In Minker, J., editor, *Logic-Based Artificial Intelligence*. Kluwer.

Shanahan, M. and Witkowski, M. (2000). High-level robot control through logic. In Castelfranchi, C. and Lesprance, Y., editors, *Agent Theories, Architectures and Languages: 7th International Workshop (ATAL'00)*. Springer.

Shanahan, M. P. (1996). Robotics and the common sense informatic situation. In *Proceedings ECAI 96*, pages 684–688.

Shanahan, M. P. (1998b). A logical formalization of ernie davis's egg-cracking problem. In working notes of the CommonSense'98 symposium.

Shapiro, E. and Takeuchi, A. (1983). Object oriented programming in concurrent prolog. *New Generation Computing*, 1:25–48.

Shepherdson, J. C. (1984). Negation as finite failure: A comparison of Clark's completed database and Reiter's closed world assumption. *Journal of Logic Programming*, 1(15):51–79.

Shi, J. and Malik, J. (1997). Normalized cuts and image segmentation. In *Proceedings of IEEE CVPR*, pages 731–737.

Shoenfield, J. R. (1967). *Mathematical Logic*. Addison-Wesley, Duke University.

Shoham, Y. (1991). Varieties of context. In Lifschitz, V., editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*. Academic Press.

Shoikhet, K. and Geiger, D. (1997). A practical algorithm for finding optimal triangulations. In *Proc. National Conference on Artificial Intelligence (AAAI '97)*, pages 185–190. Morgan Kaufmann.

Shostak, R. E. (1984). Deciding combinations of theories. *Journal of the ACM*, 31:1–12.

Siekmann, J. and Wrightson, G., editors (1983). *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, volume 2. Springer-Verlag.

Slagle, J. R. (1967). Automatic theorem proving with renamable and semantic resolution. *Journal of the ACM*, 14(4):687–697.

Slagle, J. R. (1970). Interpolation theorems for resolution in lower predicate calculus. *Journal of the ACM*, 17(3):535–542.

Slagle, J. R., Chang, C.-L., and Lee, R. C. T. (1969). Completeness theorems for semantic resolution in consequence-finding. In *Proc. First International Joint Conference on Artificial Intelligence (IJCAI '69)*, pages 281–285.

Slaney, J. and Surendonk, T. J. (1996). Combining finite model generation with theorem proving: Problems and prospects. In Baader, F. and Schulz, K., editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 141–156. Kluwer Academic Publishers.

Stallman, R. M. and Sussman, G. J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135–196.

Stein, L. A. (1997). Postmodular systems: Architectural principles for cognitive robotics. *Cybernetics and Systems*, 28(6):471–487. Available from http://www.ai.mit.edu/people/las/cv.html.

Sterling, L. and Shapiro, E. (1994). *The Art of Prolog*. The MIT Press, Cambridge, Mass., second edition.

Stickel, M. E. (1985). Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–355.

Stickel, M. E. (1988a). A Prolog technology theorem prover. In Lusk, E. and Overbeek, R., editors, *Proc. 9th International Conference on Automated Deduction*, pages 752–753. Springer LNCS, New York.

Stickel, M. E. (1988b). A Prolog Technology Theorem Prover: implementation by an extended Prolog compiler. *Journal of Automated Reasoning*, 4:353–380. Stickel, M. E.

Stickel, M. E. (1992). A Prolog Technology Theorem Prover: a new exposition and implementation in Prolog. *Theoretical Computer Science*, 104:109–128.

Stroustrup, B. (1997). *The C++ Programming Language: Third Edition*. Addison-Wesley Publishing Co., Reading, Mass.

Subramanian, D. (1989). *A Theory of Justified Reformulations*. PhD thesis, Stanford University. Also published as technical report STAN-CS-89-1260-Thesis.

Sussman, G. J. (1990). The virtuous nature of bugs. In Allen, J., Hendler, J., and Tate, A., editors, *Readings in Planning*, pages 111–117. Morgan Kaufmann.

Sutcliffe, G. C. J. (1992). A heterogeneous parallel deduction system. FGCS'92 Workshop on Automated Deduction: Logic Programming and Parallel Computing Approaches.

Suttner, C. B. (1997). SPTHEO. *Journal of Automated Reasoning*, 18:253–258.

Thielscher, M. (1998). Introduction to the fluent calculus. *Electronic Transactions on Artificial Intelligence (http://www.etaij.org)*, 3:nr 14.

Thomason, R., Horty, J., and Touretzky, D. S. (1986). A Calculus for Inheritance in Monotonic Semantic Nets. Research Note CMU-CS 86-138, Carnegie Mellon.

Tinelli, C. and Harandi, M. T. (1996). A new correctness proof of the Nelson–Oppen combination procedure. In Baader, F. and Schulz, K., editors, *Frontiers of Combining Systems: Proceedings of the 1st International Workshop, Munich (Germany)*, Applied Logic, pages 103–120. Kluwer Academic Publishers.

Tseitin, G. (1970). On the complexity of proofs in propositional logics. *Seminars in Mathematics*, 8.

Ullman, J. D. (1988). *Principles of Database and knowledge-base systems*, volume 1. Computer Science Press.

Urquhart, A. (1987). Hard examples for resolution. *Journal of the ACM*, 34(1):209–219.

Van Emden, M. H. and Kowalski, R. A. (1976). The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742.

W. T. Sullivan, I., Werthimer, D., Bowyer, S., Cobb, J., Gedye, D., and Anderson, D. (1997). A new major seti project based on project serendip data and 100,000 personal computers. In Cosmovici, C., Bowyer, S., , and Werthimer, D., editors, *Astronomical and Biochemical Origins and the Search for Life in the Universe. Proc. of the Fifth Intl. Conf. on Bioastronomy.*, IAU Colloq. No. 161. Editrice Compositori, Bologna, Italy.

Wos, L., Robinson, G. A., and Carson, D. F. (1965). Efficiency and completeness of the set of support strategy in theorem proving. *Journal of the ACM*, 12(4):536–541.

Yahya, A. H. and Henschen, L. J. (1985). Deduction in non-Horn databases. *Journal of Automated Reasoning*, 1(2):141–160.

Yates, R. A., Raphael, B., and Hart, T. P. (1970). Resolution graphs. *Artificial Intelligence*, 1:257–289.

Yuan, L. Y. and You, J.-H. (1993). Autoepistemic circumscription and logic programming. *Journal of Automated Reasoning*, 10(2):143–160.