
**THE KLUWER INTERNATIONAL SERIES
IN ENGINEERING AND COMPUTER SCIENCE**

LOGIC-BASED ARTIFICIAL INTELLIGENCE

edited by

Jack Minker

*Institute for Advanced Computer Studies
and Department of Computer Science
University of Maryland, U.S.A.*



KLUWER ACADEMIC PUBLISHERS
Boston / Dordrecht / London

Distributors for North, Central and South America:

Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, Massachusetts 02061 USA
Telephone (781) 871-6600
Fax (781) 871-6528
E-Mail <kluwer@wkap.com>

Distributors for all other countries:

Kluwer Academic Publishers Group
Distribution Centre
Post Office Box 322
3300 AH Dordrecht, THE NETHERLANDS
Telephone 31 78 6392 392
Fax 31 78 6546 474
E-Mail <services@wkap.nl>

 Electronic Services <<http://www.wkap.nl>>

Library of Congress Cataloging-in-Publication Data

Logic-based artificial intelligence / edited by Jack Minker.

p. cm.-- (The Kluwer international series in engineering and computer science; SECS 597)

Includes bibliographical references and index.

ISBN 0-7923-7224-7 (alk. paper)

1. Computer logic. 2. Logic, Symbolic and mathematical. 3. Artificial intelligence. I.

Minker, Jack. II. Series.

QA76.9.L63 L62 2000

006.3--dc21

00-052177

Copyright © 2000 by Kluwer Academic Publishers.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, mechanical, photo-copying, recording, or otherwise, without the prior written permission of the publisher, Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts 02061

Printed on acid-free paper.

Printed in the United States of America

*The Publisher offers discounts on this book for course use and bulk purchases.
For further information, send email to <lance.wobus@wkap.com>*

Contents

Contributing Authors	xiii	
Preface	xxi	
Part I Introduction to Logic-Based Artificial Intelligence		
1		
Introduction to Logic-Based Artificial Intelligence	3	
<i>Jack Minker</i>		
1	Introduction and Background	3
2	Background	4
3	Commonsense Reasoning	9
4	Knowledge Representation	10
5	Nonmonotonic Reasoning	11
6	Abductive and Inductive Reasoning	12
7	Logic, Probability and Decision Making	13
8	Logic for Causation and Actions	14
9	Planning and Problem Solving	15
10	Logic, Planning, and High Level Robotics	16
11	Logic for Agents and Actions	17
12	Theory of Beliefs	17
13	Logic and Language	18
14	Computational Logic	19
15	System Implementations	20
16	Logic Applications To Mechanical Checking and Data Integration	21
17	Summary and Conclusions	27
Part II Commonsense Reasoning		
2		
Concepts of Logical AI	37	
<i>John McCarthy</i>		
1	Introduction and exhortation	37
2	Epistemology	38
3	Heuristics	45

4	Robots	47
5	Reasoning about Action	49
6	Learning	51
7	Creativity	52
8	Acknowledgments and remarks	52
 Part III Knowledge Representation		
3		
Two Approaches to Efficient Open-World Reasoning	59	
<i>Giuseppe De Giacomo, Hector Levesque</i>		
1	Introduction	59
2	Evaluation-based reasoning	61
3	A First approach	63
4	A Second approach	68
5	Conclusion	76
4		
Declarative Problem-Solving in DLV	79	
<i>Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer</i>		
1	Introduction	79
2	Core Language	82
3	The “Guess&Check” Paradigm	84
4	Front-ends	90
5	Usage	94
6	The DLV Problem Solving Engine	95
7	Ongoing Work and Conclusion	97
 Part IV Nonmonotonic Reasoning		
5		
The Role of Default Logic in Knowledge Representation	107	
<i>James P. Delgrande, Torsten Schaub</i>		
1	Introduction	108
2	Default Logic	109
3	A Methodology for the Use of Default Logic	111
4	Preference	113
5	Inheritance of Properties	119
6	Discussion	122
6		
Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning	127	
<i>Marc Denecker, Victor Marek and Mirosław Truszczyński</i>		
1	Introduction	128
2	Preliminaries from lattice theory	130
3	Approximating operators	133
4	Stable operator and well-founded fixpoint	136
5	Applications in knowledge representation	140
6	Conclusions	142
 Part V Logic for Causation and Actions		

7	Getting to the Airport: The Oldest Planning Problem in AI <i>Vladimir Lifschitz, Norman McCain, Emilio Remolina, Armando Tacchella</i>	147
1	Introduction	148
2	Bar-Hillel's Objections	149
3	Other Difficulties	152
4	Airport Domain in Action Language C	153
5	Solving the Planning Problem	158
6	Conclusion	162
 Part VI Planning and Problem Solving		
8	Encoding Domain Knowledge for Propositional Planning <i>Henry Kautz, Bart Selman</i>	169
1	Introduction	170
2	Domain dependent planning	174
3	The logical status of heuristics	174
4	Experiments	177
5	Conclusions and Future Work	182
9	Functional Strips <i>Héctor Geffner</i>	187
1	Introduction	188
2	Strips	190
3	Functional Strips	193
4	Examples	198
5	Resources	202
6	Planning and Problem Solving	203
7	Discussion	205
 Part VII Logic, Planning and High Level Robotics		
10	Planning with Natural Actions in the Situation Calculus <i>Fiora Pirri, Raymond Reiter</i>	213
1	Introduction	214
2	Preliminaries	214
3	Natural Actions and Executable Situations	219
4	Planning with Natural Actions in the Situation Calculus	221
5	Implementation	225
6	Discussion	229
11	Reinventing Shakey <i>Murray Shanahan</i>	233
1	Introduction	233
2	Representing Action	235
3	A Logical Account of Planning	238
4	A Logical Account of Perception	241
5	Interleaving Sensing, Planning and Acting	248

6	Related Work	250
7	Concluding Remarks	252
 Part VIII Logic for Agents and Actions		
12		
Reasoning Agents in Dynamic Domains		257
<i>Chitta Baral, Michael Gelfond</i>		
1	Introduction	257
2	Modeling the agent	259
3	Reasoning algorithms	270
4	Conclusion	275
13		
Dynamic Logic for Reasoning about Actions and Agents		281
<i>J.-J. Ch. Meyer</i>		
1	Introduction	281
2	Dynamic Logic As a Basic Logic of Actions	282
3	Dynamic Update Logic	285
4	Dynamic Semantics of Reasoning Systems	288
5	Dynamic Deontic Logic	290
6	Typical Effects of Commonsense Actions and the Frame Prob- lem	293
7	Specifying Intelligent Agents	297
8	Discussion and Conclusion	306
 Part IX Inductive Reasoning		
14		
Logic-Based Machine Learning		315
<i>Stephen Muggleton and Flaviu Marginean</i>		
1	Introduction	315
2	ILP	316
3	Discovery of biological function	320
4	Learning Language in Logic	322
5	Conclusion	326
 Part X Possibilistic Logic		
15		
Decision, Nonmonotonic Reasoning, Possibilistic Logic		333
<i>Salem Benferhat, Didier Dubois, Hélène Fargier, Henri Prade, Régis Sabbadin</i>		
1	Introduction	334
2	A Framework for Qualitative Decision	335
3	Qualitative Decision Without Commensurateness	341
4	Logical Handling of Preferences	346
5	Concluding Remarks	353
 Part XI Logic and Beliefs		

16		
The Role(s) of Belief in AI <i>Don Perlis</i>		361
1 Introduction	361	
2 What are beliefs?	362	
3 Why are beliefs important?	365	
4 Why are beliefs (partly) logical in nature?	367	
5 Why are beliefs problematic?	367	
6 Conclusion	372	
17		
Modeling the Beliefs of Other Agents <i>Richmond H. Thomason</i>		375
1 Introduction and background	375	
2 The problem of achieving mutuality	376	
3 Subagent simulation as an agent modeling mechanism	382	
4 Modeling the multiplicity of single-agent beliefs	383	
5 Modeling the beliefs of many agents	384	
6 Formulating propositional generalizations	388	
7 Circumscriptive Reasoning about Beliefs	389	
8 Achieving mutuality through nonmonotonic reasoning	391	
9 An Example	396	
10 Conclusion	400	
Part XII Logic and Language		
18		
The Situations We Talk about <i>Lenhart K. Schubert</i>		407
1 Introduction: Competing Views on the Relation between Sentences and Situations	408	
2 Why we need '***'	410	
3 Situations described by negative, conjoined and quantified sentences	413	
4 Formalizing '*' and '***'	416	
5 Properties of FOL**	423	
6 Related work	434	
7 Concluding remarks	436	
Part XIII Computational Logic		
19		
Linear Time Datalog and Branching Time Logic <i>Georg Gottlob, Erich Grädel, Helmut Veith</i>		443
1 Introduction	443	
2 Modal and temporal logics	449	
3 From Datalog to Datalog LIT	451	
4 Datalog LITE	455	
5 Linear Time Algorithms	457	
6 Modal Datalog and CTLog	460	
7 Guarded fixed point logic	462	
8 Conclusion	464	

On the Expressive Power of Planning Formalisms <i>Bernhard Nebel</i>	469
1 Introduction	469
2 Propositional Planning Formalisms	471
3 Compilation Schemes	473
4 Compiling Conditional Effects Away	476
5 Compiling Boolean Preconditions Away	477
6 Compiling Boolean Preconditions into Conditional Effects	480
7 Summary and Discussion	486

Part XIV Knowledge Base System Implementations

Extending the Smodels System with Cardinality and Weight Constraints <i>Ilkka Niemelä and Patrik Simons</i>	491
1 Introduction	491
2 Weight Rules and the Stable Model Semantics	495
3 Application Methodology	508
4 Implementation Techniques	511
5 Experiments	513
6 Conclusions	516

Nonmonotonic Reasoning in <i>LCL⁺⁺</i> <i>Haixun Wang, Carlo Zaniolo</i>	523
1 Introduction	523
2 Monotonic Nondeterminism	525
3 Aggregates in Logic	530
4 Beyond Stratification	535
5 Conclusion	541

Part XV Applications of Theorem Proving and Logic Programming

Towards a Mechanically Checked Theory of Computation <i>J Strother Moore</i>	547
1 Prelude	547
2 Introduction	548
3 The ACL2 Logic	550
4 The ACL2 Theorem Prover	552
5 Elementary Examples	553
6 Merge Sort	556
7 Computing Machines	561
8 Nondeterminism and Distributed Computation	565
9 Industrial Applications	567
10 Conclusions	571
11 Acknowledgments	571

Logic-Based Techniques in Data Integration <i>Alon Y. Levy</i>	575
1 Introduction	575

2	Notation	576
3	Challenges in Data Integration	577
4	Modeling Data Sources and Query Reformulation	580
5	Answering Queries Using Views	582
6	The Use of Description Logics	589
7	Concluding Remarks	591
	Index	597

Contributing Authors

Chitta Baral, Associate Professor, Computer Science and Engineering, Arizona State University. His research interests include knowledge representation, non-monotonic reasoning, logic programming, and theory of agents. He received the NSF CAREER award in 1995. He was Co-Chair, NMR'2000. He received one of the best paper awards at ATAL'99.

Salem Benferhat, Chargé de Recherche, C.N.R.S., France. His research interests are in uncertain reasoning, possibilistic logic and possibilistic graphs, belief revision, preference modeling, data fusion, and independence relations.

Giuseppe De Giacomo, Assistant Professor, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza". His research interests include cognitive robotics, reasoning about actions, logics of programs, specification and verification of processes, concurrency, class-based formalisms, description logics, data modeling, database integration, and computational aspects of reasoning.

James Delgrande, Professor, School of Computing Science, Simon Fraser University. He has been the recipient of the Best Paper award for the American Association for Artificial Intelligence and Canadian Artificial Intelligence Conferences. From 1997 to 2000 he was Director of the School of Computing Science. Research interests include theoretical and implementational issues in nonmonotonic reasoning and belief revision.

Marc Denecker, Post-Doctoral Researcher, Department of Computer Science, Katholieke Universiteit of Leuven. His research interests focus on the logical and algebraic foundations of knowledge representation, and on abductive logic programming and its implementations.

Didier Dubois, Directeur de Recherche, C.N.R.S., France. His research areas are the modeling of uncertainty, knowledge representation and approximate reasoning, operations research and decision analysis. He co-authored two monographs and co-edited the 7 volume *Handbooks of Fuzzy Sets Series* (Kluwer). He is Co-Editor-in-Chief of *Fuzzy Sets and Systems*, and is on the editorial board of several journals.

Thomas Eiter, Professor, Computer Science, Vienna University of Technology, has been working in the fields of logic-based artificial intelligence and databases. His work laid strong emphasis on computational foundations, in particular algorithmic and complexity issues, which prove essential for the development of efficient knowledge representation systems such as the *DLV* system.

Wolfgang Faber, Research and Teaching Assistant at Vienna University of Technology, holds a Dipl.-Ing. (Master) in Computer Science and is currently working on his PhD. His research has focused on the implementation and application of logic-based intelligent systems. He is one of the key developers of *DLV* system.

Hélène Fargier, Chargé de Recherche, C.N.R.S., France. Her current research interests are in uncertainty and preference modeling, constraint satisfaction problems, qualitative decision theory, production research, information systems and operations research.

Hector Geffner, Professor, Computer Science, Universidad Simón Bolívar. He was co-winner of the 1990 ACM Dissertation Award. His research is in models of reasoning, planning, and learning, and in the development of high level tools for modeling and problem solving.

Michael Gelfond, Professor, Computer Science, Texas Tech University. His current research interests are in logic programming and artificial intelligence. He is an area Editor of the *Journal of Theory and Practice of Logic Programming* (Knowledge Representation and Nonmonotonic Reasoning) and an Executive Editor of the *Journal of Logic and Computation*.

Georg Gottlob, Professor Computer Science, Vienna University of Technology. His research is in complexity-theoretic and foundational problems in databases, logic programming, and knowledge representation. He was an invited speaker at several international conferences and chaired several program committees. He will serve as Program Chairman of IJCAI 2003. Gottlob is the winner of the 1998 Wittgenstein Award, and was elected Corresponding Member of the Austrian Academy of Sciences.

Erich Grädel, Professor Mathematical Foundations of Computer Science, RWTH Aachen (Aachen University of Technology), Germany. He works in mathematical logic and complexity theory. His main research interests include computational model theory, decidability and complexity issues in logic, database theory, verification and modal logics. He is co-author of a book on the classical decision problem for first-order logic.

Henry Kautz, Associate Professor, Department of Computer Science and Engineering, University of Washington, is known for his work on planning, satisfiability testing, temporal reasoning, and the complexity of nonmonotonic logic. He is winner of the IJCAI Computers and Thought Award, a Fellow of the AAAI, and Program Co-Chair of the AAAI-2000 national conference.

Nicola Leone, Professor, Computer Science, Vienna University of Technology, works in the areas of databases and artificial intelligence. His main research activities focus on nonmonotonic reasoning, deductive databases, algorithms and computational complexity. He was Program co-chair of LPNMR'99, and he is Guest Editor of the *AI Journal*. On the applied side, he supervised projects on advanced data and knowledge-based systems. He is the leader of the *DLV* project.

Hector Levesque, Professor, Computer Science, University of Toronto. In 1985, he received the Computers and Thought Award given by the International Joint Conferences on Artificial Intelligence. His research is in knowledge representation and reasoning in artificial intelligence. His research deals mainly with tractable reasoning and the relationship among the beliefs, goals, perception and action of autonomous rational agents. With Ray Reiter, he is Co-Director of the Cognitive Robotics Project at the University of Toronto.

Alon Levy, Assistant Professor, Department of Computer Science and Engineering, University of Washington. His research interests lie at the intersection of Artificial Intelligence and Database Systems, and include data integration, web-site management, knowledge representation and ubiquitous computing. He is a recipient of the Sloan Fellowship and an NSF CAREER Award. He is a co-founder of Nimble Technologies, a company providing distributed query processing for XML data.

Vladimir Lifschitz, Gottesman Family Centennial Professor, Computer Sciences, the University of Texas at Austin. Fellow of the American Association for Artificial Intelligence and a Member of the Editorial Board of the journal *Artificial Intelligence*.

Victor Marek, Professor, Computer Science, University of Kentucky. His research is in knowledge representation, nonmonotonic reasoning and logic programming. With Mirek Truszczyński, he is author of a research monograph *Nonmonotonic Logics: context-dependent reasoning* published by Springer-Verlag.

Flaviu Adrian Mărginean, Graduate student, University of York. He has an MS degree in Mathematics from the University of Bucharest and an MA degree in Pure Mathematics from the University of Southern California. He received an Overseas Research Student award by the UK government to undertake PhD studies in Computing Science at the University of York. His research is on Inductive Logic Programming.

Norman McCain, Assistant Professor of Computer Science, Baker University.

Torsten Schaub, University Professor, University of Potsdam, for knowledge processing and information systems. In 1999, he became Adjunct Professor at the School of Computing Science at Simon Fraser University. He authored a book on the Automation of Reasoning with Incomplete Information and co-authored a textbook on Knowledge Representation and Reasoning. His research interests range from the theoretic foundations to the practical implementation of methods for reasoning from incomplete and/or inconsistent information.

Lenhart Schubert, Professor, Computer Science, University of Rochester, conducts research in knowledge representation, computational semantics, and dialoguc systems. He was an Alexander von Humboldt Fellow (1978-9), has been program chair or co-chair for conferences such as the ACL and KR conferences, and is a Fellow of the AAAI.

Bart Selman, Associate Professor, Computer Science, Cornell University. His research includes: tractable inference, knowledge representation, natural language understanding, stochastic search methods, theory approximation, knowledge compilation, planning, default reasoning, and connections between computer science and physics (phase transition phenomena). He received four best paper awards at American and Canadian national AI conferences, and at the International Conference on Knowledge Representation. He holds an NSF CAREER Award and is an Alfred P. Sloan Research Fellow.

Murray Shanahan, Lecturer, Computer Science, Imperial College London, His research is in logic-based artificial intelligence. Much of his work has been on the topic of reasoning about action, but his most recent research has focussed on cognitive robotics. He is author of the book *Solving the Frame Problem*.

Patrik Simons, Completed his PhD in Computer Science and Engineering, at Helsinki University of Technology, in the spring of 2000. He is the author of *Smodels*, an algorithm and program for computing stable models of logic programs.

Armando Tacchella, Graduate student, University of Genoa.

Richmond H. Thomason, Professor of Philosophy, Electrical Engineering and Computer Science, and Linguistics, University of Michigan.

Miroslaw Truszczyński, Professor, Computer Science, University of Kentucky. His research centers on foundations of knowledge representation and on implementations and applicability of nonmonotonic reasoning systems. He is author, with Witold Marek of a research monograph *Nonmonotonic Logics: context-dependent reasoning* published by Springer-Verlag.

Helmut Veith, Assistant Professor Computer Science, Vienna University of Technology. His research focuses on applications of logic to computer science, in particular computer-aided verification, databases, fuzzy logic, and descriptive complexity theory.

Haixun Wang, Ph.D. student, Computer Science, University of California, Los Angeles. His research interests include database languages, nonmonotonic reasoning, data mining and knowledge discovery.

Carlo Zaniolo, Professor, Computer Science, University of California, Los Angeles. He holds the N.E. Friedmann Chair in Knowledge Science. His research interests include database systems, logic and AI.

Preface

The use of mathematical logic as a formalism for artificial intelligence was recognized by John McCarthy in 1959 in his paper on *Programs with Common Sense*. In a series of papers in the 1960's he expanded upon these ideas and continues to do so to this date. It is now 41 years since the idea of using a formal mechanism for AI arose. It is therefore appropriate to consider some of the research, applications and implementations that have resulted from this idea.

In early 1998 John McCarthy suggested to me that we have a workshop on *Logic-Based Artificial Intelligence (LBAI)*. In June 1999, the Workshop on Logic-Based Artificial Intelligence was held as a consequence of McCarthy's suggestion. The workshop came about with the support of Ephraim Glinert of the National Science Foundation (IIS-9820138), the American Association for Artificial Intelligence who provided support for graduate students to attend, and Joseph JáJá, Director of the University of Maryland Institute for Advanced Computer Studies who provided both manpower and financial support, and the Department of Computer Science. We are grateful for their support.

This book consists of refereed papers based on presentations made at the Workshop. Not all of the Workshop participants were able to contribute papers for the book. The common theme of papers at the workshop and in this book is the use of logic as a formalism to solve problems in AI.

The Workshop brought together a representative group of the major contributors who are using a logic-based formalism in AI. A formal approach to AI is important as it generally provides a clear semantics for the problem and one can then show that responses are sound (that is the answers are correct), and complete (all the answers are obtained). The use of a formal approach does not obviate the need for heuristics, nor does it imply that first-order logic is the only tool that can be used. There are many logics that may be appropriate to a particular problem, such as nonmonotonic reasoning, temporal logic, dynamic logic or modal logic, as well as non logical approaches. By having a formal approach one can, in most cases, determine the complexity of an implementation. That logic was used as the formalism does not imply that a logic language must be used to implement the system. It may be more efficient to implement the logic representation of the system in another language to gain efficiency.

Many sub-disciplines of AI are using a logic-based approach, and those in different aspects of AI had an opportunity to interact with others at the Workshop. The papers in this book represent work in: abductive reasoning, belief theory, commonsense reasoning, computational logic, high-level robotics, inductive learning, knowledge representation, logic for agents and actions, logic of causation, logic and language, logic and planning, logic, probability and decision theory, nonmonotonic reasoning, and theorem proving applications.

The workshop chairs were: John McCarthy and Jack Minker. The Program Committee for the Workshop selected the lecturers as well as the session chairs. The Program Committee consisted of: Krzysztof Apt, John Horty, Sarit Kraus, Vladimir Lifschitz, John McCarthy, Don Perlis, and Ray Reiter. Their advice and support were invaluable, and I greatly appreciate their assis-

tance. I also greatly appreciate the efforts of Juergen Dix and Parke Godfrey who were especially helpful with respect to organizing and obtaining the appropriate equipment for the system demonstration sessions. Cecilia Kullman of UMIACS worked closely with me to help organize the Workshop, coordinate with the hotel and on administrative matters regarding the Workshop. It would not have been possible to organize the Workshop without her support. V.S. Subrahmanian provided significant support for the Workshop, and Bart Selman helped to obtain AAAI funding to support student attendees. Their assistance is greatly appreciated.

The session chairs at the Workshop played an especially important role. They led the discussions of the papers and the more general topics under discussion. In addition, they wrote a summary of their session. I have made liberal use of their summaries in the introductory chapter to this book, Chapter 1. The session chairs were: Abductive and Inductive Reasoning, Francesca Toni; Applications of Theorem Proving, Don Loveland; Commonsense Reasoning, Leora Morgenstern; Computational Logic, Wittek Marek; Constraints, Jacques Cohen; Knowledge Representation, David Etherington; Logic and High Level Robotics, Fangzhen Lin; Logic and Language, Stuart Shapiro; Logic and Planning, Henry Kautz; Logic for Agents and Actions, Michael Fisher; Logic of Causation, Javier Pinto; Logic, Probability and Decision Theory, Faheim Bacchus; Theories of Belief, Melvin Fitting; Nonmonotonic Reasoning, Vladimir Lifschitz; and System Demonstrations, Juergen Dix. I am grateful to the Chairs for their work at the Workshop and for their informative reports.

The quality of the articles in the book has been improved substantially due to the referees. Each paper was refereed by two individuals and the editor. Every paper was modified and enhanced by the referee process. The referees were:

Krzysztof Apt, Faheim Bacchus, Chitta Baral, Howard Blair, Ronen Braffman, Luc DeRaedt, Juergen Dix, David Etherington, Michael Fisher, Maria Fox, Hector Geffner, Michael Gelfond, John Grant, Larry Henschen, John Horty, Neil Immerman, Henry Kautz, Nicola Leone, Hector Levesque, Alon Levy, Vladimir Lifschitz, Fangzhen Lin, Jorge Lobo, Bill McCune, Jack Minker, Leora Morgenstern, Bernhard Nebel, Ilkka Niemela, Rohit Parikh, Jeff Pelletier, Don Perlin, Jorge Pinto, Teodor Przymusinski, Rao Ramamohanarao, Ray Reiter, Carolina Ruiz, Stuart Shapiro, Len Schubert, Dietmar Seipel, Bart Selman, Olga Shumsky, VS Subrahmanian, Rich Thomason, Francesca Toni, Mirek Truszczynski, David S. Warren.

I am especially thankful to Frederick (Fritz) James McCall, who provided invaluable assistance with the details of LaTe_X, as did Bonnie Dorr, Parke Godfrey, Wittek Marek, Len Schubert, Dietmar Seipel, and Carl Smith.

I wish to thank my wife, Johanna, for her patience and understanding while I spent much of my time on this project.

The cover design for the book was done by my daughter, Sally Minker (<http://www.sallyminker.com>). I greatly appreciate her agreeing to take time from her busy schedule to do the art work and design for the book. Her colorful design captures the essence of the book. It has two robot-like figures whose heads are PC monitors with different illustrations on the screens. The

first shows a house, with a car leaving the carport. The second shows the car nearing an airport with a plane flying in the air. The robots are standing in front of a pattern of brains. The brains represent human intelligence and logic. The intent of the design is to evoke robots using commonsense reasoning to solve the “oldest planning problem in artificial intelligence,” – that of getting a person from home to the airport. It was defined by John McCarthy in 1959, who provided a partial solution to the problem. Several chapters in this book refer to this problem, while Chapter 7 by Lifschitz, McCain, Remolina, and Tacchella provide the most comprehensive solution to the problem to date.

JACK MINKER

I

INTRODUCTION TO LOGIC-BASED ARTIFICIAL INTELLIGENCE

Chapter 1

INTRODUCTION TO LOGIC-BASED ARTIFICIAL INTELLIGENCE

Jack Minker

*Department of Computer Science and
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742
minker@cs.umd.edu*

Abstract In this chapter I provide a brief introduction to the field of *Logic-Based Artificial Intelligence (LBAI)*. I then discuss contributions to LBAI contained in the chapters and some of the highlights that took place at the *Workshop on LBAI* from which the papers are drawn. The areas of LBAI represented in the book are: commonsense reasoning; knowledge representation; nonmonotonic reasoning; abductive and inductive reasoning; logic, probability and decision making; logic for causation and actions; planning and problem solving; logic, planning and high-level robotics; logic for agents and actions; theory of beliefs; logic and language; computational logic; system implementations; and logic applications to mechanical checking and data integration.

Keywords: Actions and agents, abductive reasoning, beliefs, commonsense reasoning, computational logic, inductive reasoning, knowledge base system implementations, knowledge representation, logic and causation in planning, logic and data integration, logic applications to mechanical checking , logic, planning and high-level robotics, natural language and logic. nonmonotonic reasoning, planning and problem solving, possibilistic logic.

1 INTRODUCTION AND BACKGROUND

As noted in the *Preface*, the chapters that appear in this book are an outgrowth of papers presented in June, 1999, at a *Workshop on Logic-Based Artificial Intelligence (LBAI)*, held in Washington, D.C. A short report was written that provided a brief synopsis of the lectures and demonstrations presented (Minker, 1999b). The purpose of the Workshop was to bring together

researchers who use logic as a fundamental tool in *AI* so as to permit them to review accomplishments, assess future directions, and share their research in LBAI. In this chapter I provide some background on LBAI, discuss its significance, and introduce the chapters in the book.

The parts of the book that follow this introduction are, in sequence, commonsense reasoning; knowledge representation; nonmonotonic reasoning; logic for causation and actions; planning and problem solving; logic, planning and high-level robotics; logic for agents and actions; abductive and inductive reasoning; logic, probability and decision making; theory of beliefs; logic and language; computational logic; system implementations; and logic applications to mechanical checking and data integration. There are other areas of LBAI not represented here, such as logic and constraints. It was not possible to cover all subjects in LBAI.

Why a logic-based approach to artificial intelligence (AI)? The attendees at the Workshop had a common vision that an appropriate and important approach to AI is to provide a clear formulation of the problem in logic. This does not mean that other approaches are not relevant, but that formulating AI problems in logic provides certain advantages: a clear declarative semantics of the problem, the ability to implement problems formulated in this manner in a logical language, and to implement AI problems rapidly. This neither precludes implementing a final version of a problem in a lower level language to gain efficiency, nor implies that there is no need to have hybrid approaches including non-logic implementations.

It is not possible to provide here a comprehensive history of the field of LBAI. In the following section, I provide a short introduction to some of the major events that have taken place in the field.

2 BACKGROUND

The field of LBAI started in 1958 with the appearance of John McCarthy's paper (McCarthy, 1959). LBAI owes much to McCarthy's leadership. McCarthy recognized that if one were to develop an intelligent artifact, then one has to formalize what is meant by *commonsense reasoning (CSR)*. In 1963, McCarthy (McCarthy, 1963) discussed the issues involved in situations, actions, and causal laws that enter into commonsense reasoning. Together with Hayes (McCarthy and Hayes, 1969), he discussed some of the philosophical problems in AI and the *frame problem*, the seeming need for large numbers of axioms, termed *frame axioms*, to represent changes in the situation calculus. He also discussed epistemological problems in AI (McCarthy, 1977).

The development of the modern approach to automated theorem proving (ATP) is another milestone in LBAI. In 1965, J. Alan Robinson (Robinson, 1965) wrote a seminal paper that described a single rule of inference, the *Robinson resolution principle*, to perform deduction in first-order logic. The rule of inference uses the *unification algorithm* to determine if two predicates can be made equal by some substitution. Many refinements have been proposed to the method developed by Robinson. See (Chang and Lee, 1973) for a description

of many of these inference systems, and (Loveland, 1978) for a comprehensive theoretical description of automated theorem proving.

Since 1965, there has been much work done to prove theorems in mathematics. However, automated theorem proving is difficult and it is not easy to obtain proofs of interesting theorems by an automated system. There have been only a few successes until recently, when McCune (McCune, 1997), solved an open problem posed first by Dr. Herbert Robbins, a Professor at Rutgers University, in the 1930s. McCune's program, *EQP, Equational Prover*, designed to reason, not to solve a specific problem, proved that "Every Robbins algebra is Boolean." In this book, Moore in Chapter 23 successfully applies theorem proving to mechanically check hardware. These developments have been given renewed impetus to automated theorem proving. See the report by Loveland (Loveland, 1999) on this topic.

Robinson's work was the basis for several other areas: *Logic Programming (LP)*, *Deductive Data Bases*, and *Constraint Logic Programming*. Hayes (Hayes, 1973a), discussed the relationship between logic and computation.

The field of LP started with work by Alain Colmerauer and Robert Kowalski. Colmerauer (Colmerauer et al., 1973) together with his students developed *Prolog*, a logic-based language. Kowalski (Kowalski, 1974) showed that *Horn clauses*, a subset of clauses needed for fully automated theorem proving, form the basis of *Prolog*, and are suitable as a programming language, particularly for AI applications. Together with *Lisp*, a symbolic manipulation programming language, invented in the 1960's by McCarthy (McCarthy, 1978), based on the lambda calculus, *Prolog* is the language of choice for AI applications.

van Emden and Kowalski (van Emden and Kowalski, 1976) developed three different semantics for Horn *LP* (declarative, procedural, and fixpoint), and showed that they were equivalent. *LP* has been extended substantially since the inception of the field. Clark (Clark, 1978) introduced the supported model semantics for logic programs. His construction first translates logic programs to first-order logic to obtain the only-if rule part of the if and only if completion rule. This has been investigated by Apt and van Emden (Apt and van Emden, 1982) who have shown the fixpoint interpretation of his approach. Clark's approach is closely related to the autoepistemic logic of Moore (Moore, 1985); as can be shown, using a similar mechanism for (possibly circular) beliefs to provide the semantics to logic programs. The need was then recognized to permit *LP*'s to have *default negated atoms* in the body of a clause. The first approach was to limit default negation so that atoms negated were free of recursion. This led to *stratified negation* (Apt et al., 1988; Van Gelder, 1988; Przymusinski, 1988; Chandra and Harel, 1985), and the *stratified semantics*. A second development was to permit recursion with default negation. This led to many different semantics, the most important of which are the *stable model semantics* (Gelfond and Lifschitz, 1988), and the *well-founded semantics* (Van Gelder et al., 1988). The stable model semantics is referred to as yielding answer sets, coined by Lifschitz (Lifschitz, 1987). It captures the main idea behind the approach: solutions or answers are represented as sets of objects. Stable model semantics treats negation in the bodies of clauses as 'negation by default' and is closely related to Reiter's Reiter, 1980 default logic. A third

development was to permit literals, either atoms or classical logically negated atoms, to appear in the head of clauses. Literals and default negated literals could then appear in the heads and body of clauses (Gelfond and Lifschitz, 1990). All of this work was extended to the case of disjunctive logic programs which permit disjunctions of atoms and/or literals in the head of clauses (Lobo et al., 1992; Minker, 1994). Brass et al. (Brass et al., 1996), have developed an interesting semantics for disjunctive logic programs that relates to the well-founded semantics for normal logic programs and has several nice properties. The use of the *LP* formalism was recognized as useful for reasoning by *abduction*, a form of synthetic inference. Given a set of observations, the abductive task is to find an explanation so that, from the theory and the explanations, one can derive the observations, (Kakas et al., 1993).

Research in deductive databases was started first by Green and Raphael (Green and Raphael, 1968a; Green and Raphael, 1968b). The development of *LP* led to the understanding by Gallaire, Minker and Nicolas (Gallaire and Minker, 1978; Gallaire et al., 1984), that *LP* generalized the concept of *relational databases*. This led to the field of deductive databases and to the more general area of *disjunctive deductive databases*. Relational databases now incorporate many developments in deductive databases. See (Minker, 1988; Minker, 1996) for perspectives on the fields of deductive and disjunctive deductive databases. Zaniolo et al. (Zaniolo et al., 1993) use stable model semantics to give a characterization of aggregates in databases. They showed that aggregates can be captured in logic programming using the stable model semantics. Previous work on giving semantics to aggregates introduced new constructs and new semantics. Their paper showed no new constructs are needed. Work in *LP*, deductive, and disjunctive deductive databases is related to the problem of *knowledge representation (KR)*. As shown by Baral and Gelfond (Baral and Gelfond, 1994), using a logic formalism permits a wide class of problems in *KR* to be represented and reasoned with.

Constraint Logic Programming is a natural extension of *LP* in which one can define new domains and predicates and retain the notion of resolution as a form of procedure calls. Historically, the first new domain to be considered was the domain of infinite trees developed by Colmerauer in Prolog II (Colmerauer, 1985). The built-in predicates he considered were equality and disequality of infinite trees. See Jaffar and Maher, 1994 for a comprehensive survey of this field.

One of the critiques of logic as a tool for AI was that it could not represent commonsense reasoning as applied by humans (Minsky, 1975). Minsky noted that in first-order logic, something that is a *consequence* continues to be a consequence when new truths are added to the theory. However, when humans reason, they sometimes retract things that are considered consequences when more information is found that contradicts the so-called consequence. When humans reason, they sometimes jump to conclusions based on non-logical reasoning. The principle used in many instances is *if it is not known or cannot be proven that a certain condition pertains then assume it is false*. Human reasoning is thus, *nonmonotonic* in that consequences are not always preserved as they are with first-order logic.

Nonmonotonic reasoning was developed to overcome objections to the inability of first-order logic to handle commonsense reasoning. Three major logical approaches have been developed to reason nonmonotonically. Articles on two of these approaches appeared at the same time in a seminal issue of the *Artificial Intelligence Journal*, Volume 13, Number 1 and 2, 1980. These approaches are called: *circumscription*, developed by McCarthy (McCarthy, 1980), and *default reasoning*, developed by Reiter (Reiter, 1980). A third approach, *autoepistemic logic*, was developed by Moore (Moore, 1985). See (Marek and Truszczyński, 1993) for a theoretical treatment of autoepistemic logic and nonmonotonic logic.

A second limitation of logic was recognized in applications to real problems. This is the so-called *frame problem*. The problem arises in the situation calculus when applied to reasoning about actions. There appeared to be needed an unavoidable number of axioms, called *frame axioms*, that described what would hold in a succeeding state after an action was taken, given the current state (McCarthy and Hayes, 1969; Hayes, 1973b). Several solutions have been proposed for the frame problem, even for the difficult case of actions with indirect effects (Reiter, 1991; Shanahan, 1997). The long history of the frame problem involved the investigation of many natural possibilities that turned out to be dead ends. Remarkably, the uses of nonmonotonic reasoning in the available solutions are computationally easy: they can be performed by means of simple syntactic translations into monotonic logic (first-order, or even propositional) that are similar to the *completion method* (Clark, 1978) familiar from the theory of logic programming. Nonmonotonic reasoning of this kind can be often automated. In addition, the three forms of nonmonotonic reasoning can be translated, in most cases of interest, to logic programs (Minker, 1993; Cadoli and Lenzerini, 1991; Cadoli and Schaerf, 1992). By translating the theories to logic programs, one can easily determine the complexity of the problem as there are numerous results in complexity of logic programs available as presented in (Dantsin et al., 1997; Cadoli and Schaerf, 1992; Cadoli and Lenzerini, 1991; Minker, 1993).

The oldest problem in AI planning, the ability of an agent to achieve a specified goal, was defined by McCarthy (McCarthy, 1959). He developed a logic-based approach to the problem. McCarthy's solution was not entirely satisfactory (see Chapter 7). Green (Green, 1969), was the first to formalize the planning problem as a deductive problem in logic and showed how to extract a plan from the logic representation. He used the situation calculus. His approach to actions and their effects are axiomatized for a given domain of applications. The task for a given goal was to find a sequence of actions such that the axioms entail the goal. Fikes and Nilsson (Fikes and Nilsson, 1971), developed a system called *STRIPS*, the first general approach to planning, designed as the planning component of a robot *Shakey* (Nilsson, 1984). See Lifschitz (Lifschitz, 1987) for a careful critique and formal analysis of *STRIPS*, (Fikes and Nilsson, 1993) for a historical retrospective, and (Russell and Norvig, 1995) for specific details on *STRIPS* and other approaches to planning.

One of the problems with the early approaches to the logical approach to planning was the inability to handle the frame problem, and the inefficiency of theorem provers at that time. Because of this, there was a move away from

using logic to build working robots to more practical, but *ad hoc* approaches. This culminated in the sense-model-plan-act architecture of Brooks, 1991.

Developing planning in the context of robotics has led to the field of "cognitive robotics", a term coined by Reiter in the early 1990s. Cognitive robotics is about endowing a robot with "higher level cognitive functions" to reason about goals, actions, beliefs, perceptions, and others. This is to be contrasted with more traditional robotics work that is concerned mostly with basic level tasks such as path planning and manipulator control. Cognitive robotics is one of the research areas in LBAI that has seen progress in the last decade. The research in the 90s in this area has had, and is having, an impact in many diverse areas such as: theory of agents, reactive control, database updates, active databases, model based diagnosis, and planning using model generation. Sandewall's book, (Sandewall, 1995) discusses many of the issues in cognitive robotics and presents an approach to handling many of the issues in this field.

As noted by Shanahan in Chapter 11,

... the Shakey style of architecture, having an overtly logic-based deliberative component, seems to offer researchers a direct path to robots with high-level cognitive skills, such as planning, reasoning about other agents, and communication with other agents. Accordingly, a number of researchers have instigated a Shakey revival, and are aiming to achieve robots with these sorts of high-level cognitive skills by using logic as a representational medium Lespérance et al., 1994.

The critique concerning the use of the situation calculus for planning actions was focused by McDermott and Hanks on the Yale-Shooting Problem (Hanks and McDermott, 1987). McDermott and Hanks challenged the use of nonmonotonic logic to solve the frame problem. Several works on temporal logic for AI (Allen, 1984) challenged the adequacy of the situation calculus to represent richer domains such as concurrent actions, and continuous actions.

Three major developments in 1991 changed the status-quo. *First*, Gelfond, Rabinov and Lifschitz (Gelfond et al., 1991) challenged the critics of the situation calculus and discussed how to formalize several aspects of reasoning about actions that the critics said were not formalizable in the situation calculus.

Second, the high-level language approach to formalize reasoning about actions was introduced by Gelfond and Lifschitz (Gelfond and Lifschitz, 1992; Gelfond and Lifschitz, 1998). Their approach was to develop a simple English-like language with a precise automata based semantics to represent and reason about actions.

Third, Reiter and his colleagues at the University of Toronto clarified and formalized matters. (i) Reiter (Reiter, 1991) gives a solution of the frame problem using earlier work by Pednault (Pednault, 1989), Haas (Haas, 1987), and Schubert (Schubert, 1990), where he shows how to develop successor state axioms for reasoning about actions and their effects, under certain assumptions. (ii) Reiter (Reiter, 1993) gives second order axioms that allow specifications as to what is a situation and what is not, and statements of the form that a certain property is true in *all* situations. (iii) Reiter et al. (Jenkin et al., 1997; Lespérance et al., 1994; Levesque et al., 1997) then propose the Golog language to specify complex plans. Golog lets the user specify a plan in a non-

deterministic language and the Golog interpreter resolves the non-determinism either before execution or during the execution. Golog is built on top of the situation calculus and allows a rich action language.

The general discussion at the Workshop centered on several questions. I cite three such questions posed by Melvin Fitting here, and the discussion on these questions. The first question was, "As logics become more specialized, they often become more formidable syntactically and conceptually. Does this limit a logic-based approach?" More briefly, "Can logics be too complex to be used?"

There was general agreement that the answer was "yes," but this was not seen as a limitation on potential uses of logics, but rather on their misuse. It was said that, if a logic-based approach is too complex, it is because the subject hasn't been properly analyzed and understood. This was not seen as an argument that all areas are inherently simple. Rather, it reflected a general belief that at the heart of complex phenomena, as the basis of understanding, must lie simple ideas. The phenomena are what they are; the formalization, the ontology, come from us. The consensus of the discussion on this point was: a logic-based approach that is too complex is evidence that the subject has not been understood at a deep enough level.

A second question discussed was, "There are modal logics, temporal logics, logics of action, etc. Will we ever see a 'modular' approach, where a logic can be tailored to an application by combining off-the-shelf items?" Briefly, "Can logic be modular?" It is Fitting's opinion that, in fact, combining disparate logics will always remain a difficult topic. Fitting and others are working on the relationship between quantified classical logic (including higher-order quantification) and modal logic. Each separately is well-understood. The combination, however, presents a number of complexities of which people have generally been unaware. This is not a drawback. The combination is richer than had been supposed and so potential applications are more, rather than less, likely.

The final question for discussion was, "Newtonian physics is to mechanical engineering as formal logic is to what?" Schematically:

$$\frac{\text{Newtonian Physics}}{\text{Mechanical Engineering}} = \frac{\text{Logic}}{?}$$

It provoked the least commentary. John McCarthy simply announced that the answer was, "Computer Science." There seemed to be nothing more to say.

In the remainder of this chapter I discuss the individual parts of the book, the core issues addressed by the authors in their chapters, and issues raised at the Workshop.

3 COMMONSENSE REASONING

Commonsense reasoning refers to the practical reasoning that we use daily when we act in the real world and is a crucial part of intelligent behavior. It is difficult to formalize intelligence, or to build a truly intelligent artifact, without formalizing CSR.

There is one paper in the book on CSR. McCarthy discusses logical AI and explores the concepts that are required. Some concepts are new while others have been explored previously. He references the literature on each concept.

This is an important contribution and permits others to pursue the individual topics. At the Workshop, McCarthy outlined a set of problems to be solved and issues to be addressed as part of research in commonsense reasoning. These ranged from general philosophical issues (such as the development of ontologies and the formalization of consciousness) and methodological issues (formalizing approximate theories, handling approximate concepts, and making theories elaboration tolerant), to specific formalisms (nonmonotonic reasoning, the situation calculus), and domain problems (the frame problem, the ramification problem).

McCarthy's talk served both as a celebration of what has been achieved and a wake-up call for the vast amount of work that still needs to be done. It is at once encouraging to think of the progress that has been made in the last forty years — the development of the situation calculus and other powerful temporal reasoning languages, the development of nonmonotonic reasoning — and sobering to think of the many crucial areas of commonsense reasoning — the representation of physical objects, narration, and ambiguity tolerance — which have hardly been studied from the standpoint of LBAI.

At the Workshop, Sandewall suggested a methodology to address these issues. He argues that the criteria for evaluating research results in CSR ought to be based on an idealized *representation chain*, which leads from an informal understanding or description of a domain to an implemented system. Ideally, one would begin one's research on a particular domain (e.g. furniture building, which includes aspects of physical and spatial reasoning as well as planning) with informal intuitions about the domain. Next, one would build a *microworld* of the domain, proceed to a *standard model* of the domain, and then go on to a *high-level language account* of the domain, followed by a *primary-level language account* of the domain (an example of a primary-level language could be first-order logic or a logic-programming language). The final output would be a software system operating in or for a domain.

Sandewall emphasized the importance of the microworld modeling step, which has also been recently urged by Davis (Davis, 1998; Davis, 1999). He pointed out that it is this very step that distinguishes this methodology from the axiomatic approach of Hayes (Hayes, 1985a; Hayes, 1985b). The axiomatic approach is primarily concerned with writing down logical axioms that capture intuitions about a particular domain. It is precisely the inclusion of the modeling step in the approach favored by Sandewall and Davis that facilitates the evaluation of a particular theory, and the comparison among multiple theories. One can, for example, more easily evaluate how closely an axiomatization captures a particular microworld than one can evaluate how closely an axiomatization captures some set of intuitions.

4 KNOWLEDGE REPRESENTATION

Knowledge Representation is the study of how to represent knowledge in a form that a computer can reason with (Russell and Norvig, 1995). Alternative approaches have been taken to represent knowledge: *frames* (Minsky, 1975), *scripts* (Schank and Abelson, 1977), *semantic networks* (Quillian, 1968), and

predicate calculus logic as discussed in (Nilsson, 1984). For bibliographic and historical remarks on approaches to *KR*, see (Nilsson, 1982) and the conferences on *KR*.

Two chapters in the book are devoted to *KR*. Levesque and De Giacomo, Chapter 3, explore two ways of using expressive logical representation languages without necessarily sacrificing tractability. Eiter et al., Chapter 4, discuss representing and solving problems in the *DLV* system, based on disjunctive logic programming, using a nondeterministic “guess and check” paradigm.

Levesque’s “two approaches” are to use a so-called “vivid” representation of the knowledge base, extended to allow the use of definitions, explanation closure axioms, and query regression through successor state axioms, to enable provably efficient reasoning using representation languages that are intractable in the general case, and allowing sensing operations to eliminate disjunctive ambiguities at “execution” time, effectively avoiding the combinatorial explosion inherent in the use of representations expressive enough to represent incomplete knowledge about the world. Levesque’s approach is appealing in that it allows the use of the sorts of very expressive representations necessary to adequately capture the incompleteness of typical knowledge about the world while still providing formal guarantees of performance under certain conditions that will, in many situations, likely be completely reasonable.

The *DLV* system of Eiter et al., takes a different approach to intractability. Instead of attempting to guarantee performance under restrictive assumptions, it relies on the fact that it is often easy to find solutions to instances of hard problem classes, and employs a so-called “guess and check” inference algorithm. This algorithm exploits heuristic information to construct (guess) a solution, which it then verifies against the given problem constraints. In the event that the constraints turn out not to be satisfied, another guess must be made. Considerable success was achieved with this approach in quickly solving NP-complete problems such as finding Hamiltonian paths. This paradigm appears to indicate that the effectiveness of “iterative sampling” approaches (Cholewiński et al., 1999) (following a single, possibly heuristically-guided “probe” through the search tree) discovered in the constraint satisfaction literature carries over to logic-based approaches such as disjunctive logic programming. See Section 15 for additional material on *DLV*.

Both chapters are indicative of the devotion of serious attention to computational as well as expressivity and derivability concerns in the construction and study of representational frameworks. In McCarthy’s terms (McCarthy, 1977), heuristic adequacy is taking its place alongside epistemological adequacy in the study of *KR*.

5 NONMONOTONIC REASONING

As noted in Section 2, *nonmonotonic reasoning*, was developed in the late 1970s to overcome the objection that first-order logic could not deal with *CSR*. Two chapters in the book represent current work on *NMR*. Delgrande and Schaub, Chapter 5, argue that default logic is more general and more widely applicable than previously supposed. They show how it can be used to express

preferences among defaults by means of naming default rules and encoding conditions for their invocation.

Denecker, et al., Chapter 6, develop an algebraic approach to the semantics of nonmonotonic logics. Their work is based on Fitting's theory of logic programs with negation as failure (Fitting, 2000) and generalizes that theory to other fixpoint formalisms. They show that their approach encompasses all major semantics for logic programming, autoepistemic reasoning and default logic in a uniform way by applying their algebraic fixpoint theory for a particular operator. They hypothesize that their theory provides a generalized algebraic account of non-monotone constructions and non-monotone induction in mathematics.

It has been shown that many *NMR* theories can be implemented in logic programming. There have also been many semantics generated for dealing with default negation in the body of logic programs. These semantics, in many instances, permit the *NMR* theories of circumscription, default reasoning and autoepistemic logic to be implemented (see, for example (Minker, 1993; Cadoli and Lenzerini, 1991; Cadoli and Schaerf, 1992)). Among these semantics, there are two that stand out as being particularly useful: the *stable model semantics* (Gelfond and Lifschitz, 1988), and the *well-founded semantics* (Van Gelder et al., 1988). A major need now exists to sort out the various semantics proposed to determine which are the most effective to use. This may be more important than the development of additional semantics. As noted in Section 15, there are a number of systems that have been developed that can be applied to *NMR* problems (*Smodels*, *DeReS*, *DLV*, *XSB*) (Niemela and Simons, 1997; Cholewiński et al., 1996; Eiter et al., 1997; Rao et al., 1997; Warren, 1999). Although some of these systems handle large sets of data and rules, applying the current systems to real, large, significant problems has yet to be achieved and is needed to demonstrate the effectiveness of current implementations and their use for *NMR*.

6 ABDUCTIVE AND INDUCTIVE REASONING

Abductive and *inductive reasoning* are two of the three distinguished forms of human reasoning identified by the philosopher Peirce (Peirce, 1883). *Deduction* is an analytic process based on the application of general rules to particular cases, with the inference of a result. *Induction* is a form of synthetic reasoning which infers the rule from the case and the result. *Abduction* is another form of synthetic inference, but of the case from a rule and a result. In inductive reasoning, the observations are understood as examples of the concept to be learned.

One of the earliest approaches to inductive reasoning was by Plotkin (Plotkin, 1971; Plotkin, 1969), who showed the relationship of logic programming and inductive reasoning. A number of approaches have been proposed to understand the theoretical foundations of logic-based abductive and inductive reasoning, e.g. see (Kakas et al., 1993; Dimopoulos and Kakas, 1995). Moreover, the relationship between logic-based abductive and inductive reasoning

as well as their integration within a unified framework has been studied extensively, e.g. see the *Journal of Logic Programming*, volume 44, numbers 1-3, July/August 2000 which contains a special issue on abductive logic programming.

One chapter in this book is devoted to abductive and inductive reasoning, Chapter 14. Muggleton and Marginean provide a survey of work accomplished primarily in inductive reasoning. They discuss several languages that have been used for inductive logic programming and their application to drug problems and natural language understanding. They show the results of various programs used for learning problems. Evidence exists that inductive logic programming has higher accuracies than other approaches.

7 LOGIC, PROBABILITY AND DECISION MAKING

Intelligent agents must be able to deal with uncertainty: in general they will have neither perfect knowledge of nor perfect control over their environment. We can distinguish between uncertainty that arises from simple lack of knowledge, and that which arises from noise inherent in an agent's sensors and effectors (i.e., noise in an agent's "interface" with its environment).

Lack of knowledge, or incomplete knowledge, is handled relatively easily by standard logical theory: incomplete knowledge has the simple consequence that fewer things can be deduced. Complications arise when an agent wants to conclude things that do not follow from, but are not contradicted by, its knowledge, as in nonmonotonic logics. Noise, on the other hand, requires other tools. Probability and decision theory remain the best of these tools.

This is not to say that probability and decision theory are incompatible with logic. They are not. However, it is to say that it is unlikely we can dispense with probability and decision theory. Rather, integration seems to be what is required.

One chapter is devoted to this subject. Prade et al., Chapter 15, develop a logic for making decisions based on qualitative information about preferences and likelihoods (where the likelihoods are modeled using possibility theory). The logic is a propositional one, but shows how some of the intuitions underlying decision theory can be embedded in a logical language. Possibilistic logic deals with classical logic formulas which are associated with grades of uncertainty which are lower bounds of a so-called "necessity measure" (which is only compositional w.r.t. conjunction). Possibilistic logics differ from Fuzzy logics (there are several varieties) which are generally multiple-valued logics with non-classical formulas which, at the semantical level, may have intermediary degrees of truth. These degrees of truth are compositional w.r.t. all connectives, using connective functions which depend on the particular logic. Fuzzy logics are not suitable for the modeling of uncertainty and incomplete information, but may be of interest for handling gradual properties in case of complete information.

The embedding of probabilities into logics has a long tradition in AI (Bacchus et al., 1993), the basic idea being to provide syntactic constructs suitable for expressing useful probabilistic information, and semantic constructs for provid-

ing a probabilistic interpretation for these syntactic constructs. See the book by Bacchus (Bacchus, 1990), for early work in this area and for various first-order logics for probability. Boutilier et al., (Boutilier et al., 2000), describe how to handle decision-theoretic problems in high-level agent programming in the situation calculus.

8 LOGIC FOR CAUSATION AND ACTIONS

The development of theories of action and change has always been at the core of *KR* reasoning. Causality is considered essential to model the interplay between actions and the change they provoke on the world.

There has been a good deal of work in the area of causation. I briefly note some of the more recent work. Geffner (Geffner, 1990) studied causal theories for nonmonotonic reasoning. Gelfond and Lifschitz (Gelfond and Lifschitz, 1993) studied how to represent action and change by logic programs. Giunchiglia and Lifschitz (Giunchiglia and Lifschitz, 1998) discuss an action language based on causal explanation. Lin (Lin, 1995) shows the utility of causality in specifying the indirect effects of actions. McCain and Turner (McCain and Turner, 1997) discuss causal theories of action and change.

One chapter in the book is devoted to causation. Lifschitz et al., Chapter 7, show how one can solve "the oldest problem in planning", using a causal-based approach. They state, "The discovery of the frame problem and the invention of the nonmonotonic formalisms that are capable of solving it may have been the most significant events so far in the history of reasoning about actions." They used the Causal Calculator (CCALC) to implement the planning problem. The CCALC system for handling the frame problem is related to the method of Reiter, 1980. They state that, "Claims to the opposite Hanks and McDermott, 1987 notwithstanding, Reiter's solution turned out to be completely satisfactory after the rest of the postulates were formulated in the right way". At the Workshop, Lifschitz discussed the interaction between three lines of research on LBAI: (1) the design of languages for describing actions; (2) the analysis of causal reasoning; and (3) the use of propositional solvers for planning. He described the development of his work based on these lines of research. Chapter 7 illustrates this work.

From a more technical point of view, an important discovery, is that dynamic causal laws (those describing the effects of actions) and static causal laws (that describe causal relationships between fluents in the same state) are governed by the same logic. This can be expressed with causal rules in specialized action languages (c.g., along the lines of the language developed by Gelfond, Lifschitz and associates (Gelfond and Lifschitz, 1998)) or in languages in which causality is introduced with a modal operator. This is related to the discovery of the different roles that constraints may play in a theory of action. This discovery, due to Ginsberg and Smith (Ginsberg and Smith, 1988a; Ginsberg and Smith, 1988b), is that constraints can be used to express qualifications on the executability of actions, and also to express indirect effects, or ramifications, of actions.

An important conclusion of the Workshop was that research conducted in this subfield of LBAI is at a level of maturity in which applications are beginning to emerge. This is witnessed by new planning frameworks and systems and a variety of applications in cognitive robotics, as discussed in Section 15. Also, advancements on theorem proving research (e.g., in propositional provers) is having an impact on the applicability of research on actions.

9 PLANNING AND PROBLEM SOLVING

In recent years, there has been a resurgence of interest in implementing planning in logic due to the empirical success in using new systematic or local search algorithms to solve propositional encodings of bounded-length planning problems, and to contributions to the frame problem.

There are two approaches that represent this new work. In the first, conjunctive-normal-form (CNF) propositional logic is directly used as a modeling language for a planning domain (Kautz and Selman, 1999b). In the second, CNF is replaced with a more compact data structure called a plan graph (or what is called in the program verification community, an unfolded Petri net). The two approaches can be combined by performing preliminary inference using a plan graph, then converting the graph into CNF before the final solution extraction phrase (Kautz and Selman, 1999b). The emphasis of the general approach is to trade space for speed by dealing only with propositional structures and eliminating variable binding during combinatorial search.

Several chapters in the book are devoted to planning and logic. Here, we discuss the work by Kautz and Selman, Chapter 8, and by Geffner, Chapter 9. In Section 8, we discussed the work by Lifschitz et al., Chapter 7, in Section 10, we discuss the work by Reiter and Pirri, Chapter 10, and by Shanahan, Chapter 11, and in Section 14, we discuss the work by Nebel, Chapter 20. Baral and Gelfond, Chapter 12 provide an interesting discussion of several approaches that use logic as a basis for agent design.

Kautz and Selman, Chapter 8, describe their experiments with *SATPLAN*, possibly the fastest planning system for domain-independent planning. They present a way to encode domain knowledge in a declarative, algorithm independent manner. They show that the same heuristic knowledge can be used by entirely different search engines. They describe the success of the SATPLAN framework in encoding planning problems in SAT and solving them with stochastic search algorithms. In particular, they describe how adding a small amount of randomness to a Davis-Putnam-Loveland type backtracking algorithm (Davis and Putnam, 1960) and periodically restarting it if no solution is found within a time limit, can dramatically reduce the expected solution time by eliminating very long runs. Their approach to heuristics greatly enhances the power of SATPLAN so that solution times for some problems are reduced from days to seconds.

In Chapter 9, Geffner describes a language, Functional Strips, that adds first-class function symbols to Strips. Functional Strips is both an action language and a planning language. His work is motivated by the view that planning is general problem solving. He illustrates the work with several classic AI

problems. Extensions are planned to include the A-language of Gelfond and Lifschitz (Gelfond and Lifschitz, 1993; Gelfond and Lifschitz, 1998) and others.

Although propositional planners can handle much larger problems than general AI planners could a few years ago, they do not currently scale to large domains (which can only be handled today by knowledge-intensive methods that avoid search). The more likely near-term practical applications involve medium sized domains where optimal or near-optimal planning is required. Nevertheless, the accomplishments of current planners as evidenced by those discussed in Section 15 are impressive.

10 LOGIC, PLANNING, AND HIGH LEVEL ROBOTICS

Two chapters in the book are devoted to cognitive robotics with cognitive skills that use logic. Reiter and Pirri, Chapter 10, and Shanahan, Chapter 11. Reiter and Pirri describe the theory and implementation of a deductive planner in the situation calculus with two kinds of actions: "free will" (where agents may perform or withhold their actions), and natural actions whose occurrence times are predictable. The planner, based on the work of Bacchus and Kabanza (Bacchus and Kabanza, 2000), is an extension of the situation calculus to accommodate continuous time and natural actions. They illustrate their work on a space platform example. The work relates to the system Golog, on cognitive robotics, some of whose features are described in Section 15.

Shanahan describes the logical foundations of an implemented system which employs resolution-based theorem proving techniques for high-level robot control. The chapter offers complementary logical characterizations of perception and planning, and shows how sensing, planning and acting are interleaved to control a real robot. The chapter describes one such robot and concentrates on logical foundations. A miniature robot, Khepera, with two drive wheels and a set of eight infra-red proximity sensors around its circumference was implemented to test the theory. He uses a combination of the event calculus and abductive reasoning in his work. He notes that there are two main differences between contemporary cognitive robotics and Green's work. First, there are now satisfactory solutions to the frame problem. Second whereas Shakey relied on full search-based planning, the approach he presents uses chiefly pre-compiled plans - programs.

As Reiter noted at the Workshop, cognitive robotics is "much more than the search for a theory for robotics; it is about dynamics in general." A general theory of dynamic systems will undoubtedly have a far reaching impact on AI and computer science in general. The two chapters illustrate the use of logic for cognitive robotics and further demonstrate the resurgence of the use of logic for implementing practical systems. Additional work is required to handle cognitive robotics in realistic environments.

11 LOGIC FOR AGENTS AND ACTIONS

Several approaches for dealing with actions have been described in the book. Baral and Gelfond, Chapter 12, use a logic programming approach. Lifschitz et al., Chapter 7, develop a causal approach. Reiter and Pirri, Chapter 10, use the situation calculus as the model for their work. Shanahan, Chapter 11, uses a combination of the event calculus and the abductive event calculus. Other approaches, not discussed in this book, include alternative modal logics, and the situation calculus.

One chapter of the book is devoted to logic for agents and actions. Meyer, Chapter 13, uses *dynamic logic*, a logic to reason about the dynamics of (natural or artificial) systems in general, ranging from the effects of actions of human agents to the behavior of artificial agents and software systems. Meyer uses dynamic logic both to represent knowledge about the dynamics of a domain as well as to describe/specify (the dynamic behavior of) AI systems themselves. A typical example of the former is the description of the effects of actions (of humans, for example) in the commonsense world, while the specification of a particular reasoning system would be of the latter type. Meyer provides a number of examples to illustrate the usefulness and wide scope of dynamic logic for AI.

Although agents may be *able* to perform actions, be they physical or communicative, it is often useful to add some restriction to this ability, for example defining when an agent is *permitted* to carry out an action. For this, Meyer uses deontic logic, which provides a logic framework for such concepts as permission, obligation, others.

In carrying out its actions, an agent must often be able to explain the current situation, and assimilate new (sometimes conflicting) information. A variety of techniques are used for this, including abduction (Shanahan, Chapter 11), belief revision (Meyer, Chapter 11) and logical knowledge-base techniques (Baral and Gelfond, Chapter 12).

While there is still research to be carried out concerning the logical representation and implementation of individual agents, actions, and activities, the general goal of the development of agent-based systems is to utilize *multiple* agents in order to solve problems in a coordinated manner.

While there has been research in this area, we are still a long way from developing general logic-based mechanisms for representing cooperative and coordinated activity in agents. However, work on individual agent aspects, such as actions, goals and permissions, is being (gradually) extended to multi-agent frameworks. This will eventually allow us to provide a full logical characterization of, for example, robots carrying out coordinated actions, or software agents cooperating in order to achieve complex goals. For a thorough survey of the agent-based systems field in general, see (Wooldridge and Jennings, 1995; Jennings et al., 1998).

12 THEORY OF BELIEFS

Two chapters on beliefs appear in the book. Perlis, Chapter 16, gives an overview of what sorts of things could be represented via a formal logic of

beliefs, and discusses why it is natural to do so. Thomason, Chapter 17 develops a formalism to handle a particular problem in beliefs.

Perlis notes that, from a programmer's or a logician's point of view, a belief can be thought of simply as a piece of data that has a truth value (it is *true* or *false*), and that can in some form be available to an agent to conduct its reasoning, much like an axiom or theorem. He further notes that in more cognitive terms, an agent's belief base can be thought of as its view of what the world is like. These two encompass quite a range of differences, and each of them admits of further distinctions as well.

Perlis uses the term *belief*, not in reference to a suspicion or impression that an item believed is possibly true, as in "I believe so"; but in the much stronger sense: what the believing agent takes to be true, as if it were possible to look inside the agent's "head" and see what the agent's view of the world is. He argues that with artificial agents, it is so possible; but notes that this leaves open the problem of which items found there are to count as beliefs (or as views of the world). He describes and contrasts a variety of notions of belief in AI; discusses why beliefs are essential to the AI enterprise; examines the extent to which beliefs are formal (or inference-bound), and considers certain tensions between beliefs and consistency.

Thomason develops a formalism combining elements of epistemic and non-monotonic logic, which is applied to the general topic of modeling the attitudes of other agents, and in particular to the problem of achieving mutuality. A proposition is *mutually believed* by a group in case every member of the group believes the proposition, and believes that every member believes the proposition, and believes that every member believes that every member believes the proposition, and so forth. Previous attempts to explain interagent reasoning about attitudes do not provide plausible formalizations of the reasoning that underlies mutuality in cases that seem to require it, or provide logical resources for formalizing cases where mutuality is blocked. Thomason's chapter is a first attempt in this direction.

As noted in the chapters by Perlis and Thomason, there are numerous issues that must be resolved before we can understand what is meant by a belief, how one might represent it, and reason with beliefs.

13 LOGIC AND LANGUAGE

Natural Language (NL) processing (NLP) is traditionally divided into syntax, semantics, and pragmatics. Logic-based approaches are generally not used for syntactic processing, except to the extent one might consider unification grammars and the use of logic programming languages to be logic-based. However, if one includes semantics and pragmatics, the area of *natural language understanding* and generation, which could be referred to as *natural language competence (NLC)*, is one in which there seems to be no viable alternative to a logic-based approach. Consider the process. A system gets NL input either from a person, a document, or some other system. The paradigmatic NL tasks, such as question answering, paraphrasing, or translation, require: storage either of the original language or of some representation of its meaning; computation

of information that can be inferred from the original; and generation of NL output. The representation, the procedures for drawing inferences, and the semantics needed to insure that the inference procedures are sound, constitute a logic, though not necessarily classical first-order logic. In fact, it is well-known that classical first-order logic is not as appropriate for NLC.

One chapter, by Schubert, Chapter 18, on natural language and logic, appears in the book. Schubert argues for associating situations (including events, episodes, eventualities, etc.) with arbitrarily complex sentences, rather than just atomic predicates, in the process of understanding natural language (NL) utterances. He argues that two such notions are needed: the notion of a sentence characterizing a situation; and the notion of a sentence partially describing a situation. He formalizes these notions in FOL**, an extension of first-order logic, and a part of his Episodic Logic, EL.

14 COMPUTATIONAL LOGIC

Computational logic deals with the use of logic as a mechanism to compute, and the formalization of methods to determine the computational complexity of such systems. *Prolog* is a representative tool for computational logic. In the same sense, constraint logic programming, systems developed for *NMR*, planning, and automated theorem proving fall in this category.

Several interesting developments have taken place in computational logic. One development is the emergence of fast software systems implementing non-monotonic reasoning. See the systems described in Section 15. While at various stages of the development process and at varying levels of performance, they demonstrate that nonmonotonic logics can be used as a computational tool. Another development is the recognition of very close connections between nonmonotonic logic and constraint satisfaction problems (Cholewiński et al., 1999). Specifically, non-monotonic logics provide natural and concise descriptions of constraint satisfaction problems and systems such as those listed above can serve as general-purpose constraint satisfaction solvers (Cholewiński et al., 1999).

With respect to complexity measures for computational logic, there have been a number of papers on the subject cited in Section 5.

In general, the greater the expressibility in *KR* and *CSR*, the greater the complexity. A great deal is known about the complexity of various semantics, and it is mostly that they are intractable. The challenge is to develop tractable classes of subsets of these semantics. In many instances, tractable subsets may be the ones of greatest interest. This was demonstrated by the use of Horn logic, rather than full first-order logic.

Representative of current research in computational complexity are the chapters by Gottlob et al., Chapter 19, and by Nebel, Chapter 20. Gottlob et al. discuss an application of DATALOG⁺ (a restricted form of logic programming, admitting negation but not function symbols) to hardware verification. The idea is to use a fragment of DATALOG⁺ that can be processed in linear time (data complexity). DATALOG⁺ is a convenient fragment of logic used in knowledge bases that allows for a natural expression of constructs such as transitive

closure, and thus of recursive queries. Further research, both by Database and LBAI communities showed relevance of DATALOG⁷ to traditional AI concerns such as the frame problem, inheritance reasoning and various forms of default reasoning. It turns out that some temporal logic languages, studied in verification can be faithfully represented within DATALOG⁷. This opens interesting applications of LBAI in areas such as μ -calculus, hardware verification, and model checking.

Nebel formalizes the notion of expressive power of propositional planning formalisms and analyzes some variants of STRIPS allowing for conditional effects and arbitrary Boolean formulae in preconditions. In general, planning with any of a wide variety of STRIPS formalisms is *PSPACE-complete*. However, one can still compare the expressive power of different formalisms through the notion of “compilability”: one formalism is as expressive as another if any planning problem and solution in the latter can be translated to the former with only a linear increase in size. For example, STRIPS with conditional effects is as expressive as STRIPS with CNF preconditions and effects.

He shows that STRIPS with conditional effects is incomparable to STRIPS with Boolean formulae as preconditions. He analyzes the expressive power of disjunctive preconditions and conditional effects and provides a complete classification of the relative expressiveness of STRIPS-like languages with restricted formulae and conditional effects under certain conditions.

These kinds of results apply to any scheme for implementing the propositional planning formalism, whether the STRIPS operators and problem specification are ultimately converted to a formula in CNF logic or to a plan graph. The results are useful in identifying a small “core” STRIPS language whose implementation can be highly optimized.

15 SYSTEM IMPLEMENTATIONS

As noted in this chapter, one of the major developments recognized by the attendees of the Workshop, was that reasoning systems for LBAI are able to scale up to handle large sets of data and rules. Such systems, and others will be important for dealing with realistic, large scale problems needed for LBAI. This section summarizes, in Tables 1.1, 1.2, 1.3, and 1.4, the capabilities of systems demonstrated at the Workshop.

Twelve systems were demonstrated at the Workshop. All these systems are listed at the *Homepage for Logic-Based AI Systems*: <http://www.uni-koblenz.de/ag-ki/LP> and it is planned that they will be updated on a regular basis. All of these systems are based on logic, and could not have been developed without the developments that have taken place in logic programming and nonmonotonic reasoning. They can be grouped into the following categories:

Five systems represented implementation efforts in *planning*. *BlackBox*, (Kautz and Selman, 1999b), and *TLPlan*, (Bacchus and Kabanza, 2000), deal with classical planning problems, *GPT* handles, in addition, probabilistic actions and states, *CCALC* is based on a special theoretical framework (*causal theories*), and *Golog* is a framework for controlling robots. Only *TLPlan* is used by researchers from other areas. The capabilities of these planners is impres-

sive. The development of efficient planners that can handle the frame problem and are designed to run fast was an important development at the Workshop.

Five systems represented implementations of *logic programming*, where default negation may appear in the body of a clause. *XSB* is a full-fledged *Prolog* system (with debugger, tracer etc), handles first-order programs under the *well-founded semantics (WFS)* (using tabling) and is by far the largest system. *LDL++* is one of the few existing deductive database systems and has a wide range of capabilities as described in Chapter 22. The remaining three systems are mainly propositional systems (or based on safe *Datalog* extensions): *Smodels* and *DeReS* compute *stable models* (or *answer sets*) and *DLV* computes *stable models* (or *WFS*) even for disjunctive programs. Of those systems, *XSB* and *Smodels* are able to handle large databases and rules. A new methodology of applying stable models emerges: they seem to be especially suited for constraint satisfaction problems.

One inductive logic programming system was represented. *CProgol* is one of the first and most important systems for inductive reasoning. Given positive and negative examples, it generates inductive explanations.

One system represented work on *multi-agents*. *IMPACT* is able to work within a multi-agent framework. Although logic is not used for implementing the system, it is essential in the agent decision cycle underlying each agent: the behavior of an agent is described by a so-called agent program. The semantics of these agent programs is similar to the *stable model* semantics.

Two chapters in the book relate to extensions of logic programming. Niemela and Simons, Chapter 21, describe new features in *Smodels*. The system has been extended with new constructs including cardinality and weight constraints. They summarize the extensions that have been included in the system, demonstrate their use, provide basic application methodology, illustrate the current level of performance of the system, and compare it to state-of-the-art satisfiability checkers.

Wang and Zaniolo, Chapter 22, describe new features incorporated in *LDL++* that support (i) monotonic user-defined aggregates, (ii) XY-stratified programs, and (iii) the nondeterministic choice constructs under stable model semantics. This integrated set of primitives supports a terse formulation and efficient implementation for complex computations, such as greedy algorithms and data mining functions, yielding levels of expressive power not contained in other deductive database systems.

Two chapters in the book relate to the systems discussed above. In Chapter 7, *CCALC* was used to implement the work on the “oldest planning problem in AI”. In Chapter 8, results are discussed for planning problems using a system, *SATPLAN*, related to *BlackBox*.

16 LOGIC APPLICATIONS TO MECHANICAL CHECKING AND DATA INTEGRATION

There have been many applications of logic to practical problems. These include the solution of a mathematical problem (McCune, 1997), the model-

Name	Purpose	Personnel/Web-Info	Language/ Man Years
TLPlan	AI planning and Search	Bacchus/Kabanza/Ady ???	W-98,NT/C 1
GPT	AI Planning (prob. actions)	Bonet/Geffner ???	Unix/C++ 2
Blackbox	State space planning	Kautz/Selman/Huang www.research.att.com/~kautz/blackbox/	Unix,NT/C++ 1
CCALC	Planning in action domains	Lifschitz/McCain/Turner www.cs.utexas.edu/users/mccain/cc	Sparc/Prolog 0.3
Golog	Control Programs for robots	Levesque et. al www.cs.toronto.edu/cogrobo	Prolog/120lines ??
XS B	Prolog System wrt WFS	Warren et. al. www.cs.sunysb.edu/~sbprolog/	Unix,W-95,NT/C 30
DLV	Disjunctive Programs for KR	Leone/Eiter/Gottlob et. al. www.dbae.tuwien.ac.at/proj/dlv	Unix/C++ 8
Smodels	Stable models for LP	Niemelae/Simons/Syrjanen saturn.hut.fi/pub/smodels/	Unix,Linux/C++ 2-3
DeReS	Answer Set Programming via DL	Marek/Truszczynski et.al. www.cs.engr.uky.edu/~mirek/research.html	Unix/C 5
LDL++	Deductive DB System	magna.cs.ucla.edu/ldl/	Unix/C++ 4
CProgol	Inductive Logic Programming	Muggleton/Srinivasan www.cs.york.ac.uk/~stephen/progol.html	C 5
IMPACT	Multiagent applications	Subrahmanian et. al. www.cs.umd.edu/projects/impact/	Java 4

Table 1.1 Systems I: General Information

Name	Related Systems	User Background	Used By Others
TLPlan		4th year undergraduates	Yes
GPT		Markov processes, functional STRIPS	No.
Blackbox	all planning systems	No logic required	No.
CCALC		Causal Theories, C, KR	No.
Golog		Logic	Few.
XSB	Deductive DB's	Prolog	Yes.
DLV	other systems for disjunctive programs	Logic for KR	Few.
Smodels	DLV, DeReS	No deep knowledge needed	Yes
DeReS	Graph problems	DL, stable models	Few.
LDL++	deductive DB's	No deep knowledge in logic.	Few.
CProgol	all ILP systems	Not much logic needed.	Many.
IMPACT	Systems are too different to compare	Logic helps	Few.

Table 1.2 Systems II: Related Systems and Users

Name	Methodology	Specifics
TLPlan	Standard encoding methodology (STRIPS). Input: initial state, possible transitions, goal state	arbitrary search problems
GPT	Representing actions, sensors and goals. Using state models and Markov processes. Heuristic search and dynamic programming	probabilistic reasoning
Blackbox	STRIPS approach, temporal logic	Uses TP's
CCALC	Model checking for causal theories. Methodology described in various papers.	Language <i>C</i>
Golog	Axiomatizing the domain in situation calculus, causal laws	Situation calculus
XSB	Tabling as a primitive construct. Book on its way to explain underlying intuitions and methodology.	Tabling, memoization
DLV	Encoding methodology on its way (simple and uniform). Several frontends for specific AI tasks.	
Smodels	Application Methodology on its way: Constraint Programming Paradigm	stable models
DeReS	Answer Set Programming, Constraint Programming	constraint satisfaction
LDL++	Informal and formal methodologies are available. Data intensive applications.	Deductive Database
CProgol	Explained in User Manual. user-friendly interface on its way	Inductive logic
IMPACT	Explained in forthcoming book. User friendly interfaces.	Heterogenous DB's

Table 1.3 Systems III: Methodology

Name	Benchmarks	Problem Size
TLPlan	Various planning benchmarks	Solves transportation problems (50 objects, 100 trucks, 400 locations). Generates plans of 300 steps in 350 sec.
GPT	No good benchmarks yet.	
Blackbox	AIPS planning competition: Good accepted benchmarks.	Can handle some realistic problems, but is mainly prototype. Handles 10^{17} states.
CCALC	Planning is reduced to Sat-checking. Quantitative comparison difficult.	Handles some problems of realistic size.
Golog	No benchmarks.	-
XSB	Graph reachability, same generation	Data intensive problems with 500,000 tuples. Large model checking problems are solved (competes with specialized commercially available model checkers).
DLV	Graph problems	Handles some problems of realistic size.
Smodels	Large collection of benchmarks (see homepage). Also test cases from graph problems.	Handles efficiently programs with hundreds of thousands of non-trivial ground rules.
DeReS	Graph problems.	Handable size depends on stratification. Sometimes thousands of DL-rules can be handled, sometimes not even 100 can.
LDL++	No accepted benchmarks	Handled several problems of realistic size. Goal to transfer this technology to SQL DBMs.
CProgol	Accepted benchmarks at www-ai.ijs.si/ilpnet/ .	Handles around 10,000 examples in reasonable time. Depends on hypothesis space. Used successfully for many realistic applications.
IMPACT	No benchmarks.	Hard to determine. Computes joins over heterogeneous data types on the order of 45 billion tuples. Still prototype, but on its way towards handling massive problems.

Table 1.4 Systems IV: Benchmarks

based reactive system developed for NASA's Deep Space 1 is incorporated into the space mission as discussed at the Workshop, the use of deductive databases for stock broker applications (Minker, 1999a), and numerous others. Two chapters on applications are provided in this book. Moore, Chapter 23, discusses the use of a mechanical checker for hardware verification of a processor. Levy, Chapter 24, applies deductive database techniques to the problem of data integration of systems.

Moore focuses on the capabilities of the ACL2 system (the successor of the Boyer-Moore NQTHM prover Boyer and Moore, 1979; Boyer and Moore, 1997). The ACL2 effort of Moore and Kaufmann (Kaufmann and Moore, 1997; Kaufmann and Moore, 1999) defines the state-of-the-art in microprocessor verification. ACL2 has been used to formally model a Motorola CAP digital signal processor, the Rockwell-Collins JEM1 microprocessor (the world's first silicon Java Virtual Machine) and much of the hardware in the AMD K7 (Athalon) floating-point unit. The human effort to verify any meaningful property is considerable even though any single computation is fully automatic. This is because most "real" problems require partitioning the problem and, in particular, finding appropriate lemmas or induction hypotheses. Thus, for the JEM1 microprocessor, only a selected set of properties have been verified to date. The AMD Corporation is using the ACL2 system for in-house development. The use of ACL2 at AMD has uncovered at least four hardware design bugs – bugs that had survived hundreds of millions of test cases – in time to fix them before the processor was shipped. As noted by Moore, "The main message is: mechanically checked proofs about industrial designs are possible". He also notes that it is also necessary to express ones models and their alleged properties as formulas, and that proofs are not automatic, the user must provide some form of guidance. A full discussion of future directions of automated deduction, written by Donald Loveland, can be found on the WWW at <http://www.cs.duke.edu/AutoDedFD>. The core of the report is given in the AI Magazine, Spring 1999 Loveland, 1999.

Levy addresses the nature of data integration systems and the opportunity for the exploratory use of logic-based representations in these systems. Data integration systems differ from previous database applications in that the data sources are pre-existing, independent systems. A data integration system requires a flexible mechanism for describing contents of sources and so provides an opportunity for the use of logic-based techniques. Levy surveys the main logic-based languages that have been considered for describing data sources in data integration systems, and the specialized inference techniques that were developed in this context. The inference mechanisms utilize query rewrite rules to convert a user query at the top-level (mediated) system to (one or more) local query (queries). The chapter demonstrates that (1) carefully designed logical languages have proven successful in data integration systems, and (2) there is a need for further research in extending *KR* languages to deal with real-world concerns, such as grouping and aggregation, multiset semantics and the representation of nested structures.

17 SUMMARY AND CONCLUSIONS

The general consensus of the attendees at the Workshop was that the field of LBAI has made significant contributions both to the theory and practice of AI. In the theory of LBAI, contributions have been made to the foundations of commonsense reasoning, the formalization of nonmonotonic reasoning, the solution of the frame problem, the development of semantics for nonmonotonic reasoning, and results on computational complexity. In the practice of LBAI, we have seen the area of automated reasoning successfully applied to the verification of hardware systems, to the solution of an outstanding problem in mathematics, and to improvements in solving satisfiability problems that have been incorporated into planning systems. The emergence of fast software systems implementing nonmonotonic reasoning is the most significant development in that area in the recent years, highlighted by the systems *Smodels*, *XSB*, *DLV*, and *DeReS*. System demonstrations of practical running systems that can be used for realistic knowledge-base and nonmonotonic reasoning applications, the development of planning systems and tools for robotics are important contributions, that were not available five years ago. There remain many long-term and short-term research efforts that must be handled before AI will be able to develop truly intelligent machines. The efforts in LBAI are expected to contribute to such developments.

Acknowledgements

Material in the various sections of this chapter is drawn from informal reports sent to me by participants and session chairs at the Workshop. I list the sections and those who contributed to these sections. I greatly appreciate the time and energy they spent writing their reports. Chitta Baral, Juergen Dix, John Horty, and Witek Marek, provided me with comments on the entire chapter. I made many changes to the chapter based upon their comments. I am responsible for editing, revising and integrating the material into the chapter. The contributors to the individual sections are: Section 2, Background - Jacques Cohen, Chitta Baral, and Melvin Fitting; Section 3, Commonsense Reasoning - Leora Morgenstern; Section 4, Knowledge Representation - David Etherington; Section 5, Nonmonotonic Reasoning - Vladimir Lifschitz; Section 6, Abductive and Inductive Reasoning - Francesca Toni; Section 7, Logic, Probability and Decision Theory - Faheim Bacchus; Section 8, Logic for Causation and Actions - Javier Pinto; Section 9, Logic and Planning - Henry Kautz; Section 10, Cognitive Robotics - Chitta Baral and Fangzhen Lin; Section 11, Logic for Agents and Actions - Michael Fisher; Section 12, Theories of Belief - Melvin Fitting; Section 13, Logic and Language - Stuart Shapiro and Lenhart Schubert; Section 14, Computational Logic - Victor Marek and Mirek Truszczynski; Section 15, System Implementations - Juergen Dix; Section 16, Applications of Theorem Proving - Donald Loveland;

References

- Allen, J. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154.
- Apt, K., Blair, H., and Walker, A. (1988). Towards a theory of declarative knowledge. In Minker, J., editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann Pub., Los Altos, CA.
- Apt, K. R. and van Emden (1982) Contributions to the theory of logic programming. *J. ACM*, 29(3):841–862.
- Bacchus, F., Grove, A., Halpern, J., and Koller, D. (1993). Statistical foundations for default reasoning. *Proc. IJCAI-93*, pages 563–569.
- Bacchus, F. (1990) *Representing and Reasoning with Probabilistic Knowledge*. MIT Press.
- Bacchus, F. and Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 16:123–191.
- Baral, C. and Gelfond, M. (1994). Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148.
- Boutilier, C., Reiter, R., Soutchanski, M., and Thrun, S. (2000). Decision-theoretic, high level agent programming in the situation calculus. In *Proc. Amer. Assoc. for Artificial Intelligence - 2000*.
- Boyer, R. S. and Moore, J. S. (1979). *A Computational Logic*. Academic Press.
- Boyer, R. S. and Moore, J. S. (1997). *A Computational Logic Handbook, Second Edition*. Academic Press.
- Brass, S., Dix, J., and Przymusinski, T.C. (1996). Super logic programs. *Knowledge Representation*, pages 529–540.
- Brooks, R. A. (1991). Intelligence without reason. pages 569–595. Morgan Kaufmann.
- Cadoli, M. and Lenzerini, M. (1991). The complexity of closed world reasoning and circumscription. *Knowledge Representation*, pages 550–555.
- Cadoli, M. and Schaerf, M. (1992). A survey on complexity results for non-monotonic logics. Technical report, University di Roma “La Sapienza”, Dipartimento di Informatica e sistematica, Roma, Italy.
- Chandra, A. and Harel, D. (1985). Horn clause queries and generalizations. *Journal of Logic Programming*, 2(1):1–15.
- Chang, C. L. and Lee, R. C. T. (1973). *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York.
- Cholewiński, P., Marek, V. W. and Mikitiuk, A., and Truszczyński, M. (1999). Computing with default logic. *Artificial Intelligence*, 112.
- Cholewiński, P., Marek, W., and Truszczyński, M. (1996). Default reasoning system deres. In *Proceedings of KR-96*, pages 518–528, San Francisco, California. Morgan Kaufmann.
- Clark, K. L. (1978). Negation as Failure. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York.
- Colmerauer, A. (1985). Prolog in 10 figures. *Communications of the ACM*, 28(12):1296–1310.

- Colmerauer, A., Kanoui, H., Pasero, R., and Roussel, P. (1973). Un systeme de communication homme-machine en francais. TR, Groupe de Intelligence Artificielle Univ. de Aix-Marseille II, Marseille.
- Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (1997). Complexity and expressive power of logic programming. In *Proc. of 12th annual IEEE Conference on Computational Complexity*, pages 82–101.
- Davis, E. (1998). The naive physics perplex. *AI Magazine*, 19(14):51–79.
- Davis, E. (1999). Guide to axiomatizing domains in first-order logic. *Electronic Newsletter on Reasoning About Actions and Change*.
- Davis, M. and Putnam, H. (1960). A computing procedure for quantification theory. *J. ACM*, 7:201–215.
- Dimopoulos, Y. and Kakas, A. (1995). Abduction and inductive learning. In De Raedt, L., editor, *Proc. 5th Inductive Logic Programming Workshop (ILP95)*, pages 25–28, Leuven, Belgium. KU Leuven.
- Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. (1997). A deductive system for nonmonotonic reasoning. In Dix, J., Furbach, U., and Nerode, A., editors, *Proc. 4th Int'l. Conf. on Logic Programming and Nonmonotonic Reasoning*, number 1265 in Lecture Notes in AI. Springer.
- Fikes, R. and Nilsson, N. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 5(2):189–208.
- Fikes, R. and Nilsson, N. (1993). STRIPS, a retrospective. *Journal of Artificial Intelligence*, 59(1/2):227–232.
- Fitting, M. C. (2000). Fixpoint semantics for logic programming – a survey. *Theoretical Computer Science*. To appear.
- Gallaire, H. and Minker, J., editors (1978). *Logic and Databases*. Plenum Press, New York.
- Gallaire, H., Minker, J., and Nicolas, J.-M. (1984). Logic and databases: A deductive approach. *ACM Computing Surveys*, 16(2):153–185.
- Geffner, H. (1990). Causal theories for nonmonotonic reasoning. In *Proc. AAAI-90*, pages 524–530. AAAI Press.
- Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski, R. and Bowen, K., editors, *Logic Programming: Proc. 5th Int'l Conf. and Symp.*, pages 1070–1080.
- Gelfond, M. and Lifschitz, V. (1990). Logic programs with classical negation. In Warren, D. and Szeredi, P., editors, *Proc. 7th Int'l. Conf. on Logic Programming*, pages 579–597, Jerusalem, Israel. MIT Press.
- Gelfond, M. and Lifschitz, V. (1992). Representing actions in extended logic programming. In Apt, K., editor, *Proc. Joint Int'l Conf. and Symp. on Logic Programming*, pages 559–573.
- Gelfond, M. and Lifschitz, V. (1993). Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2,3,4):301–323.
- Gelfond, M. and Lifschitz, V. (1998). Action languages. *Electronic Transactions on AI*, 3. (<http://www.ep.liu.se/ea/cis/1998/016>)
- Gelfond, M., Lifschitz, V., and Rabinov, A. (1991). What are the limitations of the situation calculus? In Boyer, R., editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 167–179. Kluwer.

- Ginsberg, M. and Smith, D. (1988a). Reasoning about action I: a possible world approach. *Artificial Intelligence*, 35:165–195.
- Ginsberg, M. and Smith, D. (1988b). Reasoning about action II: the qualification problem. *Artificial Intelligence*, 35:311–342.
- Giunchiglia, E. and Lifschitz, V. (1998). An action language based on causal explanation: Prelim. Rpt. In *Proc. AAAI-98*, pages 623–630. AAAI Press.
- Green, C. (1969). Theorem proving by resolution as a basis for question - answering systems. In Michie, B. M. D., editor, *Machine Intelligence 4*, pages 183–205. Edinburgh University Press, New York.
- Green, C. and Raphael, B. (1968a). Research in intelligent question answering systems. *Proc. ACM 23rd National Conf.*, pages 169–181.
- Green, C. and Raphael, B. (1968b). The use of theorem-proving techniques in question-answering systems. *Proc. ACM 23rd National Conf.*
- Haas, A. (1987). The case for domain-specific frame axioms. In Brown, F. M., editor, *The Frame Problem in Artificial Intelligence, (Proc. 1987 Workshop)*.
- Hanks, S. and McDermott, D. (1987). Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412.
- Hayes, P. (1973a). Computation and deduction. In *Proceedings of the 2nd Symp. on Mathematical Foundations of Computer Science*, pages 107–113, Czechoslovakia: Czechoslovakian Academy of Sciences.
- Hayes, P. (1985a). Naive physics i: ontology for liquids. In Hobbs, J. and Moore, R., editors, *Formal Theories of the Commonsense World*, chapter 3, pages 71–107. Ablex, Norwood, New Jersey.
- Hayes, P. (1985b). The second naive physics manifesto. In Hobbs, J. and Moore, R., editors, *Formal Theories of the Commonsense World*, chapter 1, pages 1–36. Ablex, Norwood, New Jersey.
- Hayes, P. J. (1973b). The frame problem and related problems in artificial intelligence. *Artificial and Human Thinking*, pages 45–59.
- Jaffar, J. and Maher, M. (1994). Constraint logic programming:a survey. *Journal of Logic Programming*, 19-20:503–581.
- Jenkin, M., Lespérance, Y., Levesque, H., Lin, F., Lloyd, J., Marcu, D., Reiter, R., Scherl, R., and Tam, K. (1997). A logical approach to portable high-level robot programming. In *Proc. 10th Australian Joint Conf. on Artificial Intelligence (AI'97)*, Perth, Australia.
- Jennings, N. R., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1:7–38.
- Kakas, A. C., Kowalski, R. A., and Toni, F. (1993). Abductive logic programming. *Journal of Logic and Computation*, 6(2):719–770.
- Kaufmann, M. and Moore, J. S. (1997). An industrial strength theorem prover for a logic based on Common Lisp. *IEEE Transactions on Software Engineering*, 23(4):203–213.
- Kaufmann, M. and Moore, J. S. (1999). *The ACL2 user's manual*. <http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html>.
- Kautz, H. and Selman, B. (1999b). Unifying sat-based and graph-based planning. In *Proc. of IJCAI 99*, pages 318–325.

- Kowalski, R. (1974). Predicate logic as a programming language. *Proc. IFIP 4*, pages 569–574.
- Lespérance, Y., Levesque, H., Lin, F., Marcu, D., Reiter, R., and Scherl, R. (1994). A logical approach to high-level robot programming – a progress report. In *Control of the Physical World by Intelligent Systems, Working Notes of the 1994 AAAI Fall Symp.*
- Lespérance, Y., Levesque, H. J., Lin, F., Marcu, D., Reiter, R., and Scherl, R. B. (1994). A logical approach to high-level robot programming: A progress report. In B. Kuipers, editor, *Control of the Physical World by Intelligent Systems: Papers from 1994 AAAI Fall Symp.*
- Levesque, H., Reiter, R., Lespérance, Y., Lin, F., and Scherl, R. (1997). GOLOG: a logic programming language for dynamic domains. *J. of Logic Programming, Special Issue on Actions*, 31(1-3):59–83.
- Lifschitz, V. (1987). On the semantics of STRIPS. In Georgeff, M. and Lansky, A., editors, *Reasoning about Actions and Plans*, pages 1–9. Morgan Kaufmann, San Mateo, CA.
- Lin, F. (1995). Embracing causality in specifying the indirect effects of actions. In *Proc. IJCAI-95*, pages 1985–1991.
- Lobo, J., Minker, J., and Rajasekar, A. (1992). *Foundations of Disjunctive Logic Programming*. MIT Press.
- Loveland, D. (1978). *Automated Theorem Proving: A Logical Basis*. North-Holland Publishing Co.
- Loveland, D. (1999). Automated deduction: Looking ahead. *AI Magazine*, 20(1):77–98.
- Marek, V. and Truszczyński, M. (1993). *Nonmonotonic Logic: Context-Dependent Reasoning*. Springer-Verlag.
- McCain, N. and Turner, H. (1997). Causal theories of action and change. In *Proc. AAAI-97*, pages 460–465.
- McCarthy, J. (1959). Programs with common sense. In *Proc. Teddington Conf. on the Mechanisation of Thought Processes*, pages 75–91, London. Her Majesty's Stationery Office. Reprinted (with an added section on 'Situations, Actions and Causal Laws') in *Semantic Information Processing*, ed. M. Minsky (Cambridge, MA: MIT Press (1963)).
- McCarthy, J. (1963). Situations, actions and causal laws. Memo 2. AI Laboratory, Stanford University, Stanford, CA.
- McCarthy, J. (1977). Epistemological problems in artificial intelligence. In *Proc. 5th International Conference on Artificial Intelligence*, pages 1038–1044.
- McCarthy, J. (1978). History of lisp. In Wexblatt, R., editor, *History of Programming Languages: Proc. of the ACM SIGPLAN Conf.*, pages 3–57. Academic Press. Published in 1981 (Conf. date: 1978).
- McCarthy, J. (1980). Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1 and 2):27–39.
- McCarthy, J. and Hayes, P. (1969b). 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.
- McCune, W. (1997). Solution of the Robbins problem. *J. Automated Reasoning*, 19(3):263–276.

- Minker, J. (1988). Perspectives in deductive databases. *Journal of Logic Programming*, 5:33–60.
- Minker, J. (1993). An overview of nonmonotonic reasoning and logic programming. *Journal of Logic Programming*, 17(2, 3 and 4):95–126.
- Minker, J. (1994). Overview of disjunctive logic programming. *Journal of Artificial Intelligence & Mathematics*, 12(1-2):1–24.
- Minker, J. (1996). Logic and databases: a 20 year retrospective. In Pedreschi, D. and Zaniolo, C., editors, *Logic in Databases*, pages 3–57. Springer. Proc. Int. Workshop LID'96, San Miniato, Italy.
- Minker, J. (1999a). Logic and databases: a 20 year retrospective - updated in honor of Ray Reiter. In Levesque, H. J. and Pirri, F., editors, *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 234–299. Springer.
- Minker, J. (1999b). The workshop on logic-based artificial intelligence. *AI Magazine*, 20(4):21–47.
- Minsky, M. (1975). A framework for representing knowledge. In Winston, P., editor, *The Psych. of Computer Vision*, pages 211–277. McGraw-Hill, NY.
- Moore, R. C. (1985). Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94.
- Niemela, I. and Simons, P. (1997). Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In Dix, J., Furbach, U., and Nerode, A., editors, *Logic Programming and Nonmonotonic Reasoning - 4th Int. Conf.*, pages 420–429, Dagstuhl, Germany. Springer.
- Nilsson, N. (1982). *Principles of Artificial Intelligence*. Springer-Verlag.
- Nilsson, N. (1984). Shakey the robot. Technical Note 323, SRI International, Menlo Park, California.
- Pednault, E. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In Brachman, R., Levesque, H., and Reiter, R., editors, *Proc. First Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 324–332.
- Peirce, C. S. (1883). A theory of probable inference. note b. the logic of relatives. In *Studies in logic by members of the Johns Hopkins Univ.*, pages 187–203.
- Plotkin, G. (1969). A note on inductive generalisation. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, Edinburgh.
- Plotkin, G. (1971). *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University.
- Przymusinski, T. C. (1988). On the declarative semantics of deductive databases and logic programming. In Minker, J., editor, *Foundations of Deductive Databases and Logic Programming*, chapter 5, pages 193–216. Morgan Kaufmann Pub., Washington, D.C.
- Quillian, R. (1968). Semantic Memory. In Minsky, M., editor, *Semantic Information Processing*, pages 216–270. MIT Press, Cambridge, Massachusetts.
- Rao, P., Sagonas, K., Swift, T., Warren, D., and Fierc, J. (1997). XSB: A system for efficiently computing well-founded semantics. In Dix, J., Ferbach, U., and Nerode, A., editors, *Logic and Nonmonotonic Reasoning - 4th Int'l Conf., LPNMR'97*, pages 430–440.

- Reiter, R. (1980). A Logic for Default Reasoning. *Artificial Intelligence*, 13(1 and 2):81–132.
- 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, *AI and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press.
- Reiter, R. (1993). Proving properties of states in the situation calculus. *Artificial Intelligence*, 64:337–351.
- Robinson, J. A. (1965). A machine oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41.
- Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.
- Sandewall, E. (1995). *Features and Fluents*, volume 1. Oxford University Press.
- Schank, R. and Abelson, R. (1977). *Scripts, Plans, Goals, and Understanding*. Lawrence Erlebaum.
- Schubert, L. (1990). Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In Kyburg, H., Loui, R., and Carlson, G., editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer.
- Shanahan, M. P. (1997). *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press.
- van Emde, M. and Kowalski, R. (1976). The Semantics of Predicate Logic as a Programming Language. *JACM*, 23(4):733–742.
- Van Gelder, A. (1988). Negation as failure using tight derivations for general logic programs. In Minker, J., editor, *Found. of Deductive Databases and Logic Programming*, pages 149–176. Morgan Kaufmann.
- Van Gelder, A., Ross, K., and Schlipf, J. (1988). Unfounded sets and well-founded semantics for general logic programs. In *Proc. 7th ACM Symp. on Principles of Database Systems.*, pages 221–230.
- Warren, D. S. (1999). The XSB programming system. Technical report, State University of New York at Stonybrook. <http://www.cs.sunysb.edu/sbprolog/xsb-page.html>.
- Wooldridge, M. and Jennings, N. (1995). Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152.
- Zanjolo, C., Arni, N., and Ong, K-L. (1993). Negation and aggregates in recursive rules. *Proceedings of DOOD93*, pages 204–221.

II

COMMONSENSE REASONING

Chapter 2

CONCEPTS OF LOGICAL AI

John McCarthy

*Computer Science Department
Stanford University
mccarthy@stanford.edu*

Abstract Logical AI involves representing knowledge of an agent's world, its goals and the current situation by sentences in logic. The agent decides what to do by inferring that a certain action or course of action is appropriate to achieve the goals. We characterize briefly a large number of concepts that have arisen in research in logical AI.

Reaching human-level AI requires programs that deal with the *common sense informatic situation*. Human-level logical AI requires extensions to the way logic is used in formalizing branches of mathematics and physical science. It also seems to require extensions to the logics themselves, both in the formalism for expressing knowledge and the reasoning used to reach conclusions.

A large number of concepts need to be studied to achieve logical AI of human level. This chapter presents candidates, some of them well known, but others new.

Keywords: Logical AI, philosophical foundations, common sense reasoning, bounded informative situation, common sense informative situation, human-level AI, epistemology, knowledge representation, elaboration tolerance, non-monotonic reasoning.

1 INTRODUCTION AND EXHORTATION

Logical AI involves representing knowledge of an agent's world, its goals and the current situation by sentences in logic. The agent decides what to do by inferring that a certain action or course of action is appropriate to achieve the goals.

Logical AI is more intellectually ambitious than approaches to AI based on evolution. It requires understanding enough about the common sense world and about the requirements for deciding on a course of action to put the basic facts in a logical computer program. Once we have a program smart enough to read, it can get any amount of book learning. In principle, evolutionary AI

could reach human level without anyone understanding how it worked. To some people this seems easier than the logical approach, but 50 years of experience has not put either approach far enough ahead to discourage advocates of the other.

It is useful to compare mathematical logic itself and its use in formalizing mathematics with the increased requirements involved in formalizing thinking and deciding in the common sense world.

Mathematical logic undertakes to formalize all reasoning that is guaranteed to be correct. By early in the previous century, this goal had been substantially achieved, and the systems of mathematical logic were adequate for formalizing mathematical facts and reasoning. However, the interactive theorem provers available today are still not convenient for doing mathematical reasoning, because too many details have to be supplied.

Logical AI has to do more if it is to reach human level intelligence than is required for formalizing mathematics. First it has to find ways to express the facts that humans have or can have about the common sense world, including facts that are not ordinarily or easily expressed in natural language. Second logical AI must often use nonmonotonic reasoning, just as humans must use nonmonotonic reasoning in deciding what to do.

This chapter is one result of exploring the concepts that are required. Some of the concepts have been extensively explored already, and some are new. Some have been explored in philosophy but not with detail required for AI. Some concepts are for use within logical AI and others are about logical AI.

Logical AI has both *epistemological* problems and *heuristic* problems. The former concern the knowledge needed by an intelligent agent and how it is represented. The latter concerns how the knowledge is to be used to decide questions, to solve problems and to achieve goals. These are discussed in (McCarthy and Hayes, 1969). Neither the *epistemological problems* nor the *heuristic* problems of logical AI have been solved. The epistemological problems are more fundamental, because the form of their solution determines what the heuristic problems will eventually be like.¹

Concepts of logical AI can be sorted rather loosely into categories. These categories include epistemology, theory of action, heuristics, learning, and creativity.

2 EPISTEMOLOGY

In philosophy, epistemology is the study of knowledge, its form and limitations. This will do pretty well for AI also, provided we include in the study common sense knowledge of the world and scientific knowledge. Both of these offer difficulties philosophers haven't studied, e.g. they haven't studied in detail what people or machines can know about the shape of an object in the field of view, remembered from previously being in the field of view, remembered

¹Thus the heuristics of a chess program that represents "My opponent has an open file for his rooks." by a sentence will be different from those of a present program which at most represents the phenomenon by the value of a numerical co-efficient in an evaluation function.

from a description or remembered from having been felt with the hands. This is discussed a little in (McCarthy and Hayes, 1969) and in (McCarthy, 1989).

Most AI work has concerned heuristics, i.e. the algorithms that solve problems, usually taking for granted a particular epistemology of a particular domain, e.g. the representation of chess positions. Progress is limited by the epistemology (often called ontology nowadays) chosen.

2.1 LOGICAL AI

Logical AI in the sense of the present chapter was proposed in (McCarthy, 1959) and updated in (McCarthy, 1989). The idea is that an agent can represent knowledge of its world, its goals and the current situation by sentences in logic and decide what to do by inferring that a certain action or course of action is appropriate to achieve its goals.

Logic is also used in more restricted ways in AI, databases, logic programming, hardware design and other parts of computer science and technology. Many AI systems represent facts by a limited subset of logic and use non-logical programs as well as logical inference to make inferences. Databases often use only ground formulas. Logic programming often restricts its representation to Horn clauses. Hardware design usually involves only propositional logic. These restrictions are almost always motivated by considerations of computational efficiency.

2.2 KNOWLEDGE LEVEL

Allen Newell (Newell, 1982 and Newell, 1993) did not advocate (as we do here) using logic as the way a system should represent its knowledge internally. He did say that a system can often be appropriately described as knowing certain facts even when the facts are not represented by sentences in memory. This view corresponds to Daniel Dennett's *intentional stance* (Dennett, 1971), reprinted in (Dennett, 1978), and was also proposed and elaborated in (McCarthy, 1979).

2.3 BOUNDED INFORMATIC SITUATION

Formal theories in the physical sciences deal with a *bounded informative situation*. Scientists decide informally in advance what phenomena to take into account. For example, much celestial mechanics is done within the Newtonian gravitational theory and does not take into account possible additional effects such as outgassing from a comet or electromagnetic forces exerted by the solar wind. If more phenomena are to be considered, scientists must make new theories—and of course they do.

Most AI formalisms also work only in a bounded informative situation. What phenomena to take into account is decided by a person before the formal theory is constructed. With such restrictions, much of the reasoning can be monotonic, but such systems cannot reach human level ability. For that, the machine will have to decide for itself what information is relevant, and that reasoning will inevitably be partly nonmonotonic.

One example is the “blocks world” where the position of a block x is entirely characterized by a sentence $At(x, l)$ or $On(x, y)$, where l is a location or y is another block.

Another example is the Mycin (Davis et al., 1977) expert system in which the ontology (objects considered) includes diseases, symptoms, and drugs, but not patients (there is only one), doctors or events occurring in time. Mycin therefore cannot project the result of a proposed treatment, because it can't understand about events. See (McCarthy, 1983) for more comment about Mycin.

Expert Systems. Expert systems are an example of systems where the informative situations are bounded.

These are designed by people, i.e. not by computer programs, to take a limited set of phenomena into account. Many of them do their reasoning using logic, and others use formalisms amounting to subsets of first order logic. Many require very little common sense knowledge and reasoning ability. Restricting expressiveness of the representation of facts is often done to increase computational efficiency.

Epistemologically Adequate Languages. A logical language for use in the common sense informative situation must be capable of expressing directly the information actually available to agents. For example, giving the density and temperature of air and its velocity field and the Navier-Stokes equations does not practically allow expressing what a person or robot actually can know about the wind that is blowing. We and robots can talk about its direction, strength and gustiness approximately, and can give a few of these quantities numerical values with the aid of instruments if instruments are available, but we have to deal with the phenomena even when no numbers can be obtained. (McCarthy and Hayes, 1969) introduced the idea of epistemological adequacy.

2.4 THE COMMON SENSE INFORMATIC SITUATION

In general a thinking human is in what we call the *common sense informative situation*, as distinct from the *bounded informative situation*. The known facts are necessarily incomplete. We live in a world of middle-sized objects which can only be partly observed. We only partly know how the objects that can be observed are built from elementary particles in general, and our information is even more incomplete about the structure of particular objects. These limitations apply to any buildable machines, so the problem is not just one of human limitations.²

In many actual situations, there is no *a priori* limitation on what facts are relevant. It may not even be clear in advance what phenomena should be taken

²Science fiction and scientific and philosophical speculation have often indulged in the *Laplacian fantasy* of super beings able to predict the future by knowing the positions and velocities of all the particles. That isn't the direction to go. Rather the super beings would be better at using the same information that is available to human senses.

into account. The consequences of actions cannot be fully determined. The *common sense informatic situation* necessitates the use of *approximate concepts* that cannot be fully defined and the use of *approximate theories* involving them. It also requires *nonmonotonic* reasoning in reaching conclusions. Many AI texts and articles effectively assume that the information situation is bounded—without even mentioning the assumption explicitly.

Consider the infamous physics exam problem of using a barometer to determine the height of a building. The student is expected to propose measuring the air pressure at the top and bottom, but physics teachers have puzzled about the acceptability of the following other answers. (1) Drop the barometer from the top of the building and measure the time before it hits. (2) Lower the barometer on a string till it reaches the ground and measure the string. (3) Compare the shadow of the barometer with the shadow of the building. (4) Say to the janitor “I’ll give you this fine barometer if you will tell me the height of the building.” In this common sense informatic situation, there may be still other solutions to the problem taking into account additional phenomena.

The common sense informatic situation often includes some knowledge about the system’s mental state as discussed in (McCarthy, 1996a).

A key problem in formalizing the common sense informatic situation is to make the axiom sets elaboration tolerant. See Section 2.7.

2.5 COMMON SENSE KNOWLEDGE ABOUT THE WORLD

As first discussed in (McCarthy, 1959), humans have a lot of knowledge of the world which cannot be put in the form of precise theories. Though the information is imprecise, we believe it can still be put in logical form. The Cyc project (Lenat and Guha, 1990) aims at making a large base of common sense knowledge. Cyc is useful, but further progress in logical AI is needed for Cyc to reach its full potential.

Common Sense Physics. Corresponds to people’s ability to make decisions involving physical phenomena in daily life, e.g. deciding that the spill of a cup of hot coffee is likely to burn Mr. A, but Mr. B is far enough away to be safe. It differs from qualitative physics, as studied by most researchers in *qualitative reasoning*, in that the system doing the reasoning must itself use common sense knowledge to decide what phenomena are relevant in the particular case. See (Hayes, 1985) for one view of this.

Representation of Physical Objects. We aren’t close to having an epistemologically adequate language for this. What do I know about my pocket knife that permits me to recognize it in my pocket or by sight or to open its blades by feel or by feel and sight? What can I tell others about that knife that will let them recognize it by feel, and what information must a robot have in order to pick my pocket of it?

Representation of Space and Shape. We again have the problem of an epistemologically adequate representation. Trying to match what a human can remember and reason about when out of sight of the scene is more what we need than any pixel by pixel representation. Some problems of this are discussed in (McCarthy, 1995a) which concerns the Lemmings computer games. One can think about a particular game and decide how to solve it away from the display of the position, and this obviously requires a compact representation of partial information about a scene.

2.6 RICH AND POOR ENTITIES

A *rich entity* is one about which a person or machine can never learn all the facts. The state of the reader's body is a rich entity. The actual history of my going home this evening is a rich entity, e.g. it includes the exact position of my body on foot and in the car at each moment. While a system can never fully describe a rich entity, it can learn facts about it and represent them by logical sentences.

Poor entities occur in plans and formal theories and in accounts of situations and events and can be fully prescribed. For example, my plan for going home this evening is a poor entity, since it does not contain more than a small, fixed amount of detail. Rich entities are often approximated by poor entities. Indeed some rich entities may be regarded as inverse limits of trees of poor entities. (The mathematical notion of inverse limit may or may not turn out to be useful, although I wouldn't advise anyone to study the subject quite yet just for its possible AI applications.)

2.7 ELABORATION TOLERANCE

A set of facts described as a logical theory needs to be modifiable by adding sentences rather than only by going back to natural language and starting over. For example, we can modify the missionaries and cannibals problem by saying that there is an oar on each bank of the river and that the boat can be propelled with one oar carrying one person but needs two oars to carry two people. Some formalizations require complete rewriting to accommodate this elaboration. Others share with natural language the ability to allow the elaboration by an addition to what was previously said.

There are degrees of elaboration tolerance. A static space formalization of the missionaries and cannibals problem in which a state is represented by a triplet $(m\ c\ b)$ of the numbers of missionaries, cannibals and boats on the initial bank is less elaboration tolerant than a situation calculus formalism in which the set of objects present in a situation is not specified in advance. In particular, the former representation needs surgery to add the oars, whereas the latter can handle it by adjoining more sentences—as can a person. The realization of elaboration tolerance requires nonmonotonic reasoning. See (McCarthy, 1997), (Lifschitz, 2000), and (Shanahan, 1997).

2.8 CONJUNCTIVITY

It often happens that two phenomena are independent. In that case, we may form a description of their combination by taking the conjunction of the descriptions of the separate phenomena. The description language satisfies *conjunctivity* if the conclusions we can draw about one of the phenomena from the combined description are the same as the conjunctions we could draw from the single description. For example, we may have separate descriptions of the assassination of Abraham Lincoln and of Mendel's contemporaneous experiments with peas. What we can infer about Mendel's experiments from the conjunction should ordinarily be the same as what we can infer from just the description of Mendel's experiments. Many formalisms for concurrent events don't have this property, but *conjunctivity* itself is applicable to more than concurrent events.

To use logician's language, the conjunction of the two theories should be a conservative extension of each of the theories. Actually, we may settle for less. We only require that the inferable sentences about Mendel (or about Lincoln) in the conjunction are the same. The combined theory may admit inferring other sentences in the language of one separate theory that weren't inferable in that theory.

2.9 FORMALIZED CONTEXTS

Any particular bit of thinking occurs in some context. Humans often specialize the context to particular situations or theories, and this makes the reasoning more definite, sometimes completely definite. Going the other way, we sometimes have to generalize the context of our thoughts to take some phenomena into account.

It has been worthwhile to admit contexts as objects into the ontology of logical AI. The prototype formula $\text{ist}(c, p)$ asserts that the proposition p is true in the context c . There are also formulas asserting relations between contexts that are themselves assertable in contexts. The formal theory is discussed in (McCarthy, 1993), (McCarthy and Buvač, 1998) and in papers by Saša Buvač, available in (Buvač, 1995).

2.10 APPROXIMATE CONCEPTS AND THEORIES

Common sense thinking cannot avoid concepts without clear definitions. Consider the welfare of an animal. Over a period of minutes, the welfare is fairly well defined, but asking what will benefit a newly hatched chick over the next year is ill defined. The exact snow, ice and rock that constitutes Mount Everest is ill defined. The key fact about approximate concepts is that while they are not well defined, sentences involving them may be quite well defined. For example, the proposition that Mount Everest was first climbed in 1953 is definite, and its definiteness is not compromised by the ill-definedness of the exact boundaries of the mountain. See (McCarthy, 1999a).

There are two ways of regarding approximate concepts. The first is to suppose that there is a precise concept, but it is incompletely known. Thus we may

suppose that there is a truth of the matter as to which rocks and ice constitute Mount Everest. If this approach is taken, we simply need weak axioms telling what we do know but not defining the concept completely.

The second approach is to regard the concept as intrinsically approximate. There is no truth of the matter. One practical difference is that we would not expect two geographers independently researching Mount Everest to define the same boundary. They would have to interact, because the boundaries of Mount Everest are yet to be defined.³

Up to this point our discussion of approximate concepts parallels that of vagueness in the philosophical literature. See (Alston, 1967). However, AI requires more. (McCarthy, 1999a) discusses formalizing the relations between different approximate concepts of the same entities. Its goal is to build firm logical towers on foundations of semantic quicksand.

Any theory involving approximate concepts is an approximate theory. We can have a theory of the welfare of chickens. However, its notions don't make sense if pushed too far. For example, animal rights people assign some rights to chickens but cannot define them precisely. It is not presently apparent whether the expression of approximate theories in mathematical logical languages will require any innovations in mathematical logic. See (McCarthy, 1999a). (McCarthy, 1999a) discusses the logical expression of the relations among theories at different levels of approximation.

2.11 USEFUL COUNTERFACTUALS

"If another car had come over the hill when you passed that Mercedes, there would have been a head-on collision." One's reaction to believing that counterfactual conditional sentence is quite different from one's reaction to the corresponding material conditional. Machines need to represent such sentences in order to learn from not-quite-experiences. See (Costello and McCarthy, 1998).

2.12 AMBIGUITY TOLERANCE

Assertions often turn out to be ambiguous with the ambiguity only being discovered many years after the assertion was enunciated. For example, it is *a priori* ambiguous whether the phrase "conspiring to assault a Federal official" covers the case when the criminals mistakenly believe their intended victim is a Federal official or when the victim is a federal official, but they didn't know it. These cases correspond to the *de re* and *de dicto* interpretations of "federal official" in the phrase "conspire to assault a federal official". An ambiguity in a law does not invalidate it in the cases where it can be considered unambiguous. Even where it is formally ambiguous, it is subject to judicial interpretation. AI systems will also require means of isolating ambiguities and also contradictions. The default rule is that the concept is not ambiguous in the particular case.

³Regarding a concept as intrinsically approximate is distinct from either regarding it as fully defined by nature or fully defined by human convention.

The ambiguous theories are a kind of approximate theory. It should be possible for an AI system, just as it is for people, to operate without recognition of the possibility of ambiguity until an ambiguity appears. (McCarthy, 1989) discusses ambiguity tolerance at somewhat greater length.

2.13 UNDERSTANDING

Ordinary language uses “understand” in a variety of ways, some of which are quite simple, e.g. “This version of Unix doesn’t understand some of the parameters of the *ls* command.” However, it also refers to understanding French, understanding calculus, or understanding situation calculus.

AI will require such stronger notions of understanding. Thus fish do not understand swimming, because they can’t use knowledge to improve their swimming, to wish for better fins, or to teach other fish. See the section on understanding in (McCarthy, 1996a). Maybe fish do learn to improve their swimming, but this presumably consists primarily of the adjustment of parameters and isn’t usefully called understanding. I would apply the term *understanding*, or maybe we should call it *strong understanding*, only to some systems that can do hypothetical reasoning—if *p* were true, then *q* would be true. Thus Fortran compilers don’t strongly understand Fortran.

3 HEURISTICS

3.1 DECLARATIVE EXPRESSION OF HEURISTICS

(McCarthy, 1959) proposes reasoning be controlled by domain-dependent and problem-dependent heuristics expressed declaratively. Expressing heuristics declaratively means that a sentence about a heuristic can be the result of reasoning and not merely something put in from the outside by a person. Josefina Sierra (Sierra, 1998b), (Sierra, 1998a), (Sierra, 1998c), (Sierra, 1999) has made some recent progress.

3.2 REIFICATION A.K.A. ONTOLOGY

To reify an entity is to “make a thing” out of it (from Latin *re* for *thing*). From a logical point of view, things are what variables can range over. Logical AI needs to *reify* hopes, intentions and “things wrong with the boat”. Some philosophers deplore reification, referring to a “bloated ontology”, but AI needs more things than are dreamed of in the philosophers’ philosophy. In general, reification gives a language more expressive power, because it permits referring to entities directly that were previously mentionable only in a metalanguage.

In philosophy, ontology is the branch that studies what things exist. W.V.O. Quine’s view is that the ontology is what the variables range over. Ontology has been used variously in AI, but I think Quine’s usage is best for AI. “Reification” and “ontology” treat the same phenomena. Regrettably, the word “ontology” has become popular in AI in much vaguer senses. Ontology and reification are basically the same concept.

3.3 DISCRIMINATION, RECOGNITION AND DESCRIPTION

Discrimination is the deciding which category a stimulus belongs to among a fixed set of categories, e.g. decide which letter of the alphabet is depicted in an image. *Recognition* involves deciding whether a stimulus belongs to the same set, i.e. represents the same object, e.g. a person, as a previously seen stimulus. *Description* involves describing an object in detail appropriate to performing some action with it, e.g. picking it up by the handle or some other designated part. Description is the most ambitious of these operations and has been the forte of logic-based approaches—as compared to connectionist or neural net approaches.

3.4 QUALITATIVE REASONING

This concerns reasoning about physical processes when the numerical relations required for applying the formulas of physics are not known. Most of the work in the area assumes that information about what processes to take into account are provided by the user. Systems that must be given this information often won't do human level qualitative reasoning. See (Weld and de Kleer, 1990) and (Kuipers, 1994).

3.5 PROBABILISTIC REASONING

Probabilistic reasoning is a kind of nonmonotonic reasoning. If the probability of one sentence is changed, say given the value 1, other sentences that previously had high probability may now have low or even 0 probability. Setting up the probabilistic models, i.e defining the sample space of “events” to which probabilities are to be given often involves more general nonmonotonic reasoning, but this is conventionally done by a person informally rather than by a computer.

In the open common sense informatic situation, there isn't any apparent overall sample space. Probabilistic theories may be formed by limiting the space of events considered and then establishing a distribution. Limiting the events considered should be done by whatever nonmonotonic reasoning techniques are developed techniques for limiting the phenomena taken into account. (You may take this as a confession that I don't know these techniques.) In forming distributions, there would seem to be a default rule that two events e_1 and e_2 are to be taken as independent unless there is a reason to do otherwise. e_1 and e_2 can't be just any events but have to be in some sense basic events.

3.6 NONMONOTONIC REASONING

Both humans and machines must draw conclusions that are true in the “*best*” models of the facts being taken into account. Several concepts of *best* are used in different systems. Many are based on minimizing something. When new facts are added, some of the previous conclusions may no longer hold. This is why the reasoning that reached these conclusions is called nonmonotonic.

Circumscription. A method of nonmonotonic reasoning involving minimizing predicates (and sometimes domains). It was introduced in (McCarthy, 1977), (McCarthy, 1980) and (McCarthy, 1986). An up-to-date discussion, including numerous variants, is (Lifschitz, 1994).

Default Logic. A method of nonmonotonic reasoning introduced in (Reiter, 1980) that is the main survivor along with circumscription.

Logic Programming. Logic programming isolates a subdomain of first order logic that has nice computational properties. When the facts are described as a logic program, problems can often be solved by a standard program, e.g. a Prolog interpreter, using these facts as a program. Unfortunately, in general the facts about a domain and the problems we would like computers to solve have that form only in special cases.

4 ROBOTS

We can generalize the notion of a robot as a system with a variant of the physical capabilities of a person, including the ability to move around, manipulate objects and perceive scenes, all controlled by a computer program. More generally, a robot is a computer-controlled system that can explore and manipulate an environment that is not part of the robot itself and is, in some important sense, larger than the robot. A robot should maintain a continued existence and not reset itself to a standard state after each task. From this point of view, we can have a robot that explores and manipulates the Internet without it needing legs, hands and eyes. The considerations of this chapter that mention robots are intended to apply to this more general notion. The internet robots discussed so far are very limited in their mentalities.

4.1 LOGICAL ROBOT

(McCarthy, 1959) proposed that a robot be controlled by a program that infers logically that a certain action will advance its goals and then does that action. This approach was implemented in (Green, 1969b), but the program was very slow. Shortly greater speed was obtained in systems like STRIPS at the cost of limiting the generality of facts the robot takes into account. See (Nilsson, 1984), (Levesque et al., 1997), and (Shanahan, 1996).

4.2 CONSCIOUSNESS, AWARENESS AND INTROSPECTION

Human level AI systems will require these qualities in order to do tasks we assign them. In order to decide how well it is doing, a robot will need to be able to examine its goal structure and the structure of its beliefs from the *outside*, i.e. to regard its current set of sentences as an object about which it can generate further sentences. See (McCarthy, 1996a).

4.3 INTENTION TO DO SOMETHING

Intentions as objects are discussed briefly in (McCarthy, 1989) and (McCarthy, 1996a).

4.4 MENTAL SITUATION CALCULUS

The idea is that there are mental situations, mental fluents and mental events that give rise to new mental situations. The mental events include observations and inferences but also the results of observing the mental situation up to the current time. This allows drawing the conclusion that there isn't yet information needed to solve a certain problem, and therefore more information must be sought outside the robot or organism. (Scherl and Levesque, 1993) treats this and so does (McCarthy, 1996a).

4.5 ROBOTIC FREE WILL

Robots need to consider their choices and decide which of them leads to the most favorable situation. In doing this, the robot considers a system in which its own outputs are regarded as free variables, i.e. it doesn't consider the process by which it is deciding what to do. The perception of having choices is also what humans consider as *free will*. The matter is discussed in (McCarthy and Hayes, 1969) and is roughly in accordance with the philosophical attitude towards free will called *compatibilism*, i.e. the view that determinism and free will are compatible.

4.6 DENNETT'S THREE STANCES

The philosopher Daniel Dennett (Dennett, 1978) proposed three stances or attitudes that an observer may take towards a system.

The *design stance* regards an entity in terms of its function rather than in terms of its physical structure. For example, a traveler using a hotel alarm clock need not notice whether the clock is controlled by a mechanical escapement, the 60 cycle power line or by an internal crystal. We formalize it in terms of (a) the fact that it can be used to wake the traveler, and (b) setting it and the noise it makes at the time for which it is set.

The *physical stance* considers an object in terms of its physical structure. This is needed for actually building it or repairing it but is often unnecessary in making decisions about how to use it.

When we take the *intentional stance* we consider the behavior of a person, animal or machine by ascribing to it belief, desires and intentions. This is discussed in (Dennett, 1971) and (Dennett, 1978) and also in (McCarthy, 1979). The latter paper shows that it is sometimes useful to ascribe some mental qualities to quite simple systems, the extreme example of which is ascribing to a thermostat one of three possible beliefs—it's too cold, too hot or ok.

5 REASONING ABOUT ACTION

A major concern of logical AI has been treating the consequences of actions and other events. The *epistemological* problem concerns what can be known about the laws that determine the results of events. A theory of causality is pretty sure to be approximate. (Shanahan, 1997) describes the current situation.

5.1 SITUATION CALCULUS

Situation calculus is the most studied formalism for doing causal reasoning. A situation is in principle a snapshot of the world at an instant. One never knows a situation—one only knows facts about a situation. Events occur in situations and give rise to new situations. There are many variants of situation calculus, and none of them has come to dominate. (McCarthy and Hayes, 1969) introduces situation calculus. (Gelfond et al., 1991) is a 1991 discussion. Raymond Reiter and Fiora Pirri (Reiter, 1993) and (Pirri and Reiter, 1999), and (Levesque et al., 1998) use a situation calculus formalism in which the situations form a tree based in an original situation S_0 in which the edges are indexed by actions.

Fluents. Functions of situations in situation calculus. The simplest fluents are *propositional* and have truth values. There are also fluents with values in numerical or symbolic domains. *Situational fluents* take on situations as values.

Yale Shooting Problem. This problem, introduced in (Hanks and McDermott, 1986), is a simple *Drosophila* for nonmonotonic reasoning. The simplest formalizations of causal reasoning using circumscription or default logic for doing the nonmonotonic reasoning do not give the result that intuition demands. Various more recent formalizations of events handle the problem ok. The Yale shooting problem is likely to remain a benchmark problem for formalizations of causality.

5.2 THE THREE LONG STANDING PROBLEMS

The literature on the theory of action has considered three main problems with which each formalism has to deal.

The *frame problem* concerns how to express the facts about the effects of actions and other events in such a way that it is not necessary to explicitly state for every event, the fluents it does not affect. Murray Shanahan (Shanahan, 1997) has an extensive discussion.

The *qualification problem* concerns how to express the preconditions for actions and other events. That it is necessary to have a ticket to fly on a commercial airplane is rather unproblematical to express. That it is necessary to be wearing clothes needs to be kept inexplicit unless it somehow comes up.

Events often have other effects than those we are immediately inclined to put in the axioms concerned with the particular kind of event. This is the *ramification problem*.

5.3 APPLICATIONS OF THEORIES OF ACTION

Theories of action can be applied in three ways.

Projection. Given information about a situation, and axioms about the effects of actions and other events, the projection problem is to determine facts about future situations. It is assumed that no facts are available about future situations other than what can be inferred from the “known laws of motion” and what is known about the initial situation. Query: how does one tell a reasoning system that the facts are such that it should rely on projection for information about the future?

Planning. The largest single domain for logical AI has been planning, usually the restricted problem of finding a finite sequence of actions that will achieve a goal. (Green, 1969a) is the first paper to use a theorem prover to do planning. Planning is somewhat the inverse problem to projection.

Narrative. A narrative tells what happened, but any narrative can only tell a certain amount. What narratives can tell, how to express that logically, how to elaborate narratives, and what questions understanding a narrative permits answering—is given a preliminary logical treatment in (McCarthy, 1995b) and more fully in (McCarthy and Costello, 1998).

(Pinto and Reiter, 1993) and (R.S.Miller and M.P.Shanahan, 1994) are relevant here. A narrative will usually give facts about the future of a situation that are not just consequences of projection from an initial situation. [While we may suppose that the future is entirely determined by the initial situation, our knowledge doesn't permit inferring all the facts about it by projection. Therefore, narratives usually give facts about the future beyond what follows by projection from the events mentioned.]

How it Happened. Actions and other events are rich objects with infinite detail. Therefore, formalisms for describing events need to allow elaborations that give more detail. Consider an action like buying a pack of cigarettes on a particular occasion and the subactions thereof. The relation between the action and its details is not like that between a program and its subroutines. On one occasion I might have bought the cigarettes from a machine, on a second occasion at a supermarket, and on a third occasion from a cigarettelegger, cigarettes having become illegal.

5.4 DISCRETE AND CONTINUOUS PROCESSES

Causal reasoning is simplest when applied to processes in which discrete events occur and have definite results. In situation calculus, the formulas $s' = \text{result}(e, s)$ gives the new situation s' that results when the event e occurs in situation s . Many continuous processes that occur in human or robot activity can have *approximate theories* that are discrete.

Humans approximate continuous processes with representations that are as discrete as possible. For example, "Junior read a book while on the airplane from Glasgow to London." Continuous processes can be treated in the situation calculus, but the theory is so far less successful than in discrete cases. We also sometimes approximate discrete processes by continuous ones. (Miller, 1996) and (Reiter, 1996) treat this problem.

5.5 NON-DETERMINISTIC EVENTS

Situation calculus and other causal formalisms are harder to use when the effects of an action are indefinite. Often $\text{Result}(e, s)$ is not usefully axiomatizable and something like $\text{Occurs}(e, s)$ must be used.

5.6 CONCURRENT EVENTS

Formalisms treating actions and other events must allow for any level of dependence between events. Complete independence is a limiting case and is treated in (McCarthy, 1995b) and (McCarthy and Costello, 1998).

6 LEARNING

Making computers learn presents two problems—*epistemological* and *heuristic*. The epistemological problem is to define the space of concepts that the program can learn. The heuristic problem is the actual learning algorithm. The heuristic problem of algorithms for learning has been much studied and the epistemological mostly ignored. The designer of the learning system makes the program operate with a fixed and limited set of concepts. Learning programs will never reach human level of generality as long as this approach is followed. (McCarthy, 1959) says, "A computer can't learn what it can't be told." We might correct this, as suggested by Murray Shanahan, to say that it can only learn what can be expressed in the language we equip it with. To learn many important concepts, it must have more than a set of weights. (Muggleton and De Raedt, 1994) and (Bratko and Muggleton, 1995) present some progress on learning within a logical language. The many kinds of learning discussed in (Mitchell, 1997) are all, with the possible exception of inductive logic programming, very limited in what they can represent—and hence can conceivably learn. (McCarthy, 1999b) presents a challenge to machine learning problems and discovery programs to learn or discover the reality behind appearance.

7 CREATIVITY

Humans are sometimes creative—perhaps rarely in the life of an individual and among people. What is creativity? We consider creativity as an aspect of the solution to a problem rather than as attribute of a person or computer program, the latter being too hard for now.

A creative solution to a problem contains a concept not present in the functions and predicates in terms of which the problem is posed. (McCarthy, 1964) and (McCarthy, 1999c) discuss the creativity of solutions to the mutilated checkerboard problem.

The problem is to determine whether a checkerboard with two diagonally opposite squares removed can be covered with dominoes, each of which covers two rectilinearly adjacent squares. The standard proof that this can't be done is *creative* relative to the statement of the problem. It notes that a domino covers two squares of opposite color, but there are 32 squares of one color and 30 of the other color to be colored.

Colors are not mentioned in the statement of the problem, and their introduction is a creative step relative to this statement. For a mathematician of moderate experience (and for many other people), this bit of creativity is not difficult. We must, therefore, separate the concept of creativity from the concept of difficulty.

Before we can have creativity we must have some elaboration tolerance. Namely, in the simple language of *A tough nut . . .*, the colors of the squares cannot even be expressed. A program confined to this language could not even be told the solution. As discussed in (McCarthy, 1996b), Zermelo-Frankel set theory is an adequate language. Set theory, in a form allowing definitions may have enough elaboration tolerance. Regard this as a conjecture that requires more study.

8 ACKNOWLEDGMENTS AND REMARKS

I am grateful to Murray Shanahan for many useful suggestions, encouraging me to finish this chapter, and for much of the bibliography. Aarati Parmar has given me extensive help in both the content and the form of this version. I also thank the referee for his comments, to most of which I have been able to respond.

This work was partly supported by ARPA (ONR) grant N00014-94-1-0775 and AFOSR ARPA Grant No. 169J266.

The web form of this chapter has links to other articles of mine. I'd like to supplement the normal references by direct links to such articles as are available.

References

- Alston, W. P. (1967). Vagueness. In Edwards, P., editor, *Encyclopedia of Philosophy*, volume 8, pages 218–221. MacMillan.
- Bratko, I. and Muggleton, S. (1995). Applications of inductive logic programming. *Communications of the ACM*, 38(11):65–70.

- Buvač, S. (1995). Saša buvač's web page. Can be found at <http://www-formal.stanford.edu/buvac/>.
- Costello, T. and McCarthy, J. (1998). Useful counterfactuals and approximate theories. In *AAAI Spring Symposium on Prospects for a Commonsense theory of Causation*. AAAI Press.
- Davis, R., Buchanan, B., and Shortliffe, E. (1977). Production rules as a representation for a knowledge-based consultation program. *Artificial Intelligence*, 8(1):15–45.
- Dennett, D. (1978). *Brainstorms: Philosophical Essays on Mind and Psychology*. Bradford Books/MIT Press, Cambridge.
- Dennett, D. C. (1971). Intentional systems. *The Journal of Philosophy*, 68(4):87–106.
- Weld, D. S. and de Kleer, J., editors (1990). *Readings in Qualitative Reasoning about Physical Systems*. Morgan-Kaufmann.
- Gelfond, M., Lifschitz, V., and Rabinov, A. (1991). What are the limitations of the situation calculus? In Boyer, R., editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 167–179. Kluwer Academic, Dordrecht.
- Green, C. (1969a). Applications of theorem proving to problem solving. In *Proceedings IJCAI 69*, pages 219–240.
- Green, C. (1969b). 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.
- Hanks, S. and McDermott, D. (1986). Default reasoning, nonmonotonic logics and frame problem. In *Proceedings of AAAI-86*, pages 328–333. Morgan Kaufmann.
- Hayes, P. J. (1985). The second naive physics manifesto. In J.R., H. and R.C., M., editors, *Formal Theories of the Commonsense World*, pages 1–36. Ablex.
- Kuipers, B. (1994). *Qualitative Reasoning*. MIT Press.
- Lenat, D. B. and Guha, R. V. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley.
- Levesque, H., Pirri, F., and Reiter, R. (1998). Foundations for the situation calculus. *Linkping Electronic Articles in Computer and Information Science, ISSN 1401-9841*, 3(18).
Can be found at <http://www.ep.liu.se/ea/cis/1998/018/>.
- Levesque, H. J., Reiter, R., Lesprance, I., Lin, F., and Scherl, R. B. (1997). Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–83.
- Lifschitz, V. (1994). Circumscription. In *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*. Oxford University Press.
- Lifschitz, V. (2000). Missionaries and cannibals in the causal calculator. In Cohn, A. G., Giunchiglia, F., and Selman, B., editors, *KR2000: Principles of Knowledge Representation and Reasoning, Proceedings of the Seventh International Conference*, pages 85–96.
- McCarthy, J. (1959). Programs with commonsense. 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 McCarthy, 1990. Can be found at
<http://www-formal.stanford.edu/jmc/mcc59.html>.
- McCarthy, J. (1964). A tough nut for theorem provers. Stanford AI Memo 16—now on the web at <http://www-formal.stanford.edu/jmc/nut.html>.
- McCarthy, J. (1977). Epistemological problems in artificial intelligence. In *Proc. 5th International Conference on Artificial Intelligence*, pages 1038–1044.
- McCarthy, J. (1979). Ascribing mental qualities to machines. In Ringle, M., editor, *Philosophical Perspectives in Artificial Intelligence*. Harvester Press. Reprinted in McCarthy, 1990. Can be found at
<http://www-formal.stanford.edu/jmc/ascribing.html>.
- McCarthy, J. (1980). Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39. Reprinted in McCarthy, 1990. Can be found at
<http://www-formal.stanford.edu/jmc/circumscription.html>.
- McCarthy, J. (1983). Some expert systems need common sense. In Pagels, H., editor, *Computer Culture: The Scientific, Intellectual and Social Impact of the Computer*, volume 426. Annals of the New York Academy of Sciences. Can be found at <http://www-formal.stanford.edu/jmc/someneed.html>.
- McCarthy, J. (1986). Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 28:89–116. Reprinted in McCarthy, 1990. Can be found at <http://www-formal.stanford.edu/jmc/applications.html>.
- McCarthy, J. (1989). Artificial intelligence, logic and formalizing common sense. In Thomason, R., editor, *Philosophical Logic and Artificial Intelligence*. Klüver Academic. Can be found at <http://www-formal.stanford.edu/jmc/ailogic.html>.
- McCarthy, J. (1990). *Formalizing Common Sense: Papers by John McCarthy*. Ablex Publishing Corporation.
- McCarthy, J. (1993). Notes on formalizing context. In *IJCAI-93*. Can be found at <http://www-formal.stanford.edu/jmc/context.html>.
- McCarthy, J. (1995a). Partial formalizations and the lemmings game. Technical report, Stanford University, Formal Reasoning Group.
 Can be found at <http://www-formal.stanford.edu/jmc/lemmings.html>.
- McCarthy, J. (1995b). Situation calculus with concurrent events and narrative. Web only, partly superseded by McCarthy and Costello, 1998. Can be found at <http://www-formal.stanford.edu/jmc/narrative.html>.
- McCarthy, J. (1996a). Making robots conscious of their mental states. In Mugleton, S., editor, *Machine Intelligence 15*. Oxford University Press. to appear in 2000. The web version
<http://www-formal.stanford.edu/jmc/consciousness.html>
 is improved from that presented at Machine Intelligence 15 in 1995.
- McCarthy, J. (1996b). The mutilated checkerboard in set theory. presented at a 1996 conference in Warsaw. Can be found at
<http://www-formal.stanford.edu/jmc/checkerboard.html>.
- McCarthy, J. (1997). Elaboration tolerance. In *McCarthy's web page*. Can be found at <http://www-formal.stanford.edu/jmc/elaboration.html>.

- McCarthy, J. (1999a). Logical theories with approximate concepts—draft. submitted but web only for now. Can be found at
<http://www-formal.stanford.edu/jmc/approximate.html>.
- McCarthy, J. (1999b). Appearance and reality. web only for now, and perhaps for the future. not publishable on paper, because it contains an essential imbedded applet. Can be found at
<http://www-formal.stanford.edu/jmc/appearance.html>.
- McCarthy, J. (1999c). Creative solutions to problems. web only for now. given at AISB Workshop on AI and Scientific Creativity, Edinburgh, 1999 April. Can be found at <http://www-formal.stanford.edu/jmc/creative.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.
- McCarthy, J. and Costello, T. (1998). Combining narratives. In *Proceedings of Sixth Intl. Conference on Principles of Knowledge Representation and Reasoning*. Morgan-Kaufman.
- 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. Can be found at
<http://www-formal.stanford.edu/jmc/mchay69.html>.
- Miller, R. S. (1996). A case study in reasoning about actions and continuous change. In *Proceedings ECAI 96*, pages 624–628.
- Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.
- Muggleton, S. and De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679.
- Newell, A. (1982). The knowledge level. *AI*, 18(1):87–127.
- Newell, A. (1993). Reflections on the knowledge level. *Artificial Intelligence*, 59(1-2):31–38.
- Nilsson, N. J. (1984). Shakey the robot, SRI technical note no. 323. Technical report, SRI International, Menlo Park, California.
- Pinto, J. and Reiter, R. (1993). Temporal reasoning in logic programming: A case for the situation calculus. In *Proceedings of the Tenth International Conference on Logic Programming*, pages 203–221.
- Pirri, F. and Reiter, R. (1999). Some contributions to the metatheory of the situation calculus. *JACM*, 46(3):261–325. Can be found at
<http://www.cs.utoronto.ca/~cogrobo>.
- Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13 (1–2):81–132.
- Reiter, R. (1993). Proving properties of states in the situation calculus. *Artificial Intelligence*, 64:337–351. Can be found at
<http://www.cs.utoronto.ca/~cogrobo>.
- Reiter, R. (1996). Natural actions, concurrency and continuous time in the situation calculus. In: Aiello, L. C., Doyle J., Shapiro S. C. (Eds.): *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*. Morgan-Kaufman.

- tation and Reasoning (KR'96), Cambridge, Massachusetts, USA, November 5-8, 1996, pages 2-13. Morgan Kaufmann.
- R.S.Miller and M.P.Shanahan (1994). Narratives in the situation calculus. *Journal of Logic and Computation*, 4(5):513-530.
- Scherl, R. and Levesque, H. (1993). The frame problem and knowledge producing actions. In *Proceedings AAAI 93*, pages 689-695.
- Shanahan, M. (1997). *Solving the Frame Problem, a mathematical investigation of the common sense law of inertia*. M.I.T. Press.
- Shanahan, M. P. (1996). Robotics and the common sense informatic situation. In *Proceedings ECAI 96*, pages 684-688.
- Sierra, J. (1998a). Declarative formalization of heuristics, chapter 11, pages 1-8. In *Workshop on Validation and Verification of Knowledge Based Systems KBS V&V'98*.
- Sierra, J. (1998b). Declarative formalization of strategies for action selection. In *Seventh International Workshop on Nonmonotonic Reasoning, NM/-98*, pages 21-29.
- Sierra, J. (1998c). Declarative formalization of STRIPS. In *Thirteenth European Conference on Artificial Intelligence, ECAI-98*, pages 509-513.
- Sierra, J. (1999). Declarative formalization of heuristics (taking advice in the blocks world). In *International Conference on Computational Intelligence for Modelling Control and Automation*, pages 221-228.

III

KNOWLEDGE REPRESENTATION

Chapter 3

TWO APPROACHES TO EFFICIENT OPEN-WORLD REASONING

Giuseppe De Giacomo,

Dipartimento di Informatica e Sistemistica

Università di Roma "La Sapienza"

Via Salaria 113, 00198 Rome, Italy

degiacomo@dis.uniroma1.it

Hector Levesque

Department of Computer Science

University of Toronto

Toronto, Canada M5S 3H5

hector@cs.toronto.edu

Abstract We show how a simple but efficient evaluation procedure that is logically correct only for closed-world knowledge bases can nonetheless be used in certain contexts with open-world ones. We discuss two cases, one based on restricting queries to be in a certain normal form, and the other, arising in reasoning about actions, based on having sensing information at the right time so as to dynamically reduce open-world reasoning to closed-world reasoning.

Keywords: Automated reasoning, computational tractability, open-world database, closed-world database, incomplete knowledge, sensing information

1 INTRODUCTION

From the very beginnings of AI, the dream of getting a machine to exhibit common sense was linked to deductive reasoning:

We shall therefore say that a program has common sense if it automatically deduces for itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows.

— John McCarthy in (McCarthy, 1968)

Since then, the enthusiasm for deduction has been tempered somewhat by what has been discovered about its computational difficulty. Regardless of how one

feels about the relevance of complexity and computability theory to AI, for knowledge bases (KBs) large enough to hold what is presumed to be necessary for human-level common sense, deduction would have to be *extremely* efficient. Recent local search based methods like GSAT (Selman et al., 1992) do show some promise on large KBs, but so far (1) they are restricted to constraint satisfaction tasks not deductive ones, and (2) they work only on problems that can be formulated in a propositional language.¹

To the best of our knowledge, there is so far only one logically correct (sound and complete) deductive technique efficient enough to be feasible on KBs of this size: the deduction underlying database query answering. In KR terms, this amounts to what was called *vivid reasoning* in (Levesque, 1986). In logical terms, the requirements for this form of reasoning are clear: every relevant atomic formula must be known to be true or known to be false. That is, the KB must be equivalent to a maximally consistent set of literals. In addition, this set of literals must be readily computable. In the propositional case, one obvious way of ensuring this is to store the positive ones in a database and infer the negative ones using negation as failure. With every atom known true or known false, it then follows that every formula can be "efficiently" (in a sense to be discussed later) determined to be true or to be false by *evaluating* it, that is, by calculating its truth value as a function of the truth values of its constituent atoms.

But this requirement for complete knowledge is very strict. It would certainly be desirable to allow some atomic formulas to be *unknown*, with the understanding that other formulas would need to be unknown as well. Allowing arbitrary disjunctions (or existential quantifications) in the KB would obviously require a very different method of reasoning. A less radical move, which still allows incomplete knowledge, is to consider a KB that is equivalent to a finite consistent set of literals, not necessarily maximal. Unfortunately, although this is a trivial extension to the above, we can already see that it will not work: for the special case of a KB equivalent to the empty set of literals, the formulas that would need to be known are precisely the *valid* ones. Computing these is co-NP hard in the propositional case, and even if we accept the argument that it may still be feasible in practice (perhaps because the query will always be small, or for reasons like those discussed in (Hogg et al., 1996)), there is no escaping the fact that it would be undecidable in the first-order case.

So it appears that even a seemingly insignificant increase in expressive power, allowing for the most basic form of incompleteness in the KB, already makes deduction too hard. Despite this, it is precisely this form of incomplete knowledge that we will consider in this paper, suitably generalized to deal with quantifiers and equality. We refer to the sort of reasoning required as *open-world reasoning*, to distinguish it from closed-world reasoning where every formula is known

¹Converting first-order reasoning problems into propositional ones remains a possibility (as done in (Kautz et al., 1996), for example), but consider that for KBs with (say) 10^6 unique names, even a single binary predicate would generate far too many atomic propositions. Recent work on satisfiability with restricted first-order formulas may help here (Parkes, 1999).

to be true or known to be false on the one hand, and fully general reasoning, which allows for the presence of disjunctions, existentials *etc.*, on the other.²

What we will argue is that open-world reasoning is a middle ground that can be dealt with effectively (sometimes) using two complementary approaches:

- by restricting the class of queries to a special normal form (\mathcal{NF}), a simple evaluation procedure provides inference that is both logically sound and complete;
- by assuming that we have sensing information, *i.e.*, information coming from outside the system, available at the right time, we can reduce an otherwise open-world reasoning problem to a closed-world one, and again use the simple evaluation procedure.

Here we describe the two approaches and state the main theorems without proof. Further technical details on the two approaches can be found in (Levesque, 1998) and (De Giacomo and Levesque, 1999) respectively.

2 EVALUATION-BASED REASONING

The reasoning procedure we have in mind (for KBs with both complete and incomplete knowledge) is one that decides whether a formula is true or false by evaluating it, reducing knowledge of complex formulas to knowledge of the ground atomic formulas, \mathcal{A} .³ Throughout, we will use 0 to mean “known to be false,” 1 to mean “known to be true,” and $\frac{1}{2}$ to mean “unknown.”⁴

Given an assignment $V \in [\mathcal{A} \rightarrow \{0, 1, \frac{1}{2}\}]$ telling us which atoms are known, we extend the assignment to all boolean formulas in the obvious way:

1. $V[\neg\alpha] = 1 - V[\alpha]$.
2. $V[\alpha \wedge \beta] = \min\{V[\alpha], V[\beta]\}$.

Disjunctions, implications, and equivalences can be handled as abbreviations. We will sometimes also use the logical constant *TRUE*, with $V[TRUE] = 1$.

To handle quantification, assume we are given a finite set H of constants (intuitively, those names are mentioned in some KB), and we define

$$3. V[\forall x.\alpha] = \min_{c \in H^+} \{V[\alpha_c^x]\}$$

²It is interesting to observe that open-world reasoning has attracted interest of the database community as well (*e.g.*, (Imielinski and Lipski, 1984; Reiter, 1984; Vardi, 1985)). More recently, researchers have looked at the problem of answering queries using only information on a given set of views, *e.g.*, answering queries using only a given set of materialized views (*e.g.*, (Levy et al., 1995; Abiteboul and Duschka, 1998; Grahne and Mendelzon, 1999)). This also is a form of reasoning with incomplete information.

³Unless otherwise specified, by an atom, we do not include equality formulas. These are handled separately below.

⁴If we were to allow for inconsistent KBs as well, we would have a *fourth* truth value, as in (Belnap, 1977; Cadoli and Schaerf, 1992; Dunn, 1976; Ginsberg, 1988; Lakemeyer, 1990; Levesque, 1984; Patel-Schneider, 1985), among many others. From an efficiency point of view, nothing is gained by this move, so we forego it for simplicity.

Here α_c^x is the result of replacing free x by c in α , and H^+ is the union of the constants in H , those mentioned in α , and one new one outside of H and not mentioned in α . Thus, to evaluate $\forall x.\alpha$, we evaluate a finite set of its instances where the x ranges over the constants in the given H , over the constants mentioned in α , and over one new constant that is neither in H nor in α . We handle existentials as abbreviations.

Finally to handle equality formulas, we use the simplest possible scheme (for ground atomic ones):

4. $V[t = t'] = 1$ if t is identical to t' , and 0 otherwise.

So all that is left to completely determine a V function is the set H and the value of V on atomic formulas. We will show how to get these from a given KB in Section 3.3. Then, using these four rules, we can evaluate any closed formula, that is, compute what is known about the formula as a function of what is known about instances of its atoms.

Of course it remains to be seen in what contexts this 3-valued evaluation scheme can be used. This is what is addressed in Sections 3.4 and 3.5.

We should be clear about what we mean by correctness. We will want to talk about making deductions from a set of formulas S (the KB), and getting the correct answer (0, 1, or $\frac{1}{2}$) for a class of formulas T (the potential queries):

Definition 1 Let $S, T \subseteq \mathcal{L}$, and let $f \in [\mathcal{L} \rightarrow \{0, 1, \frac{1}{2}\}]$. Then

- f is logically sound wrt S for T iff for every $\alpha \in T$, if $f[\alpha] = 1$ then $S \models \alpha$, and if $f[\alpha] = 0$ then $S \models \neg\alpha$;
- f is logically complete wrt S for T iff for every $\alpha \in T$, if $S \models \alpha$ then $f[\alpha] = 1$, and if $S \models \neg\alpha$ then $f[\alpha] = 0$;
- f is logically correct wrt S for T iff it is both sound and complete.

We will see below (after we establish some properties of quantifiers and equality) that whenever we begin with an evaluation function that is logically sound for atomic formulas, it will end up logically sound for all formulas. But this will not be the case for logical completeness: it is a well known property of multi-valued logics (Urquhart, 1986) that classically correct answers for atoms do not guarantee correctness for all formulas.

Observe, for example, that we would want $V[p \vee \neg p]$ to be 1 even when $V[p] = \frac{1}{2}$, contrary to what we have above. This has suggested to some authors that perhaps tautologies and their negations need to be filtered out separately in the evaluation (as in (Vassiliou, 1980) and in supervaluations (Van Fraasen, 1966)).

But the problem is not merely with tautologies. Suppose we have that $V[p] = \frac{1}{2}$, $V[q] = 1$ and $V[r] = 0$ (where e.g. $KB = \{q, \neg r\}$). Let α be the formula

$$(q \wedge (\neg r \wedge p)) \vee (\neg p \wedge (\neg r \wedge q)).$$

Then, we get $V[\alpha] = \frac{1}{2}$, whereas completeness requires a value of 1 (since $KB \models \alpha$). There is, however, a tautology hidden here: if we convert α to CNF,

we get

$$[q \wedge \neg r \wedge (p \vee \neg p)],$$

which gives a value of 1, after we filter out the tautologous clause.

But consider the dual of α : $[(\neg q \vee r \vee p) \wedge (\neg q \vee r \vee \neg p)]$. For logical completeness, this should get value 0, although again V returns $\frac{1}{2}$. Moreover, the formula here is in CNF, and there are no hidden tautologous clauses to remove.⁵ However, observe that the clause $(\neg q \vee r)$ is derivable from these two by Resolution, and if we were to conjoin this new clause to the formula, logical equivalence would be preserved and V would now return the correct answer, 0. This is the idea behind the normal form we will introduce later.

A few words on the efficiency of the above treatment of knowledge. If the query does not use quantifiers, V will ask for the value of an atom a linear (in the size of the query) number of times. So non-quantified queries are handled efficiently, assuming atoms are. But for quantified queries, the situation is less clear. Consider one like $\exists x_1 \dots \exists x_n (\rho_1 \wedge \dots \wedge \rho_m)$, where the ρ_j are atoms whose arguments are among the x_i . Even if we imagine a KB that is a simple database (a finite set of ground atoms) that uses k constants, the obvious way of handling this requires looking at all k^n vectors of constants, clearly infeasible for the sort of large k we are considering.⁶ In actual database systems, queries like this can be formulated, but they are handled in practice using a number of optimizations such as sort restrictions on variables (so that not all constants need be considered for every variable), and sophisticated implementations of relational operations (e.g., join, selections) and careful subgoal (join) ordering and selection placement. These types of optimizations will be available to us as well, and coupled with an assumption that n is very small, we take it that quantified queries can be handled efficiently (or as efficiently as can be expected), assuming again that atomic queries are.

3 A FIRST APPROACH

The first approach which will allow us to use the above evaluation procedure requires queries to be in a certain normal form. But first we must be clear about the sorts of KBs we will be using. For the purposes of this section, we start with a standard first-order language \mathcal{L} with no function symbols other than constants and a distinguished equality predicate. We assume a countably infinite set of constants $\mathcal{C} = \{c_1, c_2, \dots\}$ for which we will be making a unique-name assumption.

⁵We could convert the formula to DNF and remove the complement of tautologous clauses, and that would work here, but not in the first-order case. See below.

⁶Although it is not an issue here, the worst case complexity of this problem does not look good. Namely evaluating formulas of the form above, which are essentially conjunctive queries in databases, is polynomial in the size of the database, but NP-complete in the size of the formula (see (Chandra and Merlin, 1977)). Evaluating general first-order formulas is again polynomial in the size of the database, but PSPACE-complete in the size of the formula (see again (Chandra and Merlin, 1977)).

3.1 QUANTIFIERS AND EQUALITY

Because we will be considering KBs and queries that use equality, we will end up wanting to compute the entailments not just of the KB, but of $\mathcal{E} \cup KB$, where we have:

Definition 2 The set \mathcal{E} is the axioms of equality (reflexivity, symmetry, transitivity, substitution of equals for equals) and the (infinite) set of formulas $\{(c_i \neq c_j) \mid i \neq j\}$.

Note that because we are making a unique-name assumption for infinitely many constants, we will not be able to finitely “propositionalize” first-order KBs, despite the lack of function symbols. We will use θ to range over substitutions of all variables by constants, and write $\alpha\theta$ as the result of applying the substitutions to α . We will use ρ to range over atoms (other than equalities) whose arguments are distinct variables, so that $\rho\theta$ ranges over ground atoms. We will use $\forall\alpha$ to mean the universal closure of α . When S is finite, $\wedge S$ stands for the conjunction of its elements (and the logical constant *TRUE*, when S is empty). Finally, we will use e to range over *wffs*, by which we mean quantifier-free formulas whose only predicate is equality.

Before discussing KBs and queries, we need to establish how the quantifiers and substitution behave. First we define the notion of a standard interpretation:

Definition 3 A *standard* interpretation of \mathcal{L} is one where $=$ is interpreted as identity, and the denotation relation between \mathcal{C} and the domain of discourse is bijective.

We get the following theorem:

Theorem 1 Suppose S is any set of closed wffs, and that there is an infinite set of constants that do not appear in S . Then $\mathcal{E} \cup S$ is satisfiable iff it has a standard model.

This is like Herbrand’s Theorem (with \mathcal{C} being like the Herbrand Universe) except that S is not required to be in prenex form, can contain arbitrary alternations of quantifiers (which would otherwise introduce Skolem functions), etc. Note that this is not simply a variant of the Skolem-Lowenheim Theorem either, since our theorem does not hold when S mentions *every* constant, as in the set $\{\exists x.P(x)\} \cup \{\neg P(c) \mid c \in \mathcal{C}\}$. This is an example of a satisfiable set that has no standard model.

The second theorem concerns substitutions by constants:

Theorem 2 Let S be a set of closed wffs, let α be a wff with a single free variable x , and let H^+ be a set of constants containing those in S , those in α , and at least one constant in neither. Then for every constant $d \in \mathcal{C}$, there is a constant $c \in H^+$ such that $\mathcal{E} \cup S \models \alpha_d^x$ iff $\mathcal{E} \cup S \models \alpha_c^x$.

It is this theorem that will allow us to restrict our attention a finite set of constants in H^+ when we do substitutions, as we will show below. Note that the theorem is false if H^+ contains just the constants in S and α . For example, let α be $P(x)$, and S be $\{\forall z(z \neq a \supset P(z))\}$. In this case, the only constant in

S or α is a , and $\mathcal{E} \cup S \not\models \alpha_a^x$, but $\mathcal{E} \cup S \models \alpha_b^x$. The theorem is also false if H^+ does not contain the constants in α . For example, let α be $R(x, b)$, and S be $\{\forall y. \forall z. (y = z) \supset R(y, z)\}$. Here, $\mathcal{E} \cup S \models \alpha_b^x$, but for every other constant c , $\mathcal{E} \cup S \not\models \alpha_c^x$.

3.2 KNOWLEDGE BASES

Since we are considering a KB containing equality, variables, and universal quantifiers, we will not be able to do simple retrieval to find out what is known about the atoms. For example, let β be the formula

$$\forall x \forall y \forall z. (x \neq y \wedge z = y) \supset R(x, z, y).$$

If a KB contains β then we want $R(b, a, a)$ to be known. So we must first be clear about the form of KB we will be using:

Definition 4 We call a set S of formulas *proper* if $\mathcal{E} \cup S$ is consistent and S is a finite set of formulas of the form $\forall(e \supset \rho)$ or $\forall(e \supset \neg\rho)$, where e is an ewff, and ρ is an atom as above.

We will be interested in KBs that are proper. Observe that as a special case, we can represent any finite consistent set of literals as a proper KB: simply replace $\rho\theta$ (or its complement) by $\forall(e \supset \rho)$ where e is of the form $\wedge(x_i = c_i)$. We can also represent a variety of infinite sets of literals, as the formula β does above. We are free to characterize some of the positive instances of ρ by using $\forall(e \supset \rho)$, and leave the status of the rest open. We can do the same for negative instances. We can also make a closed world assumption about a predicate if we so choose, by using both $\forall(e \supset \rho)$ and $\forall(\neg e \supset \neg\rho)$, for some e and ρ .

It might appear that proper KBs are overly restrictive, and ought to be easy to reason with. It is worth remembering that deciding whether a proper KB entails a formula is recursively unsolvable, unless the formula is restricted in some way, as we intend to do.

Although proper sets are not the same as sets of literals, they can be used to represent them in the following way:

Definition 5 Let S be any finite set of $\forall(e \supset \alpha)$ formulas as above, but not necessarily consistent. Define

$$Lits(S) = \{\alpha\theta \mid \forall(e \supset \alpha) \in S, \mathcal{E} \models e\theta\}. \quad (3.1)$$

Then we get the following:

Theorem 3 Let S be a finite set of formulas of the above form, and let M be any standard interpretation. Then

$$M \models S \text{ iff } M \models Lits(S)$$

So S and $Lits(S)$ are satisfied by the same standard interpretations (although there will be non-standard interpretations where they diverge).

3.3 ATOMIC QUERIES

Now we want to define how atomic queries will be handled for proper KBs. We will use the fact that V has already been defined for closed cwffs, and (by a simple induction argument) satisfies the following:

Lemma 1 For any cwff e , $V[e\theta] = 1$ iff $\mathcal{E} \models e\theta$.

This establishes that V is logically correct for cwffs.

Definition 6 For any proper KB , the atomic evaluation associated with KB is the function V where the H (for handling quantifiers) is the set of constants mentioned in KB , and such that for any ground atom $\rho\theta$

$$V[\rho\theta] = \begin{cases} 1 & \text{if there is a } \forall(e \supset \rho) \in KB \\ & \text{such that } V[e\theta] = 1 \\ 0 & \text{if there is a } \forall(e \supset \neg\rho) \in KB \\ & \text{such that } V[e\theta] = 1 \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

This function is well-defined: if there were formulas $\forall(e_1 \supset \rho), \forall(e_2 \supset \neg\rho) \in KB$ such that $V[e_1\theta] = V[e_2\theta] = 1$, we would have by Lemma 1 that $\mathcal{E} \models e_1\theta \wedge e_2\theta$, and so $\mathcal{E} \cup KB \models \rho\theta \wedge \neg\rho\theta$, violating the consistency of $\mathcal{E} \cup KB$.

Furthermore, the function (as a procedure) runs in time that is no worse than linear in the size of the KB . Given the considerations discussed in the previous section, this settles the efficiency question as far as we are concerned: using the evaluation V associated with a KB , arbitrary closed queries can be answered efficiently.

We now turn to the correctness of V .

3.4 SOUNDNESS OF QUERY EVALUATION

We begin by showing that the evaluation associated with a KB always returns logically correct answers for atomic queries.

Theorem 4 For any proper KB , the evaluation associated with KB is logically correct for ground atomic queries wrt $\mathcal{E} \cup KB$.

Next we show that the evaluation associated with a proper KB always returns logically sound answers for any query:

Theorem 5 Suppose KB is proper. Then the evaluation associated with KB is logically sound for any closed formula wrt $\mathcal{E} \cup KB$.

However, as we already argued, we cannot expect to have logical correctness when knowledge is incomplete. In the next section, we show that we do get it for the special case of queries in normal form.

3.5 NORMAL FORM

This is the normal form we will be using:

Definition 7 A set S of closed formulas is logically *separable* iff for every consistent set of ground literals L , if $L \cup \{\alpha\}$ is consistent for every $\alpha \in S$, then $L \cup S$ has a standard model.

Definition 8 The normal form formulas \mathcal{NF} is the least set such that

1. if α is a ground atom or ewff, then $\alpha \in \mathcal{NF}$;
2. if $\alpha \in \mathcal{NF}$, then $\neg\alpha \in \mathcal{NF}$;
3. if $S \subseteq \mathcal{NF}$, S is logically separable, and S is finite, then $\wedge S \in \mathcal{NF}$;
4. if $S \subseteq \mathcal{NF}$, S is logically separable, and for some α , $S = \{\alpha_c^x \mid c \in C\}$, then $\forall x.\alpha \in \mathcal{NF}$.

Before explaining how the definition works, we state the main theorem:

Theorem 6 Suppose KB is proper. Then the evaluation associated with KB is logically complete for any normal form formula wrt $\mathcal{E} \cup KB$.

This theorem shows that as long as the query is in normal form, we have an “efficient” deductive reasoning procedure for first-order KBs with incomplete knowledge that is guaranteed to be logically correct. In other words, we can evaluate a query to determine if it or its negation is entailed, and always get answers that are logically correct.

Moreover, we can prove that in the propositional sublanguage, the restriction to normal form is without loss of expressive power:

Theorem 7 In the propositional sublanguage, for every $\alpha \in \mathcal{L}$, there is $\alpha' \in \mathcal{NF}$ such that $\models (\alpha \equiv \alpha')$.

This is not suggest that a good general query procedure would be to first convert a formula into normal form, and then apply the evaluation procedure; such an α' could be exponentially larger than the original α . The formula $\alpha' \in \mathcal{NF}$ used in the proof of this theorem is in what is called *Blake Canonical Form* (BCF) (Blake, 1938). Using later terminology (due to Quine), it is the conjunction of the non-tautologous prime implicants of α . Note, however, that while \mathcal{NF} includes BCF, it goes beyond it, in that it is closed under negation and has formulas of arbitrary alternations of \wedge and \vee . As a very simple example, suppose that α and β are in BCF and share no atoms. Then it is easy to show that $\{\neg\alpha, \neg\beta\}$ is logically separable, and so $(\alpha \vee \beta) \in \mathcal{NF}$.

We have as yet been unable to prove or disprove that every first-order formula has an equivalent normal form variant. However, it is useful to consider some special cases guaranteed to be in normal form. For example, we have

Theorem 8 If S is proper, then $\wedge S \in \mathcal{NF}$.

Another special case is as follows:

Definition 9 Two literals are *conflict-free* iff either they have the same polarity, or they use different predicates, or they use different constants at some argument position.

Theorem 9 If all the literals in α are conflict-free, then $\alpha \in NF$.⁷

Roughly speaking, this means that if we have a query where nothing can be inferred using the query alone (because none of its literals conflict), then we can use the evaluation procedure. As a further special case, if we have a query where every predicate letter appears only positively or only negatively, we are guaranteed to be in normal form, and so to get logically correct answers.

4 A SECOND APPROACH

The second approach to open-world reasoning which will allow us to use the evaluation procedure of Section 2 requires sensing, *i.e.*, getting knowledge from outside the system, to fill in details about otherwise unknown atoms. This approach is most meaningful in a context where we are reasoning about actions and their effects.

4.1 PROJECTION

One of the most fundamental tasks concerned with reasoning about actions is the *projection task*: determining whether a fluent⁸ does or does not hold after performing a sequence of actions. In the usual formulation, we are given a characterization of the initial state of the world and some sort of specification of what each action does. The projection task requires us to determine the cumulative effects (and non-effects) of sequences of actions.

Projection is clearly a prerequisite to *planning*: we cannot figure out if a given goal is achieved by a sequence of actions if we cannot determine what holds after doing the sequence. Similarly, the *high-level program execution task* (Levesque et al., 1997), which is that of finding a sequence of actions constituting a legal execution of a high-level program, also requires projection: to execute a program like “while there is a block on the table, pick up a block and put it away,” one needs to be able to determine after various sequences of actions if there is still a block on the table. For these reasons being able to solve the projection problem efficiently is a clear desiderata.

Reiter (Reiter, 1991) proposed action theories of a very special form in the language of the *situation calculus* (McCarthy and Hayes, 1969). Such theories, called *basic action theories*, have a notable characteristic that allows us to base projection on a special form of evaluation (*regression*) plus inference about the initial situation. This allows for a very efficient way of reasoning when we have complete information about the initial situation. Reiter’s basic action theories are the starting point of our discussion.

⁷A literal appears in α if the corresponding atom appears within the scope of an appropriate number (odd or even) of negation operators.

⁸By a fluent, we mean a property of the world that changes as the result of performing actions.

4.2 BASIC ACTION THEORIES

The basic action theories account of action and change is formulated in the language of the situation calculus (McCarthy and Hayes, 1969; Reiter, 2000). We will not go over the language here except to note the following components: there is a special constant S_0 used to denote the *initial situation*, namely the one in which no actions have yet occurred; there is a distinguished binary function symbol do where $do(a, s)$ denotes the successor situation to s resulting from performing action a ; relations whose truth values vary from situation to situation, are called (relational) *fluents*, and are denoted by predicate symbols taking a situation term as their last argument; and there is a special predicate $Poss(a, s)$ used to state that action a is executable in situation s .

Within this language, we can formulate action theories that describe how the world changes as the result of the available actions. In particular, basic action theories have the following form (Reiter, 1991):

- Some foundational, domain independent axioms.
- Unique names axioms for the primitive actions.
- Axioms describing the initial situation S_0 .
- Action precondition axioms, one for each primitive action a , characterizing $Poss(a, s)$.
- Successor state axioms, one for each fluent F , of the following form:⁹

$$F(\bar{x}, do(a, s)) \equiv \gamma(\bar{x}, a, s)$$

which state under what conditions $F(\bar{x}, do(a, s))$ holds as a function of what holds in situation s . These take the place of the so-called effect axioms, but also provide a solution to the frame problem (Reiter, 1991).

We will focus mainly on successor state axioms in the following.

Example 1 For example, the successor state axiom:

$$\begin{aligned} Broken(x, do(a, s)) \equiv \\ a = drop(x) \wedge Fragile(x) \\ \vee \exists b [a = explode(b) \wedge Bomb(b) \wedge Near(x, b, s)] \\ \vee a \neq repair(x) \wedge Broken(x, s) \end{aligned}$$

states that an object x is broken after doing action a if a is dropping it and x is fragile, a is exploding a bomb near it, or it was already broken, and a is not the action of repairing it.

In this setting the projection problem amounts to checking if

$$\Sigma \models \phi(do(\tilde{A}, S_0))$$

⁹Here and below, formulas should be read as universally quantified from the outside.

```

Formula regression (Formula  $\phi$ , Situation  $S$ )
{
  while ( $S \neq S_0$ ) {
    assume  $S$  is  $do(\vec{A}, S')$ ;
    for each  $F(\vec{t}, s)$  in  $\phi$ , simultaneously do {
      assume the SSA for  $F$  is  $F(\vec{z}, do(a, s)) \equiv \gamma(\vec{z}, a, s)$ ;
      replace  $F(\vec{t}, s)$  by  $\gamma(\vec{t}, A, s)$ ;
    }
    set  $S = S'$ ;
  }
  return  $\phi$ ;
}

```

Figure 3.1 Regression procedure for basic action theories

where Σ is the basic action theory describing the domain of interest, \vec{A} is a sequence of actions to perform, $do(\vec{A}, S_0)$ is the situation that results from performing the sequence of actions \vec{A} starting in the initial situation S_0 , and ϕ is a formula with a single situation term, a free variable ranging over situations. If the logical implication holds then we know that ϕ holds after performing \vec{A} starting from S_0 .

The special form of the successor state axioms allows us to regress fluents in the sense that whether or not they hold after performing an action can be determined by considering the action in question and what was true just before. By applying regression steps several times, we can regress each fluent in a formula ϕ all the way back to the initial situation. Intuitively we just have to use the regression procedure sketched in Figure 3.1 (see (Reiter, 1991) for the formal definition of regression). Observe that in the procedure, we use the pseudo-instruction **assume** S is $do(A, S')$; to make explicit the form of S , similarly for the SSA. Observe also that we do not instantiate the situation argument in Φ . It is the variable S that keeps track of the current situation. The formula returned is then to be evaluated in the initial situation, by substituting S_0 as the situation argument. For a formal definition of regression see (Reiter, 1991).

Note that using regression we are able to reduce a projection problem efficiently to an inference to be done in the initial situation. Now if we have *complete information* about the initial situation, then we just have to *evaluate* the formula obtained (using a variant of the procedure in Section 2) instead of making use of full logical inference. In other words, by using regression and making a closed-world assumption about the initial situation we get an efficient evaluation procedure for the entire projection task.

Of course, without this closed-world assumption, we cannot use evaluation (unless we restrict queries as we did in Section 3). In addition, basic action theories, by adopting this form of successor state axioms, also require a strong completeness assumption: after specifying the (perhaps conditional) effects of the given actions on fluents, and then allowing for possible ramifications of these

actions (e.g., (Lin and Reiter, 1994)), it is then assumed that a fluent changes *only if* it has been affected in one of these ways. What is not allowed, in other words, are cases where the value of a fluent does not depend only on the previous situation. This can arise in at least two ways. First, a fluent might change as the result of an action that is exogenous to the system. If a robot opens a door in a building, then when nobody else is around, it is justified in concluding that the door remains open until the robot closes it. But in a building with other occupants, doors will be opened and closed unpredictably. Secondly, the robot might have incomplete knowledge of the fluent in question. For example, a robot normally would not be able to infer the current temperature outdoors, since this is the result of a large number of unknown events and properties.

In cases such as these, the only way we can expect a robot to be able to perform the projection task for arbitrary queries using evaluation is if it has some sensing capabilities in order to determine the current value of certain fluents in the world. In (Levesque, 1996), sensing is modeled as an action performed by a robot that returns a binary measurement. The robot then uses so-called *sensed fluent axioms* to correlate the value returned with the state of various fluents. However, in this account, no attempt is made to be precise about the exact relation between sensing and regression. Moreover, there is no possibility of saying when regression should be used, and when sensing should be used.

In (De Giacomo and Levesque, 1999) a formal specification of a changing world is proposed which generalizes Reiter's solution to the frame problem to allow conditional successor state axioms, and generalizes the treatment of sensors by Levesque and others (e.g., (Baral and Son, 1997; Golden and Weld, 1996; Poole, 1995; Weld et al., 1998)) to allow conditional sensing axioms. The specification is sufficiently general that in some cases, there is simply not enough information to perform the projection task even with sensing. However, in many cases, it allows for solving projection efficiently, by using an evaluation procedure that combines sensing and regression. In the following, we analyze such a proposal in greater detail.

4.3 GUARDED ACTION THEORIES

We assume that a robot has a number of onboard sensors that provide sensing readings at any time. Formally, we introduce a finite number of *sensing functions*, which are unary functions whose only argument is a situation. For example, $\text{thermometer}(s)$, $\text{sonar}(s)$, $\text{depthGauge}(s)$, might all be real-valued sensing functions.¹⁰

We then define a *sensor-fluent formula* to be a formula of the language (without *Poss*, for simplicity) that uses at most one situation term, which is a variable, and that this term only appears as the final argument of a fluent or sensor function. We write $\phi(\vec{x}, s)$ when ϕ is a sensor-fluent formula with free variables among the \vec{x} and s , and $\phi(\vec{t}, t_s)$ for the formula that results after the

¹⁰Syntactically, these look like functional fluents, so to avoid confusion, we only deal with relational fluents in this paper.

substitution of \vec{x} by the vector of terms \vec{t} and s by the situation term t_s . A *fluent formula* is one that mentions no sensor functions. A *sensor formula* is a sensor-fluent formula that mentions sensor function, but does not mention fluents, and is assumed to be easily evaluable given the values of the sensors.

A guarded action theory is like a basic action theory except that for each fluent, instead of a single successor state axiom, it contains any number of *guarded successor state axioms* and *guarded sensed fluent axioms*.

- A *guarded successor state axiom* (GSSA) is a formula of the form

$$\alpha(\vec{x}, a, s) \supset [F(\vec{x}, do(a, s)) \equiv \gamma(\vec{x}, a, s)]$$

where α is a sensor-fluent formula called the *guard* of the axiom, F is a relational fluent, and γ is a fluent formula.

- A *guarded sensed fluent axiom* (GSFA) is a formula of the form

$$\alpha(\vec{x}, s) \supset [F(\vec{x}, s) \equiv \rho(\vec{x}, s)]$$

where α is a sensor-fluent formula called the *guard* of the axiom, F is a relational fluent, and ρ is a sensor formula.

The following examples show what a guarded action theories can express.

Example 2 The outdoor temperature is unpredictable from state to state. However, when the robot is outdoors, its onboard thermometer measures that temperature.

$$Outdoors(s) \supset$$

$$OutdoorTemp(n, s) \equiv thermometer(s) = n$$

Note that when the guard is false, i.e., when the robot is indoors, nothing can be concluded regarding the outdoor temperature.

Example 3 The indoor temperature is constant when the climate control is active, and otherwise unpredictable. However, when the robot is indoors, its onboard thermometer measures that temperature:

$$Indoors(s) \supset$$

$$IndoorTemp(n, s) \equiv thermometer(s) = n$$

$$ClimateControl(s) \supset$$

$$IndoorTemp(n, do(a, s)) \equiv IndoorTemp(n, s)$$

Note that in this case, if the climate control remains active, then a robot that goes first indoors and then outdoors will still be able to infer the current indoor temperature using both sensing and regressing. To our knowledge, no other representation for reasoning about actions can accommodate this combination.

Example 4 If the robot is alone in the building, the state of the door is completely determined by the robot's *open* and *close* actions. Either way, any time

the robot is in front of the door, its onboard door sensor correctly determines the state of the door.

$$\begin{aligned} \text{Alone}(s) &\supset \\ \text{DoorOpen}(x, do(a, s)) &\equiv \\ a = \text{open}(x) \\ \vee a \neq \text{close}(x) \wedge \text{DoorOpen}(x, s) \\ \text{InFrontOf}(x, s) &\supset \\ \text{DoorOpen}(x, s) &\equiv \text{doorSensor}(s) = 1 \end{aligned}$$

One intriguing possibility offered by this example is that on closing a door, and later coming back in front of the door to find it open, a security guard robot would be able to infer that $\neg \text{Alone}$.

Observe that guarded action theories are indeed an extension of basic action theories. We can handle a universally applicable successor state axiom like the one for *Broken* above by using the guard *TRUE*. Similarly, we can handle the case where nothing is known either about how to regress a fluent or how to sense its value (or both) by dropping GSSAs and GSFAs for the fluent all together.

Histories and the projection task. Once sensors are introduced, to determine if a fluent holds at some point, it is no longer sufficient to know the actions that have occurred; we also need to know the readings of the sensors along the way (*i.e.*, initially, and after each action). Consequently, we define a *history* as a sequence of the form $(\vec{\mu}_0) \cdot (A_1, \vec{\mu}_1) \cdots (A_n, \vec{\mu}_n)$ where A_i ($1 \leq i \leq n$) is a ground action term and $\vec{\mu}_i = (\mu_{i1}, \dots, \mu_{im})$ ($0 \leq i \leq n$) is a vector of values, with μ_{ij} understood as the reading of the j -th sensor after the i -th action. If λ is such a history, we then recursively define a ground situation term $\text{end}[\lambda]$ by $\text{end}[(\vec{\mu}_0)] = S_0$ and $\text{end}[\lambda \cdot (A, \vec{\mu})] = do(A, t)$ where $t = \text{end}[\lambda]$. We also define a ground sensor formula $\text{Sensed}[\lambda]$ as $\bigwedge_{i=0}^n \bigwedge_{j=1}^m h_j(\text{end}[\lambda_i]) = \mu_{ij}$ where λ_i is the subhistory up to action i , $(\vec{\mu}_0) \cdots (A_i, \vec{\mu}_i)$, and h_j is the j -th sensor function. So $\text{end}[\lambda]$ is the situation that results from doing the actions in λ and $\text{Sensed}[\lambda]$ is the formula that states that the sensors had the values specified by λ .¹¹

The projection task, becomes as follows: given an action theory Σ as above, a history λ , and a formula $\phi(s)$, where s is the situation argument, determine whether or not

$$\Sigma \cup \text{Sensed}[\lambda] \models \phi(\text{end}[\lambda]).$$

Example 5 As an example, assume we have a robot with a single sensor that measures the temperature. One possible history then is as follows:

¹¹ Obviously interesting histories λ have to satisfy certain legality criteria such as consistency of $\Sigma \cup \text{Sensed}[\lambda]$ and conformance to *Poss*.

$$\begin{aligned}\lambda = & \ (26^\circ) \cdot \\& (goIndoors, 20^\circ) \cdot \\& (turnOnClimateControl, 19^\circ) \cdot \\& (getGardenShears, 19^\circ) \cdot \\& (goOutdoors, 27^\circ) \cdot \\& (trimHedge, 28^\circ)\end{aligned}$$

This history tells us the robot initially sensed temperature 26° , then it went indoors and sensed 20° , then it turned the climate control on and sensed 19° , then it took the garden shears, still sensing 19° , then it went outdoors and started doing some gardening.

Let Σ be a guarded action theory for the robot that implies that all actions have the expected effect and moreover, that includes the GSSAs and GSFAs of Examples 2 and 3. Then we can infer the following projection:

$$\Sigma \cup Sensed[\lambda] \models IndoorTemp(19^\circ, end[\lambda]).$$

That is, although the robot is outdoors and hence cannot sense the temperature, it can infer that the temperature indoors is still 19° , since at one point in the history it was indoors and turned on the climate control when the temperature it was sensing was 19° , and the climate control remains on, keeping the indoor temperature constant.

4.4 GENERALIZED REGRESSION

In principle, the projection task as formulated can be solved using a general first-order theorem-prover. But our goal here is to keep the logical framework, but show that in common cases projection can be reduced using some form of regression plus inference about the initial situation, as done for basic action theories. In (De Giacomo and Levesque, 1999) a generalized form of regression that is a sensible compromise between syntactic transformations and logical inference is proposed. Specifically, logical inference is required only in evaluating the *guards* to decide which GSFAs and GSSAs to apply. This implies that the regression technique proposed is effective in cases where the guards are easily evaluable.

One can get an intuitive idea of how generalized regression works by looking at the nondeterministic procedure sketched in Figure 3.2, where Σ is a guarded action theory, λ is a history, ϕ is a sensor-fluent formula, and the notation $\phi \setminus \lambda$ stands for formula that results from replacing every sensor function $h_j(s)$ in ϕ by the j -th component of the final sensor reading in λ . Observe that the procedure never instantiates the situation argument. It is the history λ that keeps track of current situation (*i.e.*, $end[\lambda]$). For a more formal definition of generalized regression, see (De Giacomo and Levesque, 1999).

Generalized regression is always sound, so to perform the projection task, it is sufficient to regress the formula and check whether the regressed formula holds in the initial situation. Unfortunately, regression in general cannot be

```

Formula generalized-regression (Formula  $\phi$ , History  $\lambda$ )
{
  repeat {
    for each  $F(\vec{t}, s)$  in  $\phi$  {
      nondeterministically choose a GSFA
       $\alpha(\vec{z}, s) \supset [F(\vec{z}, s) \equiv \rho(\vec{z}, s)]$ 
      such that  $\Sigma \cup Sensed[\lambda] \models \forall \alpha(\vec{t}, end[\lambda]):$ 
      replace  $F(\vec{t}, s)$  by  $\rho(\vec{t}, s) \setminus \lambda$ ;
    }
    if  $\lambda = \lambda' \cdot (A, \vec{\mu})$  then {
      for each  $F(\vec{t}, s)$  in  $\phi$ , simultaneously do {
        nondeterministically choose a GSSA
         $\alpha(\vec{z}, a, s) \supset [F(\vec{z}, do(a, s)) \equiv \gamma(\vec{z}, a, s)]$ 
        such that  $\Sigma \cup Sensed[\lambda'] \models \forall \alpha(\vec{t}, A, end[\lambda']):$ 
        replace  $F(\vec{t}, s)$  by  $\gamma(\vec{t}, A, s)$ ;
      }
    }
    set  $\lambda = \lambda'$ ;
  } until (no  $F(\vec{t}, s)$  in  $\phi$ ) or ( $\lambda = (\vec{\mu}_0)$ );
  return  $\phi$ ;
}

```

Figure 3.2 Regression procedure for generalized action theories

complete. To see why, suppose nothing is known about fluent F ; then a formula like $(F(s) \vee \neg F(s))$ will not regress even though it will be entailed by any history.

The other drawback of generalized regression is that we need to evaluate guards. However, evaluating a guard is just a sub-projection task, and so for certain “well structured” action theories, in which guards of the GSFA for a fluent F do not depend circularly on F itself, we can again apply regression. Such theories are called *acyclic generalized action theories*.

4.5 JIT-HISTORIES

As noted above, we cannot expect to use generalized regression to evaluate sensor-fluent formulas in general: a tautology might be entailed even though nothing is entailed about the component fluents. However, in a practical setting, we can imagine never asking the robot to evaluate a formula unless the history is such that it knows enough about the component fluents, using the given GSSAs and GSFA, and their component fluents. In general, we call a history *just-in-time* (JIT) for a formula, if the actions and sensing readings it contains are enough to guarantee that suitable formulas (including guards) can be evaluated at appropriate points to determine the truth value of all the fluents in the formula (see (De Giacomo and Levesque, 1999) for the formal definition).

Example 6 For example consider the history of the Example 5:

$$\begin{aligned}\lambda = & \ (26^\circ) \cdot \\& (goIndoors, 20^\circ) \cdot \\& (turnOnClimateControl, 19^\circ) \cdot \\& (getGardenShears, 19^\circ) \cdot \\& (goOutdoors, 27^\circ) \cdot \\& (trimHedge, 28^\circ)\end{aligned}$$

It is easy to see that λ is a JIT history for $IndoorTemp(19^\circ, end[\lambda])$. Indeed at the end of λ the climate control is on, so we know that the indoors temperature is as it was in the previous situation. Thus we can regress the formula until we arrive to a point in the history where the robot was indoors, where the sensor readings measured the indoor temperature and the climate control is on. Observe that we do not need to require the robot to know whether the climate control was on before then, or even whether the robot is indoors now.

Although guarded action theories are assumed to be open-world, a JIT-history provides a sort of *dynamic* closed world assumption in that it ensures that the truth value of any fluent will be known whenever it is part of a formula whose truth value we need to determine. This allows us to evaluate complex formulas as we would if we had a normal closed world assumption, again using a variant of the procedure in Section 2 In (De Giacomo and Levesque, 1999) a procedure that evaluates a formula by generalized regression exploiting the notion of JIT-histories is presented.

5 CONCLUSION

In this paper, we have shown how a simple but efficient evaluation procedure that is logically correct only for closed-world knowledge bases could nonetheless be used in certain contexts with open-world ones. In the first case, we restrict queries to be in a certain normal form which, we conjecture, is without loss of expressive power; in the second case, we restrict queries to be for JIT-histories, where enough sensing information has been acquired to determine the truth values of the fluents in the query. For further discussion and directions for future work, see (Levesque, 1998) and (De Giacomo and Levesque, 1999).

References

- Abiteboul, S. and Duschka, O. (1998). Complexity of answering queries using materialized views. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'98)*, pages 254–265.
- Baral, C. and Son, T. (1997). Approximate reasoning about actions in presence of sensing and incomplete information. In *Proc. of the 1997 Int. Logic Programming Symposium (ILPS'97)*, pages 387–401.

- Belnap, N. (1977). A useful four-valued logic. In Dunn, J. and Epstein, G., editors, *Modern uses of multiple-valued logic*, pages 8–37. Reidel Publishing Company.
- Blake, A. (1938). *Canonical expressions in Boolean algebra*. PhD thesis, University of Chicago.
- Cadoli, M. and Schaerf, M. (1992). Approximate reasoning and non-omniscient agents. In *Proc. of the 4th Conf. on Theoretical Aspects of Reasoning about Knowledge (TARK'92)*, pages 169–183. Morgan Kaufmann Publishers.
- Chandra, A. K. and Merlin, P. M. (1977). Optimal implementation of conjunctive queries in relational data bases. In *Proc. of the 9th ACM Sym. on Theory of Computing (STOC'77)*, pages 77–90.
- De Giacomo, G. and Levesque, H. (1999). Projection using regression and sensors. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 160–165.
- Dunn, M. (1976). Intuitive semantics for first-degree entailments and coupled trees. *Philosophical Studies*, 29:149–168.
- Ginsberg, M. (1988). Multivalued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316.
- Golden, K. and Weld, D. (1996). Representing sensing actions: the middle ground revisited. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 174–185.
- Grahne, G. and Mendelzon, A. (1999). Tableau techniques for querying information sources through global schemas. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT'99)*, volume 1540 of *Lecture Notes in Computer Science*, pages 332–347. Springer-Verlag.
- Hogg, T., Huberman, B., and Williams, C. (1996). Frontiers in problem solving: phase transitions and complexity. *Artificial Intelligence*, 81(1-2):1–15.
- Imielinski, T. and Lipski, W. (1984). Incomplete information in relational databases. *Journal of ACM*, 31(4):761–791.
- Kautz, H., McAllester, D., and Selman, B. (1996). Encoding plans in propositional logic. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 374–384.
- Lakemeyer, G. (1990). *Models of belief for decidable reasoning in incomplete knowledge bases*. PhD thesis, Department of Computer Science, University of Toronto.
- Levesque, H. (1984). A logic of implicit and explicit belief. In *Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI'84)*, pages 198–202.
- Levesque, H. (1986). Making believers out of computers. *Artificial Intelligence*, 30:81–108.
- Levesque, H. (1996). What is planning in the presence of sensing? In *Proc. of the 13th Nat. Conf. on Artificial Intelligence (AAAI'96)*, pages 1139–1146.
- Levesque, H. (1998). A completeness result for reasoning with incomplete first-order knowledge bases. In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'98)*, pages 14–23.
- Levy, A. Y., Mendelzon, A. O., Sagiv, Y., and Srivastava, D. (1995). Answering queries using views. *Proc. of the 14th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS'95)*, pages 95–104.

- Levesque, H., Reiter, R., Lespérance, Y., Lin, F., and Scherl, R. (1997). GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84.
- Lin, F. and Reiter, R. (1994). State constraints revisited. *Journal of Logic and Computation*, 4(5):655–678.
- McCarthy, J. (1968). Programs with common sense. In Minsky, M., editor, *Semantic Information Processing*, pages 403–418. The MIT Press.
- McCarthy, J. and Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502.
- Parkes, A. (1999). *Lifted search engines for satisfiability*. PhD thesis, Dept. of Computer and Information Science, University of Oregon.
- Patel-Schneider, P. (1985). A decidable first-order logic for knowledge representation. In *Proc. of the 9th Int. Joint Conf. on Artificial Intelligence (IJCAI'85)*, pages 455–458.
- Poole, D. (1995). Logic programming for robot control. In *Proc. of the 14th Int. Joint Conf. on Artificial Intelligence (IJCAI'95)*, pages 150–157.
- Reiter, R. (1984). Towards a logical reconstruction of relational database theory. In Brodie, M. L., Mylopoulos, J., and Schmidt, J. W., editors, *On Conceptual Modelling*. Springer-Verlag.
- Reiter, R. (1991). The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press.
- Reiter, R. (2000). *Knowledge in Action: Logical Foundation for Describing and Implementing Dynamical Systems*. Kluwer. In preparation.
- Selman, B., Levesque, H., and Mitchell, D. (1992). A new method for solving hard instances of satisfiability. In *Proc. of the 10th Nat. Conf. on Artificial Intelligence (AAAI'92)*, pages 440–446.
- Urquhart, A. (1986). Many-valued logic. In Gabbay, D. and Guenther, F., editors, *Handbook of philosophical logic*, volume III, pages 71–116. Reidel Publishing Company.
- Van Fraasen, B. (1966). Singular terms, truth-value gaps, and free logic. *Journal of philosophical logic*, 63:481–495.
- Vardi, M. (1985). Querying logical databases. In *Proc. of the 4th ACM SIGART SIGMOD Sym. on Principles of Database Systems (PODS'85)*, pages 57–65.
- Vassiliou, Y. (1980). *A formal treatment of incomplete information in database management*. PhD thesis, Department of Computer Science, University of Toronto.
- Weld, D., Anderson, C., and Smith, D. (1998). Extending graphplan to handle uncertainty and sensing actions. In *Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI'98)*, pages 897–904.

Chapter 4

DECLARATIVE PROBLEM-SOLVING USING THE DLV SYSTEM

Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer

Information Systems Department

Technische Universität

Paniglgasse 16, A-1040 Wien, Austria

eiter@kr.tuwien.ac.at faber@kr.tuwien.ac.at leone@dbai.tuwien.ac.at pfeifer@dbai.tuwien.ac.at

Abstract The need for representing indefinite information led to disjunctive deductive databases, which also fertilized work on disjunctive logic programming. Based on this paradigm, the **DLV** system has been designed and implemented as a tool for declarative knowledge representation. In this paper, we focus on the usage of **DLV** for solving problems in a declarative manner and report on experiments that we have run on a suite of benchmark problems. We discuss how problems can be solved in a natural way using a “**Guess&Check**”-paradigm where solutions are guessed and verified by parts of the program. Furthermore, we describe various front-ends that can be used for solving problems in specific applications. The experiments show that due to the ongoing implementation efforts, which include fine-tuning of the underlying algorithms, successive and significant performance improvements have been achieved during the last two years. The results indicate that **DLV** is capable of solving some complex problems quite efficiently.

Keywords: Disjunctive logic programming, disjunctive databases, nonmonotonic reasoning

1 INTRODUCTION

During the past two decades, much research has been devoted to formal theories for advanced knowledge representation, and systems which are tailored to solving specific knowledge representation (KR) problems have been implemented. Only more recently, work has been spent on implementations of systems that are applicable to a wider range of KR problems. In the area of nonmonotonic reasoning and pure (non-Prolog) logic programming, implementations of such systems have been described e.g. in (Aravindan et al., 1997; Eiter et al., 1997c; Eiter et al., 1998b; Cholewiński et al., 1996; Cholewiński et al.,

1999; Niemelä and Simons, 1996; Niemelä and Simons, 1997; Chen and Warren, 1996; Rao et al., 1997; Seipel and Thöne, 1994) and also (McCain, 1999), as described in (Lifschitz, 1999b). These systems allow problems to be represented from various application fields in declarative languages.

The development of the **DLV** system (datalog plus vel, i.e., disjunction¹) (Eiter et al., 1997c; Eiter et al., 1998b) is an ongoing effort whose goal is to provide a KR tool which can be used for declarative problem-solving in an efficient way. The system is based on disjunctive logic programming (Lobo et al., 1992; Minker, 1994) without function symbols under the consistent answer set semantics (Gelfond and Lifschitz, 1991; Przymusinski, 1991; Eiter et al., 1997b; Leone et al., 1997), and has the following important features:

Advanced knowledge modeling capabilities. **DLV** provides support for declarative problem solving in several respects:

- It is capable of dealing with extended disjunctive logic programs that allow strong negation (Gelfond and Lifschitz, 1991), which can be easily expressed through stable models of disjunctive logic programs (Przymusinski, 1991).
- It has high expressiveness in a formally precise sense (Σ_2^P), which means that problems can be uniformly solved by a fixed program over varying instances that can not be expressed by languages with lower expressiveness such as Smodels (Niemelä and Simons, 1996), whose expressiveness is limited to NP (See Remark A at the end of Section 2). Moreover, some Σ_2^P -complete problems can be expressed in an elegant way.
- It allows for declarative problem solving following a “**Guess&Check**” paradigm where a solution to a problem is guessed by one part of a program and then verified through another part of the program. This will be discussed in more detail shortly below and in Section 3.
- It offers a number of front-ends for dealing with specific AI applications.

Solid Implementation. The high expressiveness of **DLV** comes at the price of a high computational cost in the worst case, and the performance of a naïve implementation would thus be infeasible for applications. Much effort has been spent on sophisticated algorithms and techniques for improving the performance, including the implementation of

- deductive database optimization techniques, and
- nonmonotonic reasoning optimization techniques.

Note that the Σ_2^P -expressiveness of **DLV** implies that a flat intelligent backtracking algorithm where each node has polynomial complexity is not feasible,

¹ “Vel” is Latin for “or”.

and thus requires a more complicated algorithm with involved optimization techniques. The **DLV** system is the state-of-the-art implementation of answer set semantics for extended disjunctive logic programs.

Database Interfaces. The **DLV** system provides an experimental interface to a relational database (Oracle) and, by means of a special query tool (Koch, 1999), also to an object-oriented database (Objectivity), which is useful for the integration of a specific problem solver developed in **DLV** into a more complex system.

In the present paper, we focus on the usage of the **DLV** system for declarative problem solving, and we report on experiments that we have run on a suite of benchmark problems, which show that the ongoing work on implementation of **DLV** led to significant performance improvements within the last years.

After recalling the syntax and semantics of **DLV**'s core language in Section 2, we discuss in Section 3 how problems can be expressed following a “**Guess&Check**” paradigm (which is a close relative of the generate-and-test paradigm in the AI community (Winston, 1992)). This paradigm has been used and described in the context of logic programming using stable models: e.g. in (Eiter et al., 1997b; Marek and Truszczyński, 1999); implicitly also in (Buccafurri et al., 1997); in (Niemelä, 1999) Datalog rules are viewed as constraints over the Herbrand Base; in (Cadoli et al., 1999) a declarative specification language based on Datalog is described which allows for specifying guessing predicates and constraints; in (Lifschitz, 1999b) plans are guessed and their validity is checked.

Informally, a candidate for the solution of a problem is nondeterministically generated through guessing rules of the program and verified using a checking part which consists of constraints and further rules. An important point is that, different from other programming languages, the guessing and checking parts of the program are purely declarative. The major strength of **DLV** compared to other systems such as Smodels (Niemelä and Simons, 1996; Niemelä and Simons, 1997) is that owing to the expressiveness of disjunction, also checking parts which solve co-NP complete problems can be specified; combined with a nondeterministic guessing part, this leads to Σ_2^P -expressiveness. Note that the DeReS system (Cholewiński et al., 1996; Cholewiński et al., 1999) offers similar expressiveness, but does not allow to uniformly solve varying instances of the same problem through a fixed program in a declarative language, as it lacks support for variables. Also Smodels' support for variables is limited in the sense that all variables must occur in at least one positive body literal whose predicate is non-recursively defined, which makes many problem encodings less straightforward and more cumbersome, and causes some inefficiencies in the evaluation of recursive programs. An example of this are programs computing transitive closures of input relations. The efficient support for programs with variables is one of the strong points of **DLV**, and makes **DLV** more amenable than the above mentioned systems for deductive database applications (see also Section 7).

In Section 4, we briefly describe some front-ends which we have developed for various applications. In particular, we review front-ends for major non-

monotonic reasoning modes, model-based diagnosis, SQL3, and inheritance. Our experience with developing these front-ends shows that **DLV** is a suitable declarative programming engine on top of which other declarative formalisms for knowledge representation having high expressiveness can be implemented.

After describing the user interface of **DLV** in Section 5, and providing an overview of the implementation of **DLV** in Section 6, we report in Section 7, the improvement of the system performance during the last two years, and we briefly address ongoing and future work, as well as the current dissemination of **DLV**.

2 CORE LANGUAGE

In this section, we provide a formal definition of the syntax and semantics of the kernel language of **DLV**: disjunctive Datalog extended with strong negation. For further background, see (Lobo et al., 1992; Eiter et al., 1997b; Gelfond and Lifschitz, 1991).

2.1 SYNTAX

Strings starting with uppercase letters denote variables, while those starting with lower case letters denote constants. A *term* is either a variable or a constant. An *atom* is an expression $p(t_1, \dots, t_n)$, where p is a *predicate* of arity n and t_1, \dots, t_n are terms. A *classical literal* l is either an atom p (in this case, it is *positive*), or a negated atom $\neg p$ (in this case, it is *negative*). A *negation as failure (NAF) literal* ℓ is of the form l or *not* l , where l is a classical literal; in the former case, it is *positive*, and in the latter case *negative*. Unless stated otherwise, by *literal* we mean a classical literal.

Given a classical literal l , its *complementary literal* is defined as $\neg p$ if $l = p$ and p if $l = \neg p$. A set L of literals is said to be *consistent* if for every literal $l \in L$, its complementary literal is not contained in L .

In addition to literals as defined above, **DLV** also supports built-ins like equality, less-than, and greater-than, as well as various arithmetic predicates like addition or multiplication. For details, we refer to (Faber and Pfeifer, 1996).

A *disjunctive rule* (*rule*, for short) r is a formula

$$a_1 \vee \dots \vee a_n :- b_1, \dots, b_k, \text{not } b_{k+1}, \dots, \text{not } b_m.$$

where $a_1, \dots, a_n, b_1, \dots, b_m$ are classical literals and $n \geq 0, m \geq k \geq 0$. The disjunction $a_1 \vee \dots \vee a_n$ is the *head* of r , while the conjunction $b_1, \dots, b_k, \text{not } b_{k+1}, \dots, \text{not } b_m$ is the *body* of r . A rule without head literals (i.e. $n = 0$) is usually referred to as an *integrity constraint*. If the body is empty (i.e. $k = m = 0$), we usually omit the “ $:-$ ” sign.

We denote by $H(r) = \{a_1, \dots, a_n\}$ the set of literals in the head, and by $B(r) = B^+(r) \cup B^-(r)$ the set of the body literals, where $B^+(r) = \{b_1, \dots, b_k\}$ and $B^-(r) = \{b_{k+1}, \dots, b_m\}$ are the sets of positive and negative body literals, respectively.

A disjunctive Datalog program \mathcal{P} (for short, simply *program*) is a finite set of rules. A **not-free** program \mathcal{P} (i.e. $\forall r \in \mathcal{P} : B^-(r) = \emptyset$) is called *positive*, and a **v-free** program \mathcal{P} (i.e. $\forall r \in \mathcal{P} : |H(r)| \leq 1$) is called *normal*.

As usual, a term (an atom, a rule,...) is called *ground*, if no variables appear in it. A ground program is also called a *propositional* program.

2.2 SEMANTICS

In this section we describe the semantics of consistent answer sets, originally defined in (Gelfond and Lifschitz, 1991).

Herbrand Universe. Given a program \mathcal{P} , let $U_{\mathcal{P}}$ (the Herbrand Universe) be the set of all constants appearing in \mathcal{P} . In case no constant appears in \mathcal{P} , an arbitrary constant χ is added to $U_{\mathcal{P}}$.

Herbrand Literal Base. Given a program \mathcal{P} , let $B_{\mathcal{P}}$ be the set of all ground atoms constructible from the predicate symbols appearing in \mathcal{P} and the constants of $U_{\mathcal{P}}$, possibly preceded by \neg .

Ground Instantiation. Given a rule r , $Ground(r)$ denotes the set of rules obtained by applying all possible substitutions σ from the variables in r to elements of $U_{\mathcal{P}}$. In a similar way, given a program \mathcal{P} , $Ground(\mathcal{P})$ denotes the set $\bigcup_{r \in \mathcal{P}} Ground(r)$. For propositional programs, $\mathcal{P} = Ground(\mathcal{P})$ holds.

Answer Sets. For every program \mathcal{P} , we define its answer sets using its ground instantiation $Ground(\mathcal{P})$ in two steps, following (Lifschitz, 1996): First we define the answer sets of positive programs, then we give a reduction of programs containing negation to positive ones and use it to define answer sets of arbitrary programs, possibly containing negation as failure.

An interpretation I is a set of classical literals. A consistent² interpretation $X \subseteq B_{\mathcal{P}}$ is called *closed under \mathcal{P}* (where \mathcal{P} is a positive program), if, for every $r \in Ground(\mathcal{P})$, $H(r) \cap X \neq \emptyset$ whenever $B(r) \subseteq X$. A set $X \subseteq B_{\mathcal{P}}$ is an *answer set* for \mathcal{P} if it is a minimal set (w.r.t. set inclusion) that is closed under \mathcal{P} .

Example 1 For the positive program $\mathcal{P}_1 = \{a \vee \neg b \vee c.\}$, the answer sets are $\{a\}$, $\{\neg b\}$, and $\{c\}$.

For the positive program $\mathcal{P}_2 = \{a \vee \neg b \vee c. ; :- a.\}$, the sets $\{\neg b\}$ and $\{c\}$ are the answer sets.

Finally, for the positive program $\mathcal{P}_3 = \{a \vee \neg b \vee c. ; :- a. ; \neg b :- c. ; c :- \neg b.\}$, the set $\{\neg b, c\}$ is the only answer set.

The *reduct* or *Gelfond-Lifschitz transform* of a ground program \mathcal{P} w.r.t. a set $X \subseteq B_{\mathcal{P}}$ is the positive ground program \mathcal{P}^X , obtained from \mathcal{P} by

²Note that we only consider *consistent answer sets*, while in (Lifschitz, 1996) also the inconsistent set of all possible literals is a valid answer set.

- deleting all rules $r \in \mathcal{P}$ for which $B^-(r) \cap X \neq \emptyset$ holds;
- deleting the negative body from the remaining rules.

An answer set of a program \mathcal{P} is a set $X \subseteq B_{\mathcal{P}}$ such that X is an answer set of $\text{Ground}(\mathcal{P})^X$.

Example 2 Given the program $\mathcal{P}_4 = \{a \vee \neg b :- c. ; \neg b :- \text{not } a, \text{ not } c. ; a \vee c :- \text{not } \neg b.\}$ and $I = \{\neg b\}$, the reduct \mathcal{P}_4^I is $\{a \vee \neg b :- c. ; \neg b.\}$. It is easy to see that I is an answer set of \mathcal{P}_4^I , and for this reason it is also an answer set of \mathcal{P}_4 .

Now consider $J = \{a\}$. The reduct \mathcal{P}_4^J is $\{a \vee \neg b :- c. ; a \vee c.\}$ and it can be easily verified that J is an answer set of \mathcal{P}_4^J , so it is also an answer set of \mathcal{P}_4 .

If, on the other hand, we take $K = \{c\}$, the reduct \mathcal{P}_4^K is equal to \mathcal{P}_4^J , but K is not an answer set of \mathcal{P}_4^K (for the rule $r: a \vee \neg b :- c$. It holds that $B(r) \subseteq K$, but $H(r) \cap K \neq \emptyset$ is violated). Indeed, it can be verified that I and J are the only answer sets of \mathcal{P}_4 .

Remark A. In some cases it is possible to emulate disjunction by non-stratified normal rules. However, this is not possible in general.

For example, consider $\mathcal{P}_5 = \{a \vee b.\}$ and $\mathcal{P}'_5 = \{a :- \text{not } b. ; b :- \text{not } a.\}$. Both programs have the same answer sets, namely $\{a\}$ and $\{b\}$.

On the other hand, $\mathcal{P}_6 = \{a \vee b. ; a :- b. ; b :- a.\}$ has $\{a, b\}$ as its single answer set, while $\mathcal{P}'_6 = \{a :- \text{not } b. ; b :- \text{not } a. ; a :- b. ; b :- a.\}$ has no answer set at all.

3 THE “GUESS&CHECK” PARADIGM

The core language of **DLV** can be used to encode problems in a highly declarative fashion, following a “**Guess&Check**” paradigm. In this section, we will illustrate how to apply this technique on a number of examples. We will see that many problems, also problems of high computational complexity (that is, even Σ_2^P -hard problems), can be solved naturally in **DLV** by using this declarative programming technique. The power of disjunctive rules allows one to express problems which are even more complex than NP uniformly over varying instances of the problem using a fixed program (i.e., a fixed program containing variables that works on any possible input).

Given a set F_I of facts that specify an instance I of some problem P , a **Guess&Check** program \mathcal{P} for P consists of the following two parts:

Guessing Part: The guessing part $G \subseteq \mathcal{P}$ of the program defines the search space, in a way such that answer sets of $G \cup F_I$ represent “solution candidates” for I .

Checking Part: The checking part $C \subseteq \mathcal{P}$ of the program tests whether a solution candidate is in fact a solution, such that the answer sets of $G \cup C \cup F_I$ represent the solutions for the problem instance I .

In general, we may allow both G and C to be arbitrary collections of rules in the program, and it may depend on the complexity of the problem which kind of rules are needed to realize these parts (in particular, the checking part); we defer this discussion to a later point in this section.

Without imposing restrictions on which rules G and C may contain, in the extremal case we might set G to the full program and let C be empty, i.e., all checking is moved to the guessing part such that solution candidates are always solutions. This is certainly not intended. However, in general the generation of the search space may be guarded by some rules, and such rules might be considered more appropriately placed in the guessing part than in the checking part. We do not pursue this issue any further here, and thus also refrain from giving a formal definition of how to separate a program into a guessing and a checking part.

For solving a number of problems, however, it is possible to design a natural **Guess&Check** program in which the two parts are clearly identifiable and have a simple structure:

- The guessing part G consists of a disjunctive rule which “guesses” a solution candidate S .
- The checking part C consists of integrity constraints which check the admissibility of S , possibly using auxiliary predicates which are defined by normal stratified³ rules.

Thus, the disjunctive rule defines the search space⁴, in which rule applications are branching points, while the integrity constraints prune illegal branches.

As an example which matches this scheme, let us consider the well-known **3-Colorability** problem.

3COL: Given a graph $G = (V, E)$ in the input, assign each node one of three colors (say, red, green, or blue) such that adjacent nodes always have different colors.

3-Colorability is a classical NP-complete problem. Assuming that the set of nodes V and the set of edges E are specified by means of predicates *node* (which is unary) and *edge* (binary), respectively, it can be encoded by the following **Guess&Check** program:

```
r: col(X,r) v col(X,g) v col(X,b) :- node(X). } Guess
c: :- edge(X,Y), col(X,C), col(Y,C). } Check
```

³For a definition of stratification, see (Apt et al., 1988).

⁴As described in Remark A at the end of Section 2, in some cases it would be possible to replace the disjunctive guessing rule by rules with unstratified negation. However, this is not possible in general. Disjunctive rules also have the advantage of being more compact and usually also more natural.

The rule r nondeterministically guesses color assignments for the nodes in the graph, and the constraint c checks that these choices are legal, i.e., that no two nodes which are connected by an edge have the same color.⁵

More precisely, let us suppose that the nodes and edges of the graph G are represented by a set F of facts with predicates *node* and *edge*. Then the (“guessing”) rule r above states that every node is colored either red or green or blue, while the (“checking”) constraint c forbids the assignment of the same color to two adjacent nodes. The answer sets of $F \cup \{r\}$ are all possible ways of coloring the graph. Note that minimality of answer sets guarantees that every node has only one color.

If an answer set of $F \cup \{r\}$ satisfies the constraint c , then it represents an admissible 3-coloring of the graph. There is in fact a one-to-one correspondence between the solutions of the 3-coloring problem and the answer sets of $F \cup \{r, c\}$. The graph is thus 3-colorable if and only if $F \cup \{r, c\}$ has some answer set, and each of the answer sets of $F \cup \{r, c\}$ represents a (different) legal 3-coloring of G .

Let us consider next another classical NP-complete problem in graph theory, namely *Hamiltonian Path*.

HPATH: Given a directed graph $G = (V, E)$ and a node a of this graph, does there exist a path of G starting at a and passing through each node in V exactly once?

Suppose that the graph G is specified by using predicates *node* (unary) and *arc* (binary), and the starting node is specified by the predicate *start* (unary). Then, the following **Guess&Check** program \mathcal{P}_{hp} solves the problem HPATH.

inPath(X, Y) v outPath(X, Y) :- arc(X, Y).	} Guess
$\left. \begin{array}{l} :- \text{inPath}(X, Y), \text{inPath}(X, Y_1), Y <> Y_1. \\ :- \text{inPath}(X, Y), \text{inPath}(X_1, Y), X <> X_1. \\ :- \text{node}(X), \text{not reached}(X). \end{array} \right\} \text{Constraints}$	
$\left. \begin{array}{l} \text{reached}(X) :- \text{start}(X). \\ \text{reached}(X) :- \text{reached}(Y), \text{inPath}(Y, X). \end{array} \right\} \text{Auxiliary Predicate}$	
} Check	

The first rule guesses a subset S of all given arcs to be in the path, while the rest of the program checks whether that subset S constitutes a Hamiltonian Path. Here, the checking part C uses an auxiliary predicate *reached*, which is defined using positive recursion.

In particular, the first two constraints in C check whether the set of arcs S selected by *inPath* meets the following requirements, which any Hamiltonian Path must satisfy: There must not be two arcs starting at the same node, and there must not be two arcs ending in the same node.

The two rules after the constraints define reachability from the starting node with respect to the set of arcs S . This is used in the third constraint, which enforces that all nodes in the graph are reached from the starting node in the

⁵ In this example, we assume that G contains no loops, i.e., edges from a node to itself. Such loops can be easily handled by adding $X <> Y$ to the constraint.

subgraph induced by S . This constraint also ensures that this subgraph is connected.

It is easy to see that any set of arcs S which satisfies all three constraints must contain the arcs of a path $p = v_0, v_1, \dots, v_k$ in G that starts at node a , and passes through distinct nodes until no further node is left, or it arrives at the starting node a again. In the latter case, this means that the path is a Hamiltonian Cycle, and by dropping the last arc, we have a Hamiltonian Path.

Thus, given a set of facts F for *node*, *arc*, and *start* which specify the problem input, the program $\mathcal{P}_{hp} \cup F$ has an answer set if and only if the input graph has a Hamiltonian Path.

We remark that if we want to compute a Hamiltonian Path rather than only answering that such a path exists, we can strip off the last arc from a Hamiltonian Cycle by adding a further constraint `: - start(Y), inPath(_,Y)` to the program. Then, the set S of selected arcs in an answer set of $\mathcal{P}_{hp} \cup F$ constitutes a Hamiltonian Path starting at a .

The problems 3COL and HPATH are popular examples of NP-complete problems. We next show that even some harder problem, which is located at the second level of the polynomial hierarchy, can be encoded by the Guess&Check technique. To this end, we consider the following problem *Strategic Companies*.

STRATCOMP: Given the collection $C = \{c_1, \dots, c_m\}$ of companies c_i owned by a holding, together with information about the products each company produces and company control, compute the set of the strategic companies in the holding.

Let us recall from (Cadoli et al., 1997) what a “strategic company” is in this context. Each company $c_i \in C$ in the holding is producing a collection G_i of goods, such that the holding produces a collection of goods $G = \bigcup_{c_i \in C} G_i$. Company control information models that a set of companies $D \subseteq C$ jointly may have control (e.g., by majority in shares) over another company $c_i \in C$. (Companies not in C , which we do not model here, might have shares in companies as well.) The company control information in STRATCOMP lists records of such control information in terms of “controlling sets” D for “controlled” companies c_i . Note that, in general, a company might have more than one controlling set, and only irredundant controlling sets (i.e., no proper subset is a controlling set) are recorded then.

Now, some companies should be sold by the holding, while the following two conditions have to be maintained: (1) After the transaction, the remaining set of companies $C' \subset C$ still allows one to produce all goods, i.e., $\bigcup_{c_i \in C'} G_i = G$. (2) No company is sold which would still be controlled by the holding after the transaction, i.e., if D is a controlling set for $c_i \in C$ and $D \subseteq C'$ holds, then also $c_i \in C'$ holds.

A set $C' \subseteq C$ is called a *strategic set*, if it is minimal with respect to inclusion, that is, it satisfies both (1) and (2) and no proper subset of C' satisfies both (1) and (2). In general, the strategic set is not unique, and multiple solutions for C' exist. A company $c_i \in C$ is called *strategic*, if it belongs to at least one of these strategic sets.

Computing the set of all strategic companies is relevant when companies should be sold, as selling any company which is not strategic for sure does not lead to a violation of any of the conditions (1) and (2). This problem is Σ_2^P -hard in general (Cadoli et al., 1997); reformulated as a decision problem ("Given a particular company c in the input, is c strategic?"), it is Σ_2^P -complete. To our knowledge, it is the only KR problem from the business domain of this complexity that has been considered so far.

We next present a program, $\mathcal{P}1$, which solves the complex problem STRAT-COMP in a surprisingly elegant way by a few rules:

$$\begin{array}{l} r : \text{strat}(Y) \vee \text{strat}(Z) :- \text{produced_by}(X, Y, Z). \\ c : \text{strat}(W) :- \text{controlled_by}(W, X, Y, Z), \\ \quad \text{strat}(X), \text{strat}(Y), \text{strat}(Z). \end{array} \left. \begin{array}{l} \text{Guess} \\ \text{Constraint} \end{array} \right\}$$

Here **strat(X)** means that X is strategic, **produced_by(X, Y, Z)** that product X is produced by companies Y and Z , and **controlled_by(W, X, Y, Z)** that W is jointly controlled by X, Y and Z . We assume that a set of facts for *company*, *controlled_by* and *produced_by* is part of the input and have adopted the setting from (Cadoli et al., 1997) where each product is produced by at most two companies and each company is jointly controlled by at most three other companies (in this case, the problem is still Σ_2^P -hard).

The guessing part G of the program consists of the disjunctive rule r , and the checking part C consists of the normal rule c . The program $\mathcal{P}1$ exploits the minimization which is inherent to the semantics of (consistent) answer sets for the check whether a candidate set C' of companies that produces all goods and obeys company control is also minimal with respect to this property.

The guessing rule r intuitively selects one of the companies c_1 and c_2 that produce some item g , which is described by **produced_by(g, c₁, c₂)**. If there were no company control information, minimality of answer sets would then naturally ensure that the answer sets of $F \cup \{r\}$ correspond to the strategic sets; no further checking would be needed. However, in case such control information (given by facts **controlled_by(c, c₁, c₂, c₃)**) is available, the rule c in the program checks that no company is sold that would be controlled by other companies in the strategic set, by simply requesting that this company must be strategic as well. The minimality of the strategic sets is automatically ensured by the minimality of answer sets.

The answer sets of $\mathcal{P}1 \cup F$ correspond one-to-one to the strategic sets of the holding. Thus, the set of all strategic companies is given by the set of all companies c for which the fact **strat(c)** is true under brave reasoning (see Section 4.1 for a description).

An important note here is that the checking constraint c interferes with the guessing rule r : applying c may "spoil" the minimal answer set generated by rule r . For example, suppose the guessing part gives rise to a ground rule

$$r' : \text{strat}(c1) \vee \text{strat}(c2) :- \text{produced_by}(g, c1, c2).$$

and the fact **produced_by(g, c₁, c₂)** is given in F . Now suppose the rule is satisfied in the guessing part by making **strat(c₁)** true. If, however, in the

checking part an instance of rule c is applied which derives $\text{strat}(c2)$, then the application of the rule r' to derive $\text{strat}(c1)$ is invalidated, as the minimality of answer sets implies that not both $\text{strat}(c1)$ and $\text{strat}(c2)$ can be derived from r' .

“Feedback” or “influence” of the checking part C on the guessing part G , in terms of literals which are derived in C and invalidate the application of rules in G or make further rules in G applicable (and thus change the guess), can be made precise in terms of a “potentially uses” relation (Eiter et al., 1994; Eiter et al., 1997b) and a “splitting set” (Lifschitz and Turner, 1994)). Such interference is in fact needed to solve STRATCOMP in the way we have outlined, if C does not contain disjunctive rules. This follows from complexity considerations: If C had no influence on G (augmented by F), then the answer sets of $G \cup C \cup F$ would just be the answers sets of $C \cup A_0$, where A_0 is an answer set of $G \cup F$. Furthermore, if C contains no disjunctive rule and G has complexity in NP (e.g., if G is, as in this case, head-cycle-free (Ben-Eliyahu and Dechter, 1994)), along with A_0 an answer set A of $C \cup A_0$ can be guessed and both A_0 and A can be verified in polynomial time. The complexity of deciding whether an atom belongs to some answer set of such a feedback-free program is thus in NP. Now we recall that a particular company is strategic iff it belongs to some strategic set, and deciding this is a Σ_2^P -complete problem. Unless $\Sigma_2^P = \text{NP}$, a hypothesis that is believed to be false by the experts, we can thus conclude that it is impossible to solve the STRATCOMP problem by a program whose answer sets correspond to the strategic sets, but in which C has no influence on G .

In general, if a program encodes a problem that is Σ_2^P -complete, then the checking part C must contain disjunctive rules unless C has influence on the guessing part G (or G has Σ_2^P -complexity). In particular, if the above program $P1$ is rewritten to eliminate such influence, then further disjunctive rules must be added. This is exemplified by the following program $P2$, which expresses the strategic sets in the generic **Guess&Check**-paradigm, where the guessing and the checking part are clearly separated:

<pre> strat(X) v ~strat(X) :- company(X). :- produced_by(X,Y,Z), not strat(Y), not strat(Z). :- controlled_by(W,X,Y,Z), not strat(W), strat(X), strat(Y), strat(Z). :- not min(X), strat(X). :- strat'(X,Y), ~strat(Y). :- strat'(X,X). min(X) v strat'(X,Y) v strat'(X,Z) :- produced_by(G,Y,Z), strat(X).. min(X) v strat'(X,C) :- controlled_by(C,W,Y,Z), strat(X), strat'(X,W), strat'(X,Y), strat'(X,Z). strat'(X,Y) :- min(X), strat(X), strat(Y), X<>Y. </pre>	<div style="display: flex; justify-content: space-between; align-items: center;"> } Guess } Check </div>
---	--

Here **company(X)** means that X is a company. The guessing rule selects a subset C' of the companies that serves as a candidate strategic set. The checking part of $P2$ is far more complicated than in the previous examples.

The first two constraints check whether all goods are still produced and whether company control is maintained with respect to the candidate set. The remaining part of the program then checks for the minimality of C' . For each company $c \in C'$, the predicate $\text{min}(c)$ must be true. Informally, this predicate means that $C' \setminus \{c\}$ does not contain a strategic set, i.e., for each subset $C'' \subseteq C' \setminus \{c\}$ either some good is no longer produced or company control is not obeyed. This is checked by using a binary predicate $\text{strat}'(c, Y)$, where the second argument describes those companies from $S \setminus \{c\}$ which are contained in C'' . The atom $\text{min}(c)$ is derived if such a violation is detected, and in this case $\text{strat}'(c, c')$ is set to true for each $c' \in C' \setminus \{c\}$ by the last rule. The remaining two constraints enforce that $\text{strat}(c, c')$ must be false for each $c' \notin C' \setminus \{c\}$.

We do not discuss the program $\mathcal{P}2$ in further detail here. It can be seen that its answer sets for a given instance correspond one-to-one to the strategic sets. Note, however, that the checking part of $\mathcal{P}2$ contains disjunctive rules. As we have discussed above, this is actually necessary given the one-way relationship of the guessing and the checking parts in $\mathcal{P}2$.

Further examples of programs in which the checking part necessarily contains disjunctive rules can be found in (Eiter et al., 1997b).

4 FRONT-ENDS

The **DLV** system consists of a kernel, which takes as input a disjunctive Datalog program as defined in Section 2, and several front-ends for various other knowledge representation and reasoning formalisms.

As of this writing, **DLV** contains front-ends for brave and cautious reasoning, model-based diagnosis (which we have described in detail in (Eiter et al., 1999)), SQL3, and a front-end extending the core language with objects and inheritance as described in (Buccafurri et al., 1999).

Each of these front-ends reads input in its language and transforms it into a disjunctive Datalog program. It then invokes the **DLV** kernel, post-processes the answer sets (if any) generated by the kernel, and possibly also controls the number of answer sets the kernel generates by indicating whether to continue or not after some answer set has been found and processed.

4.1 BRAVE AND CAUTIOUS REASONING

The brave and cautious reasoning front-ends are simple extensions of the kernel language. In addition to a disjunctive Datalog program, the user also specifies a query Q , which essentially looks like a ground body of a rule, terminated by a question mark:

$$b_1, \dots, b_k, \text{not } b_{k+1}, \dots, \text{not } b_m?$$

where each b_i ($1 \leq i \leq m$) is a ground literal.

When doing brave reasoning, **DLV** looks for an answer set M where the query Q evaluates to true. Upon success, the system replies: “ Q is bravely true, evidenced by M ”; otherwise the answer is “ Q is bravely false”.

Similarly, in the cautious case the system answers: “ Q is cautiously true” if and only if Q is true in all answer sets. In case of a negative answer, we get “ Q is cautiously false, evidenced by M ”, where M is an answer set witnessing the falsity of Q .

4.2 DIAGNOSIS

In diagnosis, given a background theory, some actual observations, and a set of hypotheses that indicate possible failures, one tries to find a subset of the hypotheses which can explain the observations by means of the theory.

DLV provides two different diagnosis front-ends: one for *abductive diagnosis* (Poole, 1989; Eiter et al., 1997a), and one for *Reiter’s consistency-based diagnosis* (Reiter, 1987), both of which currently support three different modes: general diagnosis, where all diagnoses are computed, subset minimal diagnosis⁶, and single failure diagnosis.

The diagnostic theory of the abductive diagnosis front-end obeys the syntax described in Section 2. Hypotheses (resp. observations) are lists of atoms (resp. NAF literals) separated by a dot (the ‘.’ character).

Intuitively, the front-end works as follows: After reading the input, rules for guessing possible diagnosis candidates are generated from the hypotheses, while the observations become constraints that forbid the generation of inconsistent diagnoses. In the case of subset minimal diagnosis, further rules are added for pruning non-minimal solutions. Finally, the **DLV** kernel is invoked, and for each reported answer set, the corresponding diagnosis is returned.

Example 3 (Network Diagnosis) Suppose that machine a is online in the computer network depicted in Figure 4.1, but we observe that it cannot reach machine e . Which machines are offline?

This reasoning task can be easily modeled as a diagnosis problem, where the theory is

```
reaches(X,X) :- node(X), not offline(X).
reaches(X,Z) :- reaches(X,Y), connected(Y,Z),
not offline(Z).
```

the observations are

```
not offline(a). not reaches(a,e).
```

and the hypotheses are

```
offline(a). offline(b). offline(c).
offline(d). offline(e). offline(f).
```

The front-end for consistency-based diagnosis, as described in (Reiter, 1987), is similar to the one for abductive diagnosis.

4.3 INHERITANCE

⁶For positive non-disjunctive theories only.

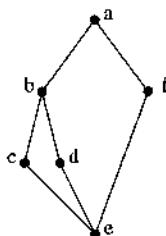


Figure 4.1 Computer network

In (Buccafurri et al., 1999) a novel language, $DLP^<$, has been defined. It extends disjunctive Datalog by grouping rules into objects and defining a partial order among these objects.

Formally, an *object* o is a pair $\langle oid(o), \Sigma(o) \rangle$, where $oid(o)$ is a member of a set of object identifiers (\mathcal{O}) and $\Sigma(o)$ is a (possibly empty) set of rules.

The program $\mathcal{P}(o)$ for an object o is the set $\{o' \mid o < o'\}$ for a given partial order $<$ on \mathcal{O} , which defines specificity among objects.

In addition to the closure condition defined in Section 2, a $DLP^<$ rule may also be overridden, which informally means that for all literals in the head of this rule, a complementary literal is supported by a more specific (w.r.t. the partial ordering $<$) rule. For details, we refer to (Buccafurri et al., 1999).

Example 4 Consider the following program $\mathcal{P}(\text{tweety})$:

```

bird      { flies. }
penguin:bird { ~flies. }
tweety:penguin { }

```

$\mathcal{P}(\text{tweety})$ consists of three objects (`bird`, `penguin`, and `tweety`). The partial order $<$ on the objects is specified in the program by the transitive reduction, which is denoted by the colon. The transitive reduction in the example is `tweety:penguin` and `penguin:bird`, so the corresponding partial order consists of `tweety < penguin`, `penguin < bird`, and `tweety < bird`. Each object is followed by its rules which are enclosed by $\{ \}$.

$\mathcal{P}(\text{tweety})$ has exactly one answer set: $\{\neg\text{flies}\}$, which fully captures the intuitive meaning of the program. The rule `flies` in object `bird` would violate the closure condition in Section 2, but this violation is allowed because the rule is overridden by `~flies` in the more specific object `penguin`.

It is important to note that $DLP^<$ directly generalizes the kernel language of **DLV** since an ordinary disjunctive Datalog program exactly corresponds to a $DLP^<$ program which involves a single object. Equally important is the fact that this language extension comes “for free” in terms of complexity (which is exactly the same as for the core language), while it serves as a more natural representation tool for reasoning with exceptions. Also frame axioms can be represented very naturally, as shown in (Buccafurri et al., 1999).

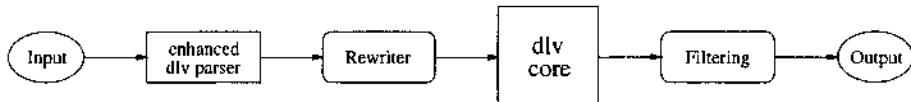


Figure 4.2 Inheritance Front-end.

In the same paper a reduction from $\text{DLP}^<$ to disjunctive Datalog under the answer set semantics is described. We have implemented this transformation as part of a complete $\text{DLP}^<$ front-end on top of **DLV**.

A schematic visualization of the architecture of this front-end is shown in Figure 4.2.

Basically, the front-end works as follows: First of all, we have extended the **DLV** parser to incorporate the $\text{DLP}^<$ syntax. In this way, all the advanced features of **DLV** (e.g. bounded integer arithmetics, comparison built-ins, etc.) are also available in $\text{DLP}^<$. The *Rewriter* module implements the translation described in (Buccafurri et al., 1999). Once the rewritten version $\pi(\mathcal{P})$ of \mathcal{P} is generated, its answer sets are then computed using the **DLV** core. Before the output is shown to the user, internal predicates are stripped from the output, which have been introduced by π .

On the web page (Faber, 1999) the prototype is described in detail, and executables are available for various platforms.

4.4 SQL3

In addition to the logic based front-ends described above, **DLV** also has an SQL3 front-end based on the current working draft of the standard. This front-end translates a subset of SQL3 query expressions to Datalog queries and supports recursive query processing such as the list-of-materials problem, where we have items that consist of other items that again consist of other items and so forth, and have to transform this tree structure into a list of all dependencies.

Since there are no column names in Datalog, we introduced the construct

```
DATALOG SCHEMA Relationname(Columnname, ...);
```

which creates a connection between the parameters of a Datalog predicate and the column names of an SQL table.

Example 5 (List of Materials) Consider the canonical list of materials query:

```

DATALOG SCHEMA consists_of(major,minor);

WITH RECURSIVE listofmaterials(major,minor) AS
(
    SELECT c.major, c.minor FROM consists_of AS c
    UNION
    SELECT c1.major, c2.minor
        FROM consists_of AS c1, listofmaterials AS c2
        WHERE c1.minor = c2.major
)

```

Table 4.1 Command-line switches

<i>Option</i>	<i>Front-end/Mode</i>
-FB	brave reasoning
-FC	cautious reasoning
-FD	generic abductive diagnosis
-FDsingle	single error abductive diagnosis
-FDMIN	subset minimal abductive diagnosis
-FR	"Reiter's" (consistency-based) diagnosis
-FRsingle	single error "Reiter's" diagnosis
-FRmin	subset minimal "Reiter's" diagnosis
-FS	SQL3

```
SELECT major, minor FROM listofmaterials;
```

This query is translated into

```
listofmaterials(A,B) :- consists_of(A,B).
listofmaterials(A,B) :- consists_of(A,C), listofmaterials(C,B).
sql2dl_intern0(A,B) :- listofmaterials(A,B).
```

and the elements of `sql2dl_intern0` are printed in tabular form as the result.

5 USAGE

While **DLV** also comes with a graphical user interface (GUI), the computational engine and its front-ends are implemented as a command-line tool, that is, the system is controlled by command-line options; it reads input from text files whose names are provided on the command-line, and any output is written to the standard output.

In its default mode, **DLV** simply computes all answer sets of the program read from the input files; specific front-ends are invoked by means of command-line options starting with `-F`.

The brave reasoning front-end is invoked by the `-FB` command-line option, while the cautious front-end is invoked by `-FC`.

For the diagnosis front-ends, hypotheses and observations have to be stored in files whose names carry the extensions `.hyp` and `.obs`, respectively, while the program is stored in files with any extension different from these.

The type of diagnostic reasoning (abductive or consistency-based) and the desired minimality criterion (none, single error, subset minimality) is selected by specifying one of the options depicted in Table 4.1 on the command-line.

The SQL front-end, finally, is selected by means of the `-FS` command-line option. Input files whose names end with `.sql` are processed with an SQL parser, all other files can be used to specify "databases" in the form of Datalog facts and non-disjunctive positive rules.

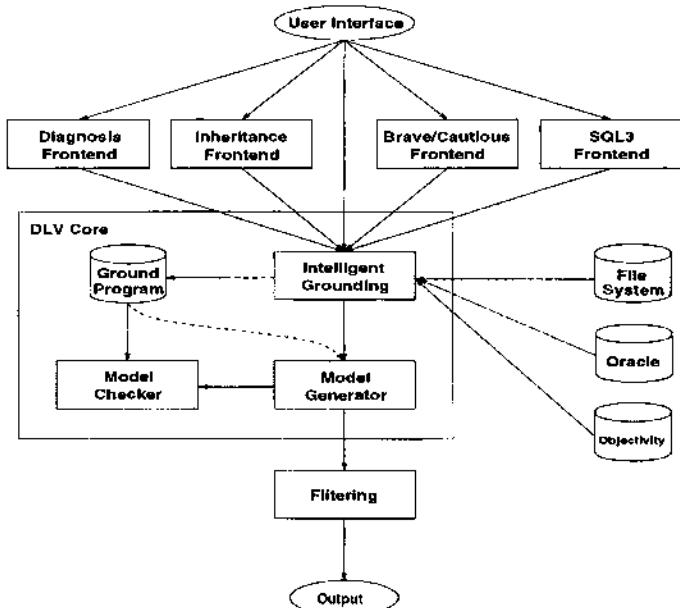


Figure 4.3 The System Architecture of DLV

6 THE DLV PROBLEM SOLVING ENGINE

Since this paper focuses mainly on knowledge representation, we cannot provide a detailed description of **DLV**'s system internal, but only give an overview. More details on the basic algorithms implemented in the system can be found in (Leone et al., 1997; Eiter et al., 1998a; Faber et al., 1999b; Koch and Leone, 1999).

The system architecture of **DLV** is shown in Figure 4.3. The internal system language is the one described in Section 2, and the **DLV** Core (the shaded part of the figure) is an efficient engine for computing all or some answer sets of its input. In addition to various front-ends (described in Section 4), there is a Graphical User Interface (GUI) that provides convenient access both to the system itself and these front-ends.

Ideally, the performance of a system reflects the complexity of the problem at hand, such that “easy” problems (say, those of polynomial complexity) are solved fast, while more time is spent for more difficult problems (like NP-hard problems), and indeed the **DLV** system is designed in this direction. For example, stratified normal programs (which have polynomial complexity) are evaluated fast using techniques from the fields of deductive databases and logic programming. The SQL3 front-end, which only generates such programs, benefits from this and has good performance. Note that in general, that is without syntactical restrictions, it is impossible to detect whether a program uniformly encodes an “easy” (even trivial) problem, since this task is clearly undecidable.

The architecture of the **DLV** Core closely reflects complexity results for various subsets of our language: As mentioned before, the Intelligent Grounding

(IG) module is able to completely solve problems of polynomial complexity (normal stratified programs are completely solved by the IG); the Model Generator (together with the Grounding) is capable of solving NP-hard problems; finally, adding the Model Checker is needed to solve Σ_2^P -hard problems.

Upon startup, the **DLV** Core or one of the front-ends parses the input specified by the user and transforms it into the internal data structures of **DLV**. In both cases, this takes only linear time and memory.

Using differential (and other advanced) database techniques together with suitable data structures, the *Intelligent Grounding* module then efficiently generates a subset of the ground instantiation of that input which has the same answer sets, but is much smaller in general. For example, in case of a stratified program, the IG module already computes the single answer set.

The heart of the computation is performed by the Model Generator and the Model Checker. Roughly, the *Model Generator* (MG) produces some “candidate” answer sets. The stability of each of them is subsequently verified by the Model Checker.

The generation of the answer sets of a program \mathcal{P} relies on a monotonic operator $\mathcal{W}_{\mathcal{P}}$ (Leone et al., 1997) which extends the well-founded operator of (van Gelder et al., 1991) for normal programs to disjunctive programs. It is defined in terms of a suitable notion of *unfounded set*. Intuitively, an unfounded set for a disjunctive program \mathcal{P} w.r.t. an interpretation I is a set of positive literals that definitely cannot be derived from \mathcal{P} assuming the facts in I (Leone et al., 1997).

Briefly, the algorithm works as follows: First, $\mathcal{W}_{\mathcal{P}}(\emptyset)$ (the fixpoint of $\mathcal{W}_{\mathcal{P}}$) is computed, which is contained in every answer set. If $\mathcal{W}_{\mathcal{P}}(\emptyset)$ is a total model, it is returned as the unique answer set. Otherwise, moving from $\mathcal{W}_{\mathcal{P}}(\emptyset)$ towards the answer sets, a literal (called *possibly-true literal* in (Leone et al., 1997)), the truth of which allows to infer new atoms, is assumed true. Clearly the choice of “good” possibly-true literals at each step (i.e., a sequence of possibly-true literals that quickly leads to an answer set) is crucial for an efficient computation, so we employ novel heuristics with extensive lookahead and also propagate knowledge about choices that lead to inconsistency. The computation of the MG proceeds by iteratively applying the disjunctive version of the Fitting operator followed by the selection of a possibly-true literal, until either a total model of $\text{Ground}(\mathcal{P})$ is reached or two contradictory literals are generated. In the latter case, backtracking is performed; in the former case, the Model Checker is invoked.

The *Model Checker* (MC) verifies whether the model at hand is an answer set. This task is very hard in general, because checking the stability of a model is well-known to be co-NP-complete (cf. (Eiter et al., 1997b)). However, recent studies (Ben-Eliyahu and Dechter, 1994; Ben-Eliyahu and Palopoli, 1994; Leone et al., 1997) showed that minimal model checking — the hardest part of answer set (or stable model) checking — can be efficiently performed for the relevant class of *head-cycle-free* (HCF) programs (Ben-Eliyahu and Dechter, 1994).

The MC satisfies the above complexity bounds. Indeed (a) it terminates in polynomial time on every HCF program; and (b) it always runs in polynomial space and single exponential time. Moreover, even on general (non-HCF) pro-

Table 4.2 DLV's Progress

Problem instance	07 '98	02 '99	04 '99	06 '99	04 '00
3COL ^a	> 1000s	26.0s	13.6s	2.0s	0.5s
Hamiltonian Path ^b	> 1000s	> 1000s	620.4s	10.6s	0.3s
Prime implicants ^c	— ^d	20.8s	12.9s	10.0s	0.8s
Strategic Companies ^d	> 1000s	171.6s	140.2s	55.0s	34.9s
Blocksword P4 ^e	> 1000s	> 1000s	> 1000s	31.4s	2.7s
Blocksword Split P4 ^f	> 1000s	> 1000s	272.3s	10.1s	1.3s
Ancestor ^g	47.9s	47.8s	48.3s	47.3s	17.9s

^afind one coloring on a random graph with 150 nodes and 350 edges^bfind one Hamiltonian Path on a random graph with 25 nodes and 120 arcs^cfind all prime implicants on a random 3CNF with 546 clauses and 127 variables^dfind all strategic sets two randomly chosen companies occur in (150 companies, 450 products)^efind one plan of length 9 involving 11 blocks, see (Erdem, 1999)^foptimized program for ^e^gcompute the ancestor relation using double recursion (196 persons and 338 parenthoods)^zcould not handle input size

grams, the MC limits the inefficient part of the computation to the modules that are not HCF (Note that it may well happen that only a very small part of the program is not HCF).

Finally, once an answer set has been found, control is returned to the front-end in use (in a sense, also the language described in Section 2 is implemented by means of a front-end, even though by a very “thin” one), which performs post-processing and possibly invokes the MG to look for further models.

7 ONGOING WORK AND CONCLUSION

As already pointed out, the high expressivity of **DLV**, which allows to express *all* KR problems with Σ_2^P -complexity, comes at the cost of a high worst case complexity, and requires sophisticated algorithms to obtain acceptable performance on average.

Improving the computational engine of **DLV** has been one of the main tasks during the last two years. The **DLV** team has been spending quite some efforts in this direction, and we have no space here to discuss the algorithms and implementation techniques that have been incorporated into the system, see (Eiter et al., 1997c; Eiter et al., 1997d; Eiter et al., 1998b; Eiter et al., 1998a; Faber et al., 1999b; Faber et al., 1999a; Koch and Leone, 1999; Leone et al., 1997). We just report in Table 4.2 the progress that has been achieved during the last two years, by comparing the performance for a few arbitrarily chosen problem instances at various stages of **DLV**'s development (The numbers there are seconds of execution time; the smaller, the better). We also refer to (Erdem, 1998), where some of **DLV**'s improvements are documented as well.

The problems 3COL, Hamiltonian Path and Strategic Companies are encoded using the programs from Section 3. The programs of the other problems are contained in (Eiter et al., 1998b; Faber et al., 1999a; Erdem, 1999). The experiments have been performed on a Pentium II with 400 MHz and 128MB memory, running FreeBSD 3.4. The way the timings were obtained is described more in detail in (Faber and Pfeifer, 1996). The improvements are quite impressive and the **DLV** system is nowadays able to solve medium-sized instances of hard problems in reasonable time.

DLV can (most likely) not compete with specialized graph algorithms, but as the results show, it can solve large instances of the problems in acceptable time. The big advantage of **DLV** is, however, that a program for solving a graph problem might be developed and run rather quickly.

The only other answer set system which is able to evaluate all benchmark programs above is **GnT** (Janhunen et al., 2000) – a disjunctive extension of Smodels.⁷ We did not carry out a detailed performance comparison of the systems yet, but, roughly, first experimental results show that: (i) **DLV** sensibly outperforms **GnT** on deductive database benchmarks like Ancestor or Reachability, (ii) on Σ_2^P -complete problems (like Strategic Companies above), **DLV** is on average faster than **GnT**, (iii) there are NP-complete problems where **GnT** is faster than **DLV** and vice versa. Comments (i) and (iii) apply also to standard Smodels, which, however, is not able to solve Σ_2^P -complete problems.⁸

It is, at this stage, not clear to which extent the rate of performance improvement in the past can be sustained. Obviously, there will be some natural limit, and increasingly more sophistication has to be put into the system for constant performance gain. To the best of our knowledge, it is unknown how the Σ_2^P problem complexity generally behaves in this respect, and no experience from thorough studies on this subject are reported in the literature.

Further improvements might be gained by refining the algorithms. An approach that seems promising in this respect is the use of refined heuristics for pruning the search space of answer set candidates. In particular, the choice of the next “possibly-true literal” (see (Faber et al., 1999b)) considered by the algorithm can be done in a number of ways. Currently, the use of heuristics is at an early stage. Our ongoing and future work includes the development and evaluation of different heuristics.

Apart from heuristics, we are also working on various other tasks. On the system side, possible refinements of the data structures that are used by the computation engine will be explored. Another task is the implementation of weak constraints (Buccafurri et al., 1997; Faber, 1998), which allow for expressing optimization problems such as the well-known traveling salesman problem in a natural way. On the application side, we are currently developing a front-end for declarative planning. Note that planning by using answer sets of logic programs has been proposed in (Subrahmanian and Zaniolo, 1995; Dimopoulos

⁷ Actually, the input programs must be slightly modified, mainly because **GnT** requires that the range of each variable is restricted by a domain predicate.

⁸ Note that these experimental results consider only standard features of Answer Set Programming, disregarding special extensions.

et al., 1997; Turner, 1997; Lifschitz, 1999a; Lifschitz, 1999b). The front-end should support the declarative specification of modest-sized planning problems under incomplete information, such as moves of a mobile robot or the actions taken by an information agent.

The **DLV** system is disseminated in academia, and presumably soon also in industry. It is widely used for educational purposes in courses on Databases and on AI, both in European and American universities. Furthermore, **DLV** is currently under evaluation at CERN, the European Laboratory for Particle Physics located near Geneva in Switzerland and France, for an advanced deductive database application that involves complex knowledge manipulations on large-sized real databases. We believe that the strengths of **DLV** — its expressivity and solid implementation — make it an attractive tool for other similar applications in practice.

Acknowledgments

This work has been supported by FWF (Austrian Science Funds) under the projects P11580-MAT and Z29-INF.

We would like to thank Georg Gottlob and all other people who participated in the project: Robert Bihlmeyer, Francesco Buccafurri, Antonella Capalbo, Simona Citrigno, Christoph Koch, Cristinel Mateis, Simona Perri, Axel Polleres, and Francesco Scarcello.

References

- Apt, K. R., Blair, H. A., and Walker, A. (1988). Towards a theory of declarative knowledge. In Minker, J., editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann Publishers, Inc., Los Altos, California.
- Aravindan, C., Dix, J., and Niemelä, I. (1997). Dislop: A research project on disjunctive logic programming. *AI Communications*, 10(3/4):151–165.
- Ben-Eliyahu, R. and Dechter, R. (1994). Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87.
- Ben-Eliyahu, R. and Palopoli, L. (1994). Reasoning with minimal models: efficient algorithms and applications. In *Proc. Fourth International Conf. on Principles of Knowledge Representation and Reasoning (KR-94)*, pages 39–50.
- Buccafurri, F., Faber, W., and Leone, N. (1999). Disjunctive logic programs with inheritance. In Schreye, D. D., editor, *Proceedings of the 16th International Conference on Logic Programming (ICLP '99)*, pages 79–93, Las Cruces, New Mexico, USA. The MIT Press.
- Buccafurri, F., Leone, N., and Rullo, P. (1997). Strong and weak constraints in disjunctive Datalog. In Dix, J., Furbach, U., and Nerode, A., editors, *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, number 1265 in Lecture Notes in AI (LNAI), pages 2–17, Dagstuhl, Germany. Springer Verlag.
- Cadoli, M., Eiter, T., and Gottlob, G. (1997). Default logic as a query language. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):448–463.

- Cadoli, M., Palopoli, L., Schaerf, A., and Vasile, D. (1999). NP-SPEC: An executable specification language for solving all problems in NP. In Gupta, G., editor, *Proceedings of the 1st International Workshop on Practical Aspects of Declarative Languages (PADL'99)*, number 1551 in Lecture Notes in Computer Science, pages 16–30. Springer.
- Chen, W. and Warren, D. S. (1996). Computation of stable models and its integration with logical query processing. *IEEE Transactions on Knowledge and Data Engineering*, 8(5):742–757.
- Cholewiński, P., Marek, V. W., Mikitiuk, A., and Truszczyński, M. (1999). Computing with default logic. *Artificial Intelligence*, 112(2–3):105–147.
- Cholewiński, P., Marek, V. W., and Truszczyński, M. (1996). Default reasoning system DeReS. In *Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR '96)*, pages 518–528. Cambridge, Massachusetts, USA. Morgan Kaufmann Publishers.
- Dimopoulos, Y., Nebel, B., and Kochler, J. (1997). Encoding planning problems in nonmonotonic logic programs. In *Proceedings of the European Conference on Planning 1997 (ECP-97)*, pages 169–181. Springer Verlag.
- Eiter, T., Faber, W., Leone, N., and Pfeifer, G. (1999). The diagnosis frontend of the DLV system. *AI Communications – The European Journal on Artificial Intelligence*, 12(1–2):99–111.
- Eiter, T., Gottlob, G., and Leone, N. (1997a). Abduction from logic programs: Semantics and complexity. *Theoretical Computer Science*, 189(1–2):129–177.
- Eiter, T., Gottlob, G., and Mannila, H. (1994). Adding disjunction to Datalog. In *Proc. of the Thirteenth ACM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems (PODS-94)*, pages 267–278. ACM Press.
- Eiter, T., Gottlob, G., and Mannila, H. (1997b). Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):315–363.
- Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. (1997c). A deductive system for nonmonotonic reasoning. In Dix, J., Furbach, U., and Nerode, A., editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*, number 1265 in Lecture Notes in AI (LNAI), pages 363–374, Berlin. Springer.
- Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. (1997d). The architecture of a disjunctive deductive database system. In Falaschi, M., Navarro, M., and Policriti, A., editors, *Proceedings Joint Conference on Declarative Programming (APPIA-GULP-PRODE '97)*, pages 141–151.
- Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. (1998a). Progress report on the disjunctive deductive database system DLV. In Andreassen, T., Christiansen, H., and Larsen, H. L., editors, *Proc. International Conf. on Flexible Query Answering Systems (FQAS'98)*, number 1495 in Lecture Notes in AI (LNAI), pages 148–163, Roskilde University, Denmark. Springer.
- Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. (1998b). The KR system DLV: Progress report, comparisons and benchmarks. In Cohn, A. G., Schubert, L., and Shapiro, S. C., editors, *Proceedings Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR '98)*, pages 406–417. Morgan Kaufmann Publishers.

- Erdem, E. (1999). Applications of logic programming to planning: Computational experiments. Unpublished draft.
<http://www.cs.utexas.edu/users/esra/papers.html>.
- Erdem, E. (since 1998). Website for applications of logic programming to planning: Computational experiments.
<http://www.cs.utexas.edu/users/esra/experiments/experiments.html>.
- Faber, W. (1998). Disjunctive Datalog with strong and weak constraints: Representational and computational issues. Master's thesis, Institut für Informationssysteme, Technische Universität Wien.
- Faber, W. (1999). DLV homepage.
 <URL:<http://www.dbaи.tuwien.ac.at/proj/dlv/inheritance/>>.
- Faber, W., Leone, N., Mateis, C., and Pfeifer, G. (1999a). Using database optimization techniques for monomonotonic reasoning. In Committee, I. O., editor, *Proceedings of the 7th International Workshop on Deductive Databases and Logic Programming (DDLP'99)*, pages 135–139. Prolog Association of Japan.
- Faber, W., Leone, N., and Pfeifer, G. (1999b). Pushing goal derivation in DLP computations. In Gelfond, M., Leone, N., and Pfeifer, G., editors, *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, number 1730 in Lecture Notes in AI (LNAI), pages 177–191, El Paso, Texas, USA. Springer Verlag.
- Faber, W. and Pfeifer, G. (since 1996). dlv homepage. <URL:<http://www.dbaи.tuwien.ac.at/proj/dlv/>>.
- Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385.
- Janhunen, T., Niemela, I., Simons, P., and You, J.-H. (2000). Partiality and disjunctions in stable model semantics. In Cohn, A. G., Giunchiglia, F., and Selman, B., editors, *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, pages 411–419, Breckenridge, Colorado, USA. Morgan Kaufmann Publishers, Inc.
- Koch, C. (1999). A simple query facility for the objectivity/db persistent object manager. Unpublished draft, available via the author's homepage <URL: <http://home.cern.ch/~chkoch/>>.
- Koch, C. and Leone, N. (1999). Stable model checking made easy. In Dean, T., editor, *The International Joint Conferences on Artificial Intelligence (IJCAI) 1999*, pages 70–75, Stockholm, Sweden. Morgan Kaufmann Publishers.
- Leone, N., Rullo, P., and Scarcello, F. (1997). Disjunctive stable models: Unfounded sets, fixpoint semantics and computation. *Information and Computation*, 135(2):69–112.
- Lifschitz, V. (1996). Foundations of logic programming. In Brewka, G., editor, *Principles of Knowledge Representation*, pages 69–127. CSLI Publications, Stanford.
- Lifschitz, V. (1999a). Action languages, answer sets and planning. In Apt, K., Marek, V. W., Truszczyński, M., and Warren, D. S., editors, *The Logic Programming Paradigm – A 25-Year Perspective*, pages 357–373. Springer Verlag.

- Lifschitz, V. (1999b). Answer set planning. In Schreye, D. D., editor, *Proceedings of the 16th International Conference on Logic Programming (ICLP '99)*, pages 23–37, Las Cruces, New Mexico, USA. The MIT Press.
- Lifschitz, V. and Turner, H. (1994). Splitting a logic program. In Van Hentenryck, P., editor, *Proceedings of the 11th International Conference on Logic Programming (ICLP'94)*, pages 23–37, Santa Margherita Ligure, Italy. MIT Press.
- Lobo, J., Minker, J., and Rajasekar, A. (1992). *Foundations of Disjunctive Logic Programming*. The MIT Press, Cambridge, Massachusetts.
- Marek, V. W. and Truszczyński, M. (1999). Stable models and an alternative logic programming paradigm. In Apt, K., Marek, V. W., Truszczyński, M., and Warren, D. S., editors, *The Logic Programming Paradigm – A 25-Year Perspective*, pages 375–398. Springer Verlag.
- McCain, N. (1999). The clausal calculator homepage. <URL:<http://www.cs.utexas.edu/users/mccain/cc/>>.
- Minker, J. (1994). Overview of disjunctive logic programming. *Annals of Mathematics and Artificial Intelligence*, 12:1–24.
- Niemelä, I. (1999). Logic programming with stable model semantics as constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):241–273.
- Niemelä, I. and Simons, P. (1996). Efficient implementation of the well-founded and stable model semantics. In Maher, M. J., editor, *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming (ICLP'96)*, pages 289–303, Bonn, Germany. MIT Press.
- Niemelä, I. and Simons, P. (1997). Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In Dix, J., Furbach, U., and Nerode, A., editors, *Proceedings of the 4th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*, volume 1265 of *Lecture Notes in AI (LNAI)*, pages 420–429, Dagstuhl, Germany. Springer Verlag.
- Poole, D. (1989). Explanation and prediction: An architecture for default and abductive reasoning. *Computational Intelligence*, 5(1):97–110.
- Przymusinski, T. C. (1991). Stable semantics for disjunctive programs. *New Generation Computing*, 9:401–424.
- Rao, P., Sagonas, K. F., Swift, T., Warren, D. S., and Freire, J. (1997). XSB: A system for efficiently computing well-founded semantics. In Dix, J., Furbach, U., and Nerode, A., editors, *Proceedings of the 4th Int. Conf. on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, number 1265 in Lecture Notes in AI (LNAI), pages 2–17, Dagstuhl, Germany. Springer Verlag.
- Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95.
- Seipel, D. and Thöne, H. (1994). DisLog – A system for reasoning in disjunctive deductive databases. In Olivé, A., editor, *Proceedings International Workshop on the Deductive Approach to Information Systems and Databases (DAISD'94)*, pages 325–343. Universitat Politecnica de Catalunya (UPC).

- Subrahmanian, V. and Zaniolo, C. (1995). Relating stable models and AI planning domains. In Sterling, L., editor, *Proceedings of the 12th International Conference on Logic Programming*, pages 233–247, Tokyo, Japan. MIT Press.
- Turner, H. (1997). Representing actions in logic programs and default theories: A situation calculus approach. *Journal of Logic Programming*, 31(1–3):245–298.
- van Gelder, A., Ross, K., and Schlipf, J. (1991). The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650.
- Winston, P. H. (1992). *Artificial Intelligence*. Addison Wesley.

IV

NONMONOTONIC REASONING

Chapter 5

THE ROLE OF DEFAULT LOGIC IN KNOWLEDGE REPRESENTATION

James P. Delgrande,
Department of Computer Science
Simon Fraser University
Burnaby, BC V5A 1S6, Canada
jim@cs.sfu.ca

Torsten Schaub
Universität Potsdam
torsten@cs.uni-potsdam.de

Abstract Various researchers in Artificial Intelligence have advocated formal logic as an analytical tool and as a formalism for the representation of knowledge. Our thesis in this paper is that commonsense reasoning frequently has a nonmonotonic aspect, either explicit or implicit, and that to this end Default Logic (DL) provides an appropriate elaboration of classical logic for the modeling of such phenomena. That is, DL is a very general, flexible, and powerful approach to nonmonotonic reasoning, and its very generality and power makes it suitable as a tool for modeling a wide variety of applications.

We propose a general methodology for using Default Logic, involving the naming of default rules and the introduction of special-purpose predicates, for detecting conditions for default rule applicability and controlling a rule's application. This allows the encoding of specific strategies and policies governing the set of default rules. Here we show that DL can be used to formalize preferences among properties and the inheritance of default properties, and so we essentially use DL to axiomatize such phenomena.

Keywords: Default logic, knowledge representation, nonmonotonic reasoning, preference handling, property inheritance

1 INTRODUCTION

First-order logic (FOL) has long and often been advocated as an appropriate tool for formalizing knowledge about particular domains and for the analysis of various systems and approaches. The role of logic in Artificial Intelligence (AI) and Knowledge Representation is discussed in, for example, (Moore, 1982; Hayes, 1985b); in addition, many introductory AI texts assume or promote the centrality of formal logic in Knowledge Representation and Reasoning. Among other domains, logic has been used to formalize knowledge about liquids (Hayes, 1985a), time (Allen and Hayes, 1989), actions and planning (McCarthy and Hayes, 1969; Levesque et al., 1998), and concepts, as well as modal notions concerning possible worlds, knowledge, belief, etc. (McCarthy, 1979; Moore, 1980). It has been used to analyze, for instance, relational database systems (Reiter, 1984) and assumption-based truth maintenance systems (Reiter and de Kleer, 1987). Default and diagnostic reasoners have been built via specifying how logic is to be used (Poole, 1988).

Our thesis in this paper is that commonsense reasoning frequently has a nonmonotonic aspect, and that to this end Default Logic is an appropriate elaboration of classical logic for the modeling of such phenomena. As we discuss in the next section, Default Logic has found wide application in AI, most obviously in the direct encoding of default information, but also in areas ranging from database theory to natural language understanding. To some extent our theme that Default Logic can be used as a general knowledge representation formalism is implicit in the quantity and breadth of approaches that make use of it. However we further argue this point by suggesting a methodology for Default Logic in which Default Logic appears more broadly applicable to various diverse phenomena than might be otherwise suspected.

The general framework is quite straightforward. We begin with a language for expressing some phenomenon in which we are interested, along with a notion of what inferences should obtain in this language. A translation is given, such that the original theory is mapped into a (standard, Reiter) default theory, wherein the desired inferences provably obtain. This translation serves two purposes. It provides a semantics for the original theory, in that it demonstrates how the original theory is expressible in Default Logic. Second, it provides a direction for implementation: given a modular translation into Default Logic and a (presumed) Default Logic theorem prover, it is straightforward to implement the desired inferences of the original theory. In one example developed here, we show how a preference ordering on properties can be mapped into a (standard, Reiter) default theory, such that one can determine the highest-ranked, consistently-assumable properties.

We do not provide a specific encoding and translation, but rather we outline a methodology by means of which one may carry out such an encoding and translation. To illustrate, consider a general default rule $\frac{\alpha : \beta}{\gamma}$ with informal meaning that if α is true and β is consistent with a set of beliefs then γ is believed. Assume that constant n is associated with this rule as its name in the object theory. Then the rule $\frac{\beta}{bl(n)}$ will be applied just when the original rule cannot be applied due to its justification being not consistent with the set of

beliefs. $\text{bl}(n)$ is a newly-introduced predicate; the concluding of $\text{bl}(n)$ signals in the object theory that the original rule is not applied. Similarly, if we replace the rule $\frac{\alpha:\beta}{\gamma}$ by $\frac{\text{ok}(n) \wedge \alpha:\beta}{\gamma}$, where $\text{ok}(\cdot)$ is a newly-introduced predicate, then application of this rule can be controlled in the object theory, in that the rule cannot be applied unless $\text{ok}(n)$ has been asserted. This notion of adding "tags" to detect and control rule application yields surprisingly powerful results. Using this technique, we have shown in (Delgrande and Schaub, 1997a) how an ordered default theory, consisting of a theory with a partial order on the default rules, can be translated into a "standard" default theory in which the rule ordering is adhered to. Hence, among other things, we show that an explicit ordering on a set of default rules adds nothing to the overall expressibility of Default Logic.

The paper is organized as follows. Following a brief introduction to Default Logic and a discussion of approaches that have employed it, we summarize the basic components of our approach. We then illustrate the applicability of the approach in two "modeling exercises". First, we review our earlier and most basic approach to dealing with preferences on rules. Second, we show how Default Logic can be used to implement a mechanism for default property inheritance. The overall theme is that by using Default Logic we can axiomatize various diverse phenomena. Hence we suggest that, in addition to directly representing nonmonotonic theories, Default Logic is appropriate as a general AI formalism in which specific phenomena may be encoded. Thus, via various examples we show how Default Logic can be employed to provide a semantics for such phenomena and, on the other hand, provide an encoding for reasoning with such phenomena.

2 DEFAULT LOGIC

Default logic (Reiter, 1980) augments classical logic by *default rules* of the form $\frac{\alpha:\beta_1,\dots,\beta_n}{\gamma}$ where $\alpha, \beta_1, \dots, \beta_n, \gamma$ are sentences of first-order or propositional logic. Here we deal with *singular* defaults for which $n = 1$. A singular rule is *normal* if β is equivalent to γ ; it is *semi-normal* if β implies γ . (Janhunen, 1999) shows that any default rule can be transformed into a set of semi-normal defaults. We sometimes denote the *prerequisite* α of a default δ by $\text{Prereq}(\delta)$, its *justification* β by $\text{Justif}(\delta)$, and its *consequent* γ by $\text{Conseq}(\delta)$. Accordingly, $\text{Prereq}(D)$ is the set of prerequisites of all default rules in D ; $\text{Justif}(D)$ and $\text{Conseq}(D)$ are defined analogously. Empty components, such as no prerequisite or even no justifications, are assumed to be tautological. *Open defaults* with unbound variables are taken to stand for all corresponding instances. A set of default rules D and a set of formulas W form a *default theory* (D, W) that may induce a single, multiple, or even zero *extensions* in the following way.

Definition 1 Let (D, W) be a default theory and let E be a set of formulas. Define $E_0 = W$ and for $i \geq 0$:

$$\begin{aligned} D_i &= \left\{ \frac{\alpha:\beta_1,\dots,\beta_n}{\gamma} \in D \mid \alpha \in E_i, \neg\beta_1 \notin E, \dots, \neg\beta_n \notin E \right\} \\ E_{i+1} &= \text{Th}(E_i) \cup \{ \text{Conseq}(\delta) \mid \delta \in D_i \} \end{aligned}$$

Then E is an extension for (D, W) iff $E = \bigcup_{i=0}^{\infty} E_i$.

Any such extension represents a possible set of beliefs about the world at hand. The above procedure is not constructive since E appears in the specification of E_{i+1} . We define $GD(D, E) = \bigcup_{i=0}^{\infty} D_i$ as the set of *generating defaults* of extension E . An enumeration $\langle \delta_i \rangle_{i \in I}$ of default rules is *grounded* in a set of formulas W , if we have for every $i \in I$ that $W \cup \text{Conseq}(\{\delta_0, \dots, \delta_{i-1}\}) \vdash \text{Prereq}(\delta_i)$.

2.1 APPLICATIONS OF DEFAULT LOGIC

The most obvious and direct use of Default Logic is in the “brute force” encoding of default information. Thus “penguins normally do not fly” may be represented by the rule $\frac{P \wedge \neg F}{\neg F}$ and “birds normally fly” may be represented by $\frac{B \wedge F}{F}$. Interactions between defaults are most directly handled by encoding explicitly what happens in various cases (Reiter and Crisculo, 1981). Thus, we replace our second default above with $\frac{B \wedge F \wedge \neg P}{F}$ so that if we know that some bird is also a penguin, only the first default can be applied.

In (Etherington and Reiter, 1983) inheritance networks are translated into default theories. Strict links are mapped to universally quantified material implications; default links are mapped to open normal defaults. Exception links have the effect of providing exceptions to a default link. Thus a default link from α to β , and with an exception link from γ to this link maps onto the rule $\frac{\alpha \wedge \beta \wedge \neg \gamma}{\beta}$, asserting that γ blocks the default inference from α to β . Consequently this approach provides a semantics for these inheritance networks. In so doing it enables the development of provably correct inference procedures, as well as proofs of the existence of extensions in certain cases.

Default logic has also found application in a wide range of diverse fields. It has been applied in natural language understanding, in speech act theory (Perault, 1987) and for deriving natural language presuppositions (Mercer, 1988), and in database systems (Cadoli et al., 1994). It has been used to extend the expressive power of terminological logics to handle default information (Baader and Hollunder, 1992; Baader and Hollunder, 1993). In (Turner, 1997), in an approach similar to that given here, a high-level language for describing the effects of actions is defined, following which a provably-correct translation into Default Logic is provided.

In (Gelfond and Lifschitz, 1990), *extended logic programs* under the *answer set semantics*, are defined; it is also shown there that extended logic programs correspond to the fragment of Default Logic in which the justifications and consequents of default rules are literals and the prerequisites are conjunctions of literals. We don’t discuss logic programming here (but see (Lifschitz, 1996) for an introduction and survey); however this correspondence has a number of important results for our purposes. First, the mapping of a logic program into a Default Logic theory, essentially provides an alternative semantics for this approach to logic programming. Secondly, one can regard results involving extended logic programs as applying to Default Logic. Thus, (Wang et al., 1999) presents a compilation of defeasible inheritance networks into (a subset of) extended logic programs and so into a fragment of Default Logic. On the

other hand, (Gelfond and Son, 1998) define a language for strict and defeasible rules that (among other things) can encode inheritance networks; they show how this language can be “implemented” using extended logic programs. Lastly, this correspondence suggests an even stronger thesis than that presented here, that extended logic programs (or the corresponding fragment of Default Logic) provides an appropriate language for encoding commonsense notions. We return to this point in the concluding section.

Finally, it should be noted that there are now comprehensive implementations of Default Logic available. DeReS, (Cholewiński et al., 1999), is a full implementation of Default Logic. XRay, (Schaub and Nicolas, 1997), provides an implementation platform for query-answering in semi-monotonic default logics. Smodels, (Niemelä and Simons, 1997), implements the stable model semantics and well-founded semantics, while DLV, (Eiter et al., 1997), is able to handle disjunctive logic programming with the answer set semantics.

3 A METHODOLOGY FOR THE USE OF DEFAULT LOGIC

In this section we outline the general approach. Our methodology involves the appropriate “deconstruction” of default rules, so that one can detect the application of default rules and control their application. Fundamentally, we show how one can detect whether a default rule is applicable or not, and how one can control the invocation of a default rule *within* a (first-order) default theory. This is accomplished, first, by associating a unique name with each default rule, so that it can be referred to within a theory. Second, special-purpose predicates are introduced for detecting conditions in a default rule, and for controlling rule invocation.

This in turn allows a fine-grained control over what default rules are applied and in what cases. By means of these named rules and special-purpose predicates, one can formalize various phenomena of interest. This is done as follows. We begin with a theory \mathcal{D} expressed in some language. In the examples presented here \mathcal{D} is a (regular) default theory but with an external order on default rules. In the first example, the static order represents a notion of preference among the defaults and, in the second, it represents a simplified version of default property inheritance. One then provides a translation of the given theory into a standard theory in Default Logic, $T(\mathcal{D})$. Assuming that things have been set up correctly, one can then show that the translated “standard” default theory $T(\mathcal{D})$ provably captures the phenomenon of interest expressed in \mathcal{D} .

3.1 THE APPROACH

We begin by extending the original language by a set of constants (as is done in (Brewka, 1994b) for example), such that there is a bijective mapping between these constants and the default rules.¹ Assume then that we have a

¹In the case of open defaults, we associate names with individual rule instances.

default theory (D, W) where each default rule $\frac{\alpha : \beta}{\gamma}$ has an associated name n . This can be written $n : \frac{\alpha : \beta}{\gamma}$; the next section describes naming in more detail. Consider though where we augment our original language with three predicate symbols $\text{ap}(\cdot)$, $\text{blp}(\cdot)$, and $\text{blj}(\cdot)$. Moreover each default rule $n : \frac{\alpha : \beta}{\gamma}$ is replaced by three rules:

$$\frac{\alpha : \beta}{\gamma \wedge \text{ap}(n)}, \quad \frac{\gamma : \neg\alpha}{\text{blp}(n)}, \quad \frac{\neg\beta :}{\text{blj}(n)} \quad (5.1)$$

Call the resulting default theory (D', W) . Any extension E of (D, W) will have a corresponding extension E' of (D', W) , where $E \subseteq E'$ and E' informally consists of E along with ground instances of the introduced predicates.

Moreover, we will have $\text{ap}(n) \in E'$ just if the default named n is one of the *generating defaults* of the extension. We will have $\text{blp}(n) \in E'$ if the default named by n failed to be applied by virtue of its prerequisite being unproven, and we will have $\text{blj}(n) \in E'$ if the default named by n failed to be applied by virtue of its justification being inconsistent with E' . We can go further (see the following section) and prove that for every name n

$$\text{ap}(n) \in E' \quad \text{iff} \quad (\text{blp}(n) \notin E' \text{ and } \text{blj}(n) \notin E').$$

This is all relatively straightforward and of limited use. However, it can be seen from (5.1) that via these introduced predicates we can detect when a rule is or is not applied.

Consider next another introduced predicate $\text{ok}(\cdot)$. If we replace a rule $n : \frac{\alpha : \beta}{\gamma}$ by

$$\frac{\alpha \wedge \text{ok}(n) : \beta}{\gamma}$$

then clearly the transformed rule can (potentially) be applied only if $\text{ok}(n)$ is asserted. More generally we can combine this with the preceding mapping and so have $\text{ok}(n)$ appear in the prerequisite of each rule in (5.1). We use a similar translation in the next section where, in axiomatizing preferences among default rules, we employ ok to “force” a given order on default rules. That is, we ensure that, with respect to the set of generating defaults, the most preferred rules are first flagged as “ok”, then the next most preferred, and so on.

Similarly, one could imagine replacing a default $n : \frac{\alpha : \beta}{\gamma}$ by

$$\frac{\alpha : \beta \wedge \neg\text{ko}(n)}{\gamma}.$$

Now the transformed rule behaves exactly as the original, except that we can “knock out” this rule by asserting $\text{ko}(n)$. This means of blocking a rule’s application has of course appeared earlier in the literature, where ko was most commonly called *Ab* (for “abnormal”).

There are obviously many other possibilities, and we don’t mean to suggest that the above constitutes a complete survey. In (Delgrande and Schaub, 1997c), for example, we map a rule $n : \frac{\alpha : \beta}{\gamma}$ onto (among other rules) $\frac{\alpha \wedge \text{ok}(n) : \beta}{\text{ap}(n)}$. Notably the consequent of the original rule, γ , is gone, replaced by $\text{ap}(n)$,

recording just the fact that the rule is applicable. We use this in (Delgrande and Schaub, 1997c) to axiomatize preferences over sets of default rules, where the idea is to apply rules en masse. For a set of rules, if all are found to be applicable, then and only then are all the consequents asserted.

This use of introduced predicates is central to our approach, and provides a means for modeling a very broad class of domains and applications. In the following sections we provide two examples, or “modeling exercises”, that illustrate this approach. In each case, we fix the meaning of the phenomenon in question by providing appropriate translations (or compilations) that can be performed automatically by appeal to this “tagging” technique. That is, we formalize preference and inheritance by essentially providing an axiomatization in (standard, Reiter) Default Logic. Being a formalization in Default Logic, we can prove that things work out as expected, that for example, preference among default rules works out correctly, that rules are applied in the specified order, and so on.

4 PREFERENCE

The notion of *preference* is pervasive in AI. In reasoning about default properties, one wants to apply defaults pertaining to a more specific class. In decision making, one may have various desiderata, not all of which can be simultaneously satisfied; in such a situation, preferences among desiderata may allow one to come to an appropriate compromise solution. In legal reasoning, laws may conflict. Conflicts may be resolved by principles such as ruling that newer laws will have priority over less recent ones, and laws of a higher authority have priority over laws of a lower authority. For a conflict among these principles one may further decide that the “authority” preference takes priority over the “recency” preference. (Boutilier, 1992; Brewka, 1994a; Baader and Hollunder, 1993) consider adding preferences in Default Logic while (McCarthy, 1986; Lifschitz, 1985; Grosof, 1991) and (Brewka, 1996; Zhang and Foo, 1997; Brewka and Eiter, 1997) do the same in circumscription and logic programming, respectively. In (Delgrande and Schaub, 1997a) we address preference in the context of Default Logic.

For adding preferences between default rules, a default theory is usually extended with an ordering on the set of default rules. In analogy to (Baader and Hollunder, 1993; Brewka, 1994a), an *ordered default theory* $(D, W, <)$ is a finite set D of default rules, a finite set W of formulas, and a strict partial order $< \subseteq D \times D$ on the default rules. That is, $<$ is a binary irreflexive and transitive relation on D . For simplicity, in the following development we assume the existence of a default $\delta_T = \frac{T \vdash T}{T} \in D$ where for every rule $\delta \in D$, we have $\delta < \delta_T$ if $\delta \neq \delta_T$. This gives us a (trivial) maximally preferred default that is always applicable.

Consider an example, where in the north of Québec the first language is French, then English, then Cree. A useful preference ordering is as follows.

$$\frac{N\text{Que : Cree}}{\text{Cree}} < \frac{Can : English}{English} < \frac{Que : French}{French}. \quad (5.2)$$

In this case we obtain the correct result for the north of Québec; also we obtain the correct result for the non-north of Québec (where the first language is French followed by English), and for the rest of Canada, where the first language is English. For a second example, one might prefer something (say, a car) that is red, then green; this might be expressed as²

$$\frac{\text{:Green}}{\text{Green}} < \frac{\text{:Red}}{\text{Red}}. \quad (5.3)$$

In (Delgrande and Schaub, 1997a) we show how an ordered default theory $(D, W, <)$ can be translated into a regular default theory (D', W') using the methodology outlined in Section 3 such that the explicit preferences in $<$ are “compiled” into D' and W' . In the next subsection we briefly review this approach.

4.1 STATIC PREFERENCES ON DEFAULTS

We begin with an ordered default theory $(D, W, <)$. The relation $\delta_1 < \delta_2$ has the informal interpretation that for $\delta_1, \delta_2 \in D$, δ_2 is to be considered for application before δ_1 . This theory is translated into a regular default theory (D', W') such that the explicit preferences in $<$ are “compiled” into D' and W' , in the following manner.

First, a unique name is associated with each default rule. This is done by extending the original language by a set of constants³ N such that there is a bijective mapping $n : D \rightarrow N$. We write n_δ instead of $n(\delta)$ (and abbreviate n_δ , by n , to ease notation). Also, for default rule δ with name n , we sometimes write $n : \delta$ to render naming explicit. To encode the fact that we deal with a finite set of distinct default rules, we adopt a unique names assumption (UNA_N) and domain closure assumption (DCA_N) with respect to N . That is, for a name set $N = \{n_1, \dots, n_m\}$, we add axioms

UNA_N : $(n_i \neq n_j)$ for all $n_i, n_j \in N$ with $i \neq j$.

DCA_N : $\forall x. \text{name}(x) \equiv (x = n_1 \vee \dots \vee x = n_m)$.

For convenience, we write $\forall x \in N. P(x)$ instead of $\forall x. \text{name}(x) \supset P(x)$.

Given $\delta_i < \delta_j$, we want to ensure that, before δ_i is applied, δ_j can be applied or found to be inapplicable. More formally, we wish to exclude the case where $\delta_i \in D_n$ and $\delta_j \in D_m$ for $n \leq m$ in Definition 1. For this purpose, we need to be able to, first, detect when a rule has been applied or when a rule is blocked, and, second, control the application of a rule based on other antecedent conditions. For a default rule $\frac{\alpha : \beta}{\gamma}$ there are two cases for it to not be applied: it may be that the antecedent is not known to be true (and so its negation is consistent), or it may be that the justification is not consistent (and so its negation is known to be true). For detecting this case, we introduce a new, special-purpose predicate

²To be sure, this is a naïve encoding; see (Brewka and Gordon, 1994) for a more realistic formalization.

³(McCarthy, 1986) effectively first suggested the naming of defaults using a set of aspect functions. Poole (1988) uses atomic propositions to name defaults.

$\text{bl}(\cdot)$. Similarly we introduce a predicate $\text{ap}(\cdot)$ to detect when a rule has been applied. To control application of a rule we introduce predicate $\text{ok}(\cdot)$. Then, a default rule $\delta = \frac{\alpha:\beta}{\gamma}$ is mapped to

$$\frac{\alpha \wedge \text{ok}(n_\delta) : \beta}{\gamma \wedge \text{ap}(n_\delta)}, \quad \frac{\text{ok}(n_\delta) : \neg\alpha}{\text{bl}(n_\delta)}, \quad \frac{\neg\beta \wedge \text{ok}(n_\delta) :}{\text{bl}(n_\delta)}. \quad (5.4)$$

These rules are sometimes abbreviated by $\delta_a, \delta_{b_1}, \delta_{b_2}$, respectively. While δ_a is more or less the image of the original rule δ , rules δ_{b_1} and δ_{b_2} capture the non-applicability of the rule.

None of the three rules in the translation can be applied unless $\text{ok}(n_\delta)$ is true. Since $\text{ok}(\cdot)$ is a new predicate symbol, it can be expressly made true in order to potentially enable the application of the three rules in the image of the translation. If $\text{ok}(n_\delta)$ is true, the first rule of the translation may potentially be applied. If a rule has been applied, then this is indicated by assertion $\text{ap}(n_\delta)$. The last two rules give conditions under which the original rule is inapplicable: either the negation of the original antecedent α is consistent (with the extension) or the justification β is known to be false; in either such case $\text{bl}(n_\delta)$ is concluded.

We can assert that default $n_j : \frac{\alpha_j:\beta_j}{\gamma_j}$ is preferred to $n_i : \frac{\alpha_i:\beta_i}{\gamma_i}$ in the object language by introducing a new predicate \prec and then asserting that $n_i \prec n_j$. However, this translation so far does nothing to control the order of rule application. Nonetheless, for $\delta_i < \delta_j$ we can now control the order of rule application: we can assert that if δ_j has been applied (and so $\text{ap}(n_j)$ is true), or known to be inapplicable (and so $\text{bl}(n_j)$ is true), then it's ok to apply δ_i . The idea is thus to *delay* the consideration of less preferred rules until the applicability question has been settled for the higher ranked rules.

We obtain the following translation, mapping ordered default theories in some language \mathcal{L} onto standard default theories in the language \mathcal{L}^+ obtained by extending \mathcal{L} by new predicates symbols $(\cdot \prec \cdot)$, $\text{ok}(\cdot)$, $\text{bl}(\cdot)$, and $\text{ap}(\cdot)$, and a set of associated default names:

Definition 2 (Delgrande and Schaub, 1997a) Given an ordered default theory $(D, W, <)$ over \mathcal{L} and its set of default names $N = \{n_\delta \mid \delta \in D\}$, define $T((D, W, <)) = (D', W')$ over \mathcal{L}^+ by

$$D' = \left\{ \frac{\alpha \wedge \text{ok}(n) : \beta}{\gamma \wedge \text{ap}(n)}, \frac{\text{ok}(n) : \neg\alpha}{\text{bl}(n)}, \frac{\neg\beta \wedge \text{ok}(n) :}{\text{bl}(n)} \mid n : \frac{\alpha:\beta}{\gamma} \in D \right\} \cup D_\prec$$

$$W' = W \cup W_\prec \cup \{\text{DCA}_N, \text{UNA}_N\}$$

where

$$D_\prec = \left\{ \frac{\neg(x \prec y)}{\neg(x \prec y)} \right\}$$

$$W_\prec = \{n_\delta \prec n_{\delta'} \mid (\delta, \delta') \in <\}$$

$$\cup \{\text{ok}(n_T)\}$$

$$\cup \{\forall x \in N. [\forall y \in N. (x \prec y) \supset (\text{bl}(y) \vee \text{ap}(y))] \supset \text{ok}(x)\}.$$

W' contains the prior world knowledge W , together with assertions for managing the priority order $<$ on defaults. The first part of W_\prec specifies that \prec is

a predicate whose positive instances mirror those of the strict partial order \prec . $\text{ok}(n_T)$ asserts that it is ok to apply the maximally preferred (trivial) default. The third formula in W_\prec controls the application of defaults: for every n_i , we derive $\text{ok}(n_i)$ whenever for every n_j with $n_i \prec n_j$, either $\text{ap}(n_j)$ or $\text{bl}(n_j)$ is true. This axiom allows us to derive $\text{ok}(n_i)$, indicating that δ_i may potentially be applied whenever we have for all δ_j with $\delta_i < \delta_j$ that δ_j has been applied or cannot be applied.

For the last formula in W_\prec to work properly we must have complete information about \prec . This is addressed by the default rule in D_\prec . With this rule we can generate the complement of W_\prec with a single open default; an explicit encoding would likely have much larger size. We also have $(n_\delta \prec n_T) \in W_\prec$ for every rule $\delta \neq \delta_T$ by the definition of ordered default theories. Since \prec is a strict partial order, W_\prec also includes the transitive closure of \prec and no reflexivities, such as $n \prec n$.

Note that the translation results in a manageable increase in the size of the default theory. For ordered theory (D, W, \prec) , the translation $T((D, W, \prec))$ is only a constant factor larger than (D, W, \prec) (assuming that we count the default in D_\prec as a single default).

As an example, consider the defaults:

$$n_1 : \frac{A_1 : B_1}{C_1}, \quad n_2 : \frac{A_2 : B_2}{C_2}, \quad n_3 : \frac{A_3 : B_3}{C_3}, \quad n_T : \frac{T : T}{T}.$$

We obtain for $i = 1, 2, 3$:

$$\frac{A_i \wedge \text{ok}(n_i) : B_i}{C_i \wedge \text{ap}(n_i)}, \quad \frac{\text{ok}(n_i) : \neg A_i}{\text{bl}(n_i)}, \quad \frac{\neg B_i \wedge \text{ok}(n_i) :}{\text{bl}(n_i)},$$

and analogously for δ_T where A_i, B_i, C_i are T . Given $\delta_1 < \delta_2 < \delta_3$, we obtain $n_1 \prec n_2, n_2 \prec n_3, n_1 \prec n_3$ along with $n_k \prec n_T$ for $k \in \{1, 2, 3\}$ as part of W_\prec . From D_\prec we get $\neg(n_i \prec n_j)$ for all remaining combinations of $i, j \in \{1, 2, 3, T\}$. It is instructive to verify that $\text{ok}(n_3)$, along with

$$\begin{aligned} (\text{ap}(n_3) \vee \text{bl}(n_3)) &\supset \text{ok}(n_2), \quad \text{and} \\ ((\text{ap}(n_2) \vee \text{bl}(n_2)) \wedge (\text{ap}(n_3) \vee \text{bl}(n_3))) &\supset \text{ok}(n_1) \end{aligned}$$

are obtained after a few iterations in Definition 1 (see below); from this we get that n_3 must be taken into account first, followed by n_2 and then n_1 .

The following theorem summarizes the major technical properties of our approach, and demonstrates that rules are applied in the desired order:

Theorem 1 (Delgrande and Schaub, 1997a) Let E be a consistent extension of $T((D, W, \prec))$ for ordered default theory (D, W, \prec) . We have for all $\delta, \delta' \in D$ that

1. $n_\delta \prec n_{\delta'} \in E$ iff $\neg(n_\delta \prec n_{\delta'}) \notin E$
2. $\text{ok}(n_\delta) \in E$
3. $\text{ap}(n_\delta) \in E$ iff $\text{bl}(n_\delta) \notin E$
4. $\text{ok}(n_\delta) \in E_i$ and $\text{Prereq}(\delta) \in E_j$ and $\neg \text{Justif}(\delta) \notin E$ implies $\text{ap}(n_\delta) \in E_{\max(i,j)+3}$

5. $\text{ok}(n_\delta) \in E_i$ and $\text{Prereq}(\delta) \notin E$ implies $\text{bl}(n_\delta) \in E_{i+1}$
6. $\text{ok}(n_\delta) \in E_i$ and $\neg \text{Justif}(\delta) \in E$ implies $\text{bl}(n_\delta) \in E_j$ for some $j > i + 1$
7. $\text{ok}(n_\delta) \notin E_{i-1}$ and $\text{ok}(n_\delta) \in E_i$ implies $\text{ap}(n_\delta) \notin E_j$ for $j < i + 2$ and $\text{bl}(n_\delta) \notin E_j$ for $j < i + 1$

Moreover, it turns out that our translation T amounts to selecting those extensions of the original default theory that are in accord with the provided ordering. This can be expressed in the following way.

Definition 3 (Delgrande and Schaub, 1999a) Let (D, W) be a default theory and let $< \subseteq D \times D$ be a strict partial order. An extension E of (D, W) is $<$ -preserving if there exists a grounded enumeration $(\delta_i)_{i \in I}$ of $GD(D, E)$ such that for all $i, j \in I$ and $\delta \in D \setminus GD(D, E)$, we have that

1. if $\delta_i < \delta_j$ then $j < i$ and
2. if $\delta_i < \delta$ then $\text{Prereq}(\delta) \notin E$ or $W \cup \text{Conseq}(\{\delta_0, \dots, \delta_{i-1}\}) \vdash \neg \text{Justif}(\delta)$.

In the first condition above, applied rules are applied in the order specified by $<$. Second, if a rule δ is not applied but a less-highly ranked rule is, then it must be the case that either the prerequisite of δ is not derivable (at all) or its justification is refuted by other, higher-ranked rules. In any case, the applicability issue must first be settled for higher-ranked default rules before it is for lower-ranked rules.

Theorem 2 (Delgrande and Schaub, 1999a) Let (D, W) be a default theory and let $< \subseteq D \times D$ be a strict partial order. Let E be a set of formulas.

E is a $<$ -preserving extension of (D, W) iff $E = E' \cap \mathcal{L}$ for some extension E' of $T((D, W, <))$.

Consequently, the notion of $<$ -preservation can be seen as providing an informal semantics for our approach.

One might expect that ordered default theories would enjoy the same properties as standard Default Logic. This indeed is the case, but with one important exception: in the instance of our approach described here, normal ordered default theories do not guarantee the existence of extensions. For example, the image of the ordered default theory (under our translation)

$$(\{n_1 : \frac{B}{B}, n_2 : \frac{B \cdot C}{C}\}, \emptyset, \{\delta_1 < \delta_2\}) \quad (5.5)$$

has no extension. Informally the problem is that our preference $\delta_1 < \delta_2$ conflicts with the normal order of rule application. If $W = \emptyset$, only δ_1 is applicable, but once it has applied, δ_2 becomes applicable. Thus we have an ordering implicit in the form of the defaults and world knowledge, but where this implicit ordering is contradicted by the assertion $\delta_1 < \delta_2$. Not surprisingly then there is no extension.

4.2 EXTENSIONS

In (Delgrande and Schaub, 1997a) we also show that standard default theories (D, W) over a language including a predicate expressing a preference over (named) default rules can similarly be translated into a default theory where no such mention of preferences is made. Thus for example, let bf be the name of default $\frac{B:F}{F}$, asserting that birds fly by default and b_{nf} be the name of default $\frac{B:\neg F}{\neg F}$. Further, let $loc(NZ)$ assert that the location is New Zealand. Then if we believed that birds normally fly, but New Zealand birds don't, we could encode this in the object theory by

$$\frac{\neg loc(NZ)}{bf \prec b_{nf}}, \quad \frac{loc(NZ)}{b_{nf} \prec bf}.$$

This extension allows the expression of preference in a particular context (as above), preferences applying by default, and preferences among preferences (as in the legal reasoning example mentioned at the start of this section).

Second, we show how a default theory, with an attendant ordering on *sets* of defaults, can similarly be translated into a standard default theory where no mention of preferences is made. In this case, we can express that in buying a car one may rank price (E) of a car model over safety features (S) over power (P), but safety features together with power is ranked over price, as follows:

$$\left\{ \frac{P}{P} \right\} < \left\{ \frac{S}{S} \right\} < \left\{ \frac{E}{E} \right\} < \left\{ \frac{P}{P}, \frac{S}{S} \right\} \quad (5.6)$$

If we were given only that not all desiderata can be satisfied (and so W contains $\neg P \vee \neg S \vee \neg E$) then intuitively we would want to apply the defaults in the set $\left\{ \frac{P}{P}, \frac{S}{S} \right\}$ and conclude that P and S can be met. On the other hand if we know that P and S are mutually exclusive (and so W contains $\neg P \vee \neg S$) then intuitively we would want to apply the defaults in the sets $\left\{ \frac{S}{S} \right\}$ and $\left\{ \frac{E}{E} \right\}$ and so conclude that S and E can be met. Again we show how this can be expressed in a standard default theory.

4.3 APPLICATION TO MODEL-BASED DIAGNOSIS

In (Reiter, 1987), Default Logic is used to provide an account of a theory of diagnosis from first principles. Roughly, in this framework one has an axiomatization of a domain, or *system description*, in which there is a distinguished set of *components*, given by a set of constants $COMPS$ that may be normal or abnormal. These components are assumed to be normal by default, expressed by the rule $\frac{\neg Ab(x)}{Ab(x)}$. For example, in the circuit domain the system description would include a description of a circuit, while the set $COMPS$ could represent specific gates. We can express that an AND gate that is not abnormal has output *on* when its inputs are *on* by:

$$(AndG(x) \wedge \neg Ab(x) \wedge value(in(1, x), on) \wedge value(in(2, x), on)) \\ \supset value(out(x), on).$$

As well there is a set of *observations* OBS , for example expressing that both inputs of AND gate a_1 are *on* but the output is not *on*. A diagnosis can be

expressed by an extension of the resulting theory. In an extension, one has complete information concerning all instances of Ab ; moreover the extension of Ab is minimal.

We can use our approach to incorporate further assumptions into a theory of diagnosis. The original approach appeals to a *principle of parsimony*, wherein a diagnosis is a conjecture that some minimal set of components are faulty. This can be strengthened by preferring a single-fault diagnosis over two-fault diagnosis, over three-fault diagnosis, etc. Suppose we have three components whose normal behavior is modeled⁴ by the rule $\frac{:\neg Ab(x)}{\neg Ab(x)}$. In our extension to preference that allows preferences among sets of defaults, we can model the strengthened principle of parsimony by

$$\left\{ \frac{:\neg Ab(c_1)}{\neg Ab(c_1)} \right\} < \left\{ \frac{:\neg Ab(c_1)}{\neg Ab(c_1)}, \frac{:\neg Ab(c_2)}{\neg Ab(c_2)} \right\} < \left\{ \frac{:\neg Ab(c_1)}{\neg Ab(c_1)}, \frac{:\neg Ab(c_2)}{\neg Ab(c_2)}, \frac{:\neg Ab(c_3)}{\neg Ab(c_3)} \right\}$$

for every $c_1, c_2, c_3 \in COMPS$. Suppose our system description entails $Ab(a) \vee (Ab(b) \wedge Ab(c))$. In standard Default Logic, we obtain two extensions, which violates the strengthened principle of parsimony. With the given preferences, however, we obtain only the single-fault extension, containing $Ab(a)$ along with $\neg Ab(b)$ and $\neg Ab(c)$.

In a second extension, we can model preferences for faults over types of components. In the extension to our approach where preferences can be expressed in the language, we can express the fact that an OR gate is expected to fail over an AND gate as follows. For $c \in COMPS$ let the name of the rule $\frac{:\neg Ab(c)}{\neg Ab(c)}$ be given by predicate $AbRule(c)$. We assert:

$$\forall x, y. (OrG(x) \wedge AndG(y)) \supset (AbRule(y) \prec AbRule(x)).$$

Clearly other elaborations can be addressed within this framework. For example, it would be an elementary extension to allow different types of faults, and then assert, say, that an AND gate that is stuck on is to be expected over an OR gate that negates its correct output.

5 INHERITANCE OF PROPERTIES

5.1 PREFERENCE AND INHERITANCE OF PROPERTIES

A common problem in Knowledge Representation is the inheritance of (default) properties. Informally, individuals may be expected to have (or inherit) properties by virtue of being instances of particular classes. Thus by default Sue will be assumed to be employed since Sue is an adult and adults are normally employed. The principle of *specificity* says (for our purposes) that properties are inherited from more specific classes in preference to less specific classes. Thus if Sue is a student also, and we know that students normally are not employed, then we now can conclude nothing about Sue's employment status. If

⁴Note that we haven't addressed the problem of preferences on open defaults. We skirt any difficulties here by assuming complete knowledge of the (circuit) domain.

we have that adult students are normally employed, then this would be applied in preference to the students-are-not-employed default.

It might seem that we could use the approach of the previous section to implement inheritance of properties, and indeed many approaches implement inheritance via a preference ordering. However an example shows that this can lead to unfortunate results. Consider defaults concerning primary means of locomotion: "animals normally walk", "birds normally fly", "penguins normally swim". This can be expressed in an ordered default theory as follows:

$$n_1 : \frac{\text{Animal : Walk}}{\text{Walk}} < n_2 : \frac{\text{Bird : Fly}}{\text{Fly}} < n_3 : \frac{\text{Penguin : Swim}}{\text{Swim}}. \quad (5.7)$$

If we learn that some thing is penguin (and so a bird and animal), then we would want to apply the highest-ranked default and, all other things being equal, conclude that it swims. However, if the penguin in question is hydrophobic, and so doesn't swim, preference tells us that we should try to apply the next default and so, again all other things being equal, conclude that it flies. This situation then is very different from our example (5.2), and moreover in this instance gives us an undesirable conclusion.

We can characterize this difference as follows. Let $(D, W, <)$ be a normal (i.e. the defaults in D are normal) ordered default theory. For preference, as described in the previous section, we want to "apply" defaults as constrained by $<$. For inheritance, we want to apply the $<$ -maximum defaults where the prerequisite is true, if possible. If a $<$ -maximum default is inapplicable, then no less specific default is considered. In the next section we make these intuitions precise and provide an axiomatization in Default Logic.

5.2 EXPRESSING INHERITANCE

For default property inheritance, the ordering on defaults reflects a relation of *specificity* among the prerequisites. For example, in the preceding, the class of penguins is strictly narrower than the class of birds. As (5.2) and (5.3) illustrate, this isn't the case for preference. Informally, for adjudicating among conflicting defaults, one determines the most specific (with respect to rule prerequisite) defaults as candidates for application. In approaches such as (Pearl, 1990; Geffner and Pearl, 1992), among many others, specificity is determined implicitly, emerging as a property of an underlying formal system. (Reiter and Crisculo, 1981; Etherington and Reiter, 1983; Delgrande and Schaub, 1997b) have addressed encoding specificity information in Default Logic. Here we briefly provide an account of how a mechanism of inheritance may be encoded.

To begin with, for simplicity, our account is incomplete. In particular we ignore the fact that, in formulating an ordered default theory, the specification of $<$ must take into account the relevant properties (i.e. consequents) of the default rules. For example, given defaults concerning flight, and a rule that birds normally have wings (viz. $\frac{\text{Bird : Fly}}{\text{Fly}}$, $\frac{\text{Bird : Wing}}{\text{Wing}}$, $\frac{\text{Penguin : } \neg\text{Fly}}{\neg\text{Fly}}$) we would assert $\frac{\text{Bird : Fly}}{\text{Fly}} < \frac{\text{Penguin : } \neg\text{Fly}}{\neg\text{Fly}}$. Obviously we would not want to assert $\frac{\text{Bird : Wing}}{\text{Wing}} < \frac{\text{Penguin : } \neg\text{Fly}}{\neg\text{Fly}}$ even though *Bird* subsumes *Penguin*. So here

we have nothing to say about *how* the information concerning $<$ is obtained.⁵ Rather, we assume that inheritance information *has been* appropriately captured in $<$, and our task is to provide a semantic account of property inheritance via an appropriate translation into Default Logic.

In the previous section, our approach to preference used special purpose predicates to detect when a rule in D was applied or blocked. A more fine-grained approach would be to distinguish the source of blockage by replacing δ_{b_1} and δ_{b_2} by

$$\frac{\text{ok}(n_\delta) : \neg\alpha}{\text{bl}_P(n_\delta)}, \quad \frac{\neg\beta \wedge \text{ok}(n_\delta) :}{\text{bl}_J(n_\delta)}.$$

That is, we replace bl by two new predicate symbols, bl_P and bl_J . Accordingly, in Definition 2 the final formula in $W_<$ would be

$$\forall x \in N. [\forall y \in N. (x < y) \supset (\text{bl}_P(y) \vee \text{bl}_J(y) \vee \text{ap}(y))] \supset \text{ok}(x).$$

Interestingly, a generalization of this axiom, namely

$$\forall x \in N. [\forall y \in N. ((x < y) \wedge (y \neq n_T)) \supset \text{bl}_P(y)] \supset \text{ok}(x), \quad (5.8)$$

allows us to specify *inheritance*. Given a chain of defaults $\delta_1 < \delta_2 < \dots < \delta_m$, we apply δ_i , if possible, where δ_i is the $<$ -maximum default such that for every default δ_j , $j = i + 1, \dots, m$, the prerequisite of δ_j is not known to be true. Otherwise no default in the chain is applicable. Technically, the formula (5.8) allows lower ranked default rules to be applied only in case higher ranked rules are blocked because their prerequisite is not derivable. Otherwise, the propagation of $\text{ok}(\cdot)$ -predicates is interrupted so that no defaults below δ_i are considered.

Definition 4 Given an ordered default theory $(D, W, <)$ over \mathcal{L} and its set of default names $N = \{n_\delta \mid \delta \in D\}$, define $\mathcal{I}((D, W, <)) = (D', W')$ over \mathcal{L}^+ by

$$\begin{aligned} D' &= \left\{ \frac{\alpha \wedge \text{ok}(n) : \beta}{\gamma}, \frac{\text{ok}(n) : \neg\alpha}{\text{bl}_P(n)} \mid n : \frac{\alpha : \beta}{\gamma} \in D \right\} \cup D_< \\ W' &= W \cup W_< \cup \{\text{DCA}_N, \text{UNA}_N\} \end{aligned}$$

where

$$\begin{aligned} D_< &= \left\{ \frac{\neg(x < y)}{\neg(x < y)} \right\} \\ W_< &= \{n_\delta < n_{\delta'} \mid (\delta, \delta') \in <\} \\ &\cup \{\text{ok}(n_T)\} \\ &\cup \{\forall x \in N. [\forall y \in N. ((x < y) \wedge (y \neq n_T)) \supset \text{bl}_P(y)] \supset \text{ok}(x)\}. \end{aligned}$$

Consider the ordered defaults theory $(D, W, <)$ where D and $<$ are as in (5.7), and where

$$W = \{ \text{Penguin} \supset \text{Bird}, \text{Bird} \supset \text{Animal} \}.$$

⁵This issue is dealt with in (Delgrande and Schaub, 1997b).

We obtain $n_1 \prec n_2$, $n_2 \prec n_3$, $n_1 \prec n_3$ along with $n_k \prec n_T$ for $k \in \{1, 2, 3\}$ as part of W_\prec . From D_\prec we get $\neg(n_i \prec n_j)$ for all remaining combinations of $i, j \in \{1, 2, 3, T\}$.

For the theory $(D, W \cup \{\text{Bird}\}, \prec)$ it is useful to verify that one extension is obtained. Initially $\text{ok}(n_3)$ is obtained (in Definition 1) as is $\text{bl}_P(n_3)$. Since we can deduce $\text{bl}_P(n_3) \supset \text{ok}(n_2)$, the rule $\frac{\text{Bird} \wedge \text{ok}(n_2); \text{Fly}}{\text{Fly}}$ can be applied. Since the rule $\frac{\text{ok}(n_2); \neg \text{Bird}}{\text{bl}_P(n_2)}$ clearly cannot be applied, we do not obtain $\text{bl}_P(n_2)$ and since this is the only way in which $\text{ok}(n_1)$ can be obtained (from the (reduced) formula $(\text{bl}_P(n_2) \wedge \text{bl}_P(n_3)) \supset \text{ok}(n_1)$ in W_\prec) we do not obtain $\text{ok}(n_1)$. If instead we have the theory $(D, W \cup \{\text{Penguin}, \neg \text{Swim}\}, \prec)$, we again obtain one extension, containing *Animal*, *Bird*, *Penguin*, and $\text{ok}(n_3)$. However we do not obtain $\text{ok}(n_1)$, $\text{ok}(n_2)$, nor $\text{bl}_P(n_1)$, $\text{bl}_P(n_2)$.

The following theorem summarizes the major technical properties of our approach, and demonstrates that rules are applied in the desired order:

Theorem 3 Let E be a consistent extension of $\mathcal{I}((D, W, \prec))$ for ordered default theory (D, W, \prec) . We have for all $\delta, \delta' \in D$ that

1. $n_\delta \prec n_{\delta'} \in E$ iff $\neg(n_\delta \prec n_{\delta'}) \notin E$
2. $\text{ok}(n_\delta) \in E_i$ and $\text{Prereq}(\delta) \in E_j$ and $\neg \text{Justif}(\delta) \notin E$ implies $\text{Conseq}(\delta) \in E_{\max(i,j)+3}$
3. $\text{ok}(n_\delta) \in E_i$ and $\text{Prereq}(\delta) \notin E$ implies $\text{bl}_P(n_\delta) \in E_{i+1}$
4. $\text{Prereq}(\delta) \in E$ where $\delta \neq \delta_T$ implies for every δ' where $\delta' < \delta$ we have $\text{ok}(n_{\delta'}) \notin E$
5. $\text{ok}(n_\delta) \in E$ iff for every δ' where $\delta < \delta'$ we have $\text{ok}(n_{\delta'}) \in E$ and $\text{bl}_P(n_\delta) \in E$

It follows immediately from the last two parts above that for $\delta \in D$, where $\delta \neq \delta_T$, if $\delta_a \in GD(D, E)$ then

- for every δ' where $\delta < \delta'$ we have $\text{bl}_P(n_\delta) \in E$, and
- for every δ' where $\delta' < \delta$ we have $\text{ok}(n_{\delta'}) \notin E$.

That is, if a non-trivial default is applied, then every \prec -greater default has an unprovable prerequisite, and every \prec -lesser default is not considered.

6 DISCUSSION

We have proposed and illustrated a general methodology for using Default Logic as an analytical tool and as an underlying formalism for the representation of knowledge. This role for Default Logic extends that advocated for formal logic in Knowledge Representation. The methodology involves the naming of default rules and the introduction of special-purpose predicates, for detecting conditions for default rule applicability and controlling a rule's application. This allows the encoding of specific strategies and policies governing a set of default rules. Given this, we present two examples, wherein Default Logic is

used to formalize preferences among properties and the inheritance of default properties. In (Delgrande and Schaub, 1999b) we have also shown how a notion of "similar individuals" can be encoded so that default rules apply uniformly to such similar individuals.

Thus, in our examples we show how Default Logic can be employed to provide a semantics for such phenomena and, on the other hand, provide an encoding for reasoning with such phenomena. Given that there are now comprehensive implementations of Default Logic available, it also becomes a straightforward matter to implement, for example, preference or property inheritance: one needs just implement the translation into Default Logic, and feed the result into a Default Logic theorem prover.

These examples suggest that the general methodology proposed here provides a general and useful approach to analyze and axiomatize various diverse phenomena. For example, our translations demonstrate that there are distinct notions having to do with preference and priority on the one hand, and property inheritance on the other. Hence we suggest that, in addition to directly representing nonmonotonic theories, Default Logic is appropriate as a general Artificial Intelligence formalism in which specific phenomena may be encoded. In fact, a stronger, perhaps more pragmatic, thesis⁶ can be advanced in view of implemented reasoning systems such as Smodels, (Niemelä and Simons, 1997), and DLV, (Eiter et al., 1997): that a subset of Default Logic, corresponding to extended logic programs under the answer set semantics, might provide just the appropriate approach for encoding and addressing such phenomena as advocated here.

References

- Allen, J. and Hayes, P. (1989). Moments and points in an interval-based temporal logic. *Computational Intelligence*, 5(4):225–238.
- Baader, F. and Hollunder, B. (1992). Embedding defaults into terminological knowledge representation formalisms. In Nebel, B., Rich, C., and Swartout, W., editors, *Proceedings of the Third International Conference on the Principles of Knowledge Representation and Reasoning*, pages 306–317, Cambridge, MA.
- Baader, F. and Hollunder, B. (1993). How to prefer more specific defaults in terminological default logic. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 669–674, Chambéry, France.
- Boutilier, C. (1992). What is a default priority? In Glasgow, J. and Hadley, R., editors, *Proceedings of the Ninth Canadian Conference on Artificial Intelligence*, pages 140–147, Vancouver, B.C.
- Brewka, G. (1994a). Adding priorities and specificity to default logic. In Pereira, L. and Pearce, D., editors, *European Workshop on Logics in Artificial Intelligence (JELIA '94)*, Lecture Notes in Artificial Intelligence, pages 247–260. Springer Verlag.

⁶We thank a reviewer for suggesting this.

- Brewka, G. (1994b). Reasoning about priorities in default logic. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, volume 2, pages 940–945. AAAI Press/The MIT Press.
- Brewka, G. (1996). Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research*, 4:19–36.
- Brewka, G. and Eiter, T. (1997). Preferred answer sets for extended logic programs. In Cohn, A., Schubert, L., and Shapiro, S., editors, *Proceedings of the Sixth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 86–97. Morgan Kaufmann Publishers.
- Brewka, G. and Gordon, T. (1994). How to buy a porsche: An approach to defeasible decision making. In *AAAI-94 Workshop on Computational Dialectics*, pages 28–38, Seattle, WA. AAAI Press.
- Cadoli, M., Eiter, T., and Gottlob, G. (1994). Default logic as a query language. In Doyle, J., Torasso, P., and Sandewall, E., editors, *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 99–108. Morgan Kaufmann Publishers.
- Cholewiński, P., Marek, V., Mikitiuk, A., and Truszczyński, M. (1999). Computing with default logic. *Artificial Intelligence*, 112(1-2):105–146.
- Delgrande, J. and Schaub, T. (1997a). Compiling reasoning with and about preferences into default logic. In Dean, T., editor, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 168–174, Nagoya, Japan. Morgan Kaufmann Publishers.
- Delgrande, J. and Schaub, T. (1997b). Compiling specificity into approaches to nonmonotonic reasoning. *Artificial Intelligence*, 90(1-2):301–348.
- Delgrande, J. and Schaub, T. (1997c). Reasoning with sets of preferences in default logic. In *The Second IJCAI Workshop on Nonmonotonic Reasoning, Action and Change*, Nagoya, Japan.
- Delgrande, J. and Schaub, T. (1999a). Expressing preferences in default logic. Technical report, Universität Potsdam. submitted for publication.
- Delgrande, J. and Schaub, T. (1999b). The role of default logic in knowledge representation. Technical Report, School of Computing Science, Simon Fraser University., presented at the *Workshop on Logic-Based Artificial Intelligence*, Washington, D.C.
- Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. (1997). A deductive system for nonmonotonic reasoning. In Dix, J., Furbach, U., and Nerode, A., editors, *Proceedings of the Fourth International Conference on Logic Programming and Non-Monotonic Reasoning*, volume 1265 of *Lecture Notes in Artificial Intelligence*, pages 363–374. Springer Verlag.
- Etherington, D. and Reiter, R. (1983). On inheritance hierarchies with exceptions. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 104–108. Morgan Kaufmann Publishers.
- Geffner, H. and Pearl, J. (1992). Conditional entailment: Bridging two approaches to default reasoning. *Artificial Intelligence*, 53(2-3):209–244.
- Gelfond, M. and Lifschitz, V. (1990). Logic programs with classical negation. In Warren, D. H. D. and Szeredi, P., editors, *Proceedings of the International Conference on Logic Programming*, pages 579–597. MIT.

- Gelfond, M. and Son, T. C. (1998). Reasoning with prioritized defaults. In Dix, J., Pereira, L. M., and Przymusinski, T. C., editors, *Proceedings of the 3th International Workshop on Logic Programming and Knowledge Representation (LWKR-97)*, volume 1471 of *LNAI*, pages 164–223, Berlin. Springer.
- Grosop, B. (1991). Generalizing prioritization. In Allen, J. A., Fikes, R., and Sandewall, E., editors, *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning*, pages 289–300, San Mateo, CA. Morgan Kaufmann.
- Hayes, P. (1985a). Naive physics I: Ontology for liquids. In Hobbs, J. and Moore, R., editors, *Formal Theories of the Commonsense World*, pages 71–108. Ablex.
- Hayes, P. (1985b). The second naive physics manifesto. In Hobbs, J. and Moore, R., editors, *Formal Theories of the Commonsense World*, pages 1–36. Ablex.
- Janhunen, T. (1999). Classifying semi-normal default logic on the basis of its expressive power. In Gelfond, M., Leone, N., and Pfeifer, G., editors, *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, volume 1730 of *Lecture Notes in Artificial Intelligence*, pages 19–33. Springer Verlag.
- Levesque, H., Pirri, F., and Reiter, R. (1998). Foundations for the situation calculus. *Linköping Electronic Articles in Computer and Information Science*, 3(18).
- Lifschitz, V. (1985). Closed-world databases and circumscription. *Artificial Intelligence*, 27:229–235.
- Lifschitz, V. (1996). Foundations of logic programming. In Brewka, G., editor, *Principle of Knowledge Representation*, pages 69–127. CSLI.
- McCarthy, J. (1979). First order theories of individual concepts and propositions. In Michie, D., editor, *Machine Intelligence 9*, pages 129–147. Edinburgh University Press.
- McCarthy, J. (1986). Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–116.
- McCarthy, J. and Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Michie, D. and Meltzer, B., editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press.
- Mercer, R. (1988). Using default logic to derive natural language suppositions. In Goebel, R., editor, *Proceedings of the Seventh Biennial Canadian Conference on Artificial Intelligence*, pages 14–21.
- Moore, R. (1980). Reasoning about knowledge and action. Technical Report 181, SRI International.
- Moore, R. (1982). The role of logic in knowledge representation and common-sense reasoning. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 428–433, Pittsburgh, PA.
- Niemelä, I. and Simons, P. (1997). Smodels: An implementation of the stable model and well-founded semantics for normal logic programs. In Dix, J., Furbach, U., and Nerode, A., editors, *Proceedings of the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 420–429. Springer-Verlag.

- Pearl, J. (1990). System Z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning. In Parikh, R., editor, *Proc. of the Third Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 121–135, Pacific Grove, Ca. Morgan Kaufmann Publishers.
- Perrault, C. (1987). An application of default logic to speech act theory. Technical Report CSLI-87-90, Stanford University.
- Poole, D. (1988). A logical framework for default reasoning. *Artificial Intelligence*, 36(1):27–48.
- Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13:81–132.
- Reiter, R. (1984). Towards a logical reconstruction of relational database theory. In Brodie, M., Mylopoulos, J., and Schmidt, J., editors, *On Conceptual Modeling*, pages 191–233. Springer-Verlag.
- Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–96.
- Reiter, R. and Crisculo, G. (1981). On interacting defaults. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 270–276, Vancouver, B.C.
- Reiter, R. and de Kleer, J. (1987). Foundations for assumption-based truth maintenance systems: Preliminary report. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 183–188. Morgan Kaufmann Publishers.
- Schaub, T. and Nicolas, P. (1997). An implementation platform for query answering in default logics: The XRay system, its implementation and evaluation. In Dix, J., Furbach, U., and Nerode, A., editors, *Proceedings of the Fourth International Conference on Logic Programming and Non-Monotonic Reasoning*, volume 1265 of *Lecture Notes in Artificial Intelligence*, pages 442–453. Springer Verlag.
- Turner, H. (1997). Representing actions in logic programs and default theories: A situation calculus approach. *Journal of Logic Programming*, 31(1–3):245–298.
- Wang, X., You, J., and Yuan, L. (1999). Compiling defeasible inheritance networks to general logic programs. *Artificial Intelligence*, 113(1–2):247–268.
- Zhang, Y. and Foo, N. (1997). Answer sets for prioritized logic programs. In Maluszynski, J., editor, *Proceedings of the International Symposium on Logic Programming (ILPS-97)*, pages 69–84. MIT Press.

Chapter 6

APPROXIMATIONS, STABLE OPERATORS, WELL-FOUNDED FIXPOINTS AND APPLICATIONS IN NONMONOTONIC REASONING

Marc Denecker,

*Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
marcd@cs.kuleuven.ac.be*

Victor Marek and Mirosław Truszczyński

*Computer Science Department, University of Kentucky
Lexington, KY 40506-0046, USA
marek@cs.engr.uky.edu mirek@cs.engr.uky.edu*

Abstract In this paper we develop an algebraic framework for studying semantics of nonmonotonic logics. Our approach is formulated in the language of lattices, bilattices, operators and fixpoints. The goal is to describe fixpoints of an operator O defined on a lattice. The key intuition is that of an *approximation*, a pair (x, y) of lattice elements which can be viewed as an approximation to each lattice element z such that $x \leq z \leq y$. The key notion is that of an *approximating operator*, a monotone operator on the bilattice of approximations whose fixpoints approximate the fixpoints of the operator O . The main contribution of the paper is an algebraic construction which assigns a certain operator, called the *stable operator*, to every approximating operator on a bilattice of approximations. This construction leads to an abstract version of the well-founded semantics. In the paper we show that our theory offers a unified framework for semantic studies of logic programming, default logic and autoepistemic logic.

Keywords: Nonmonotonic logics, operators on lattices, fixpoints, approximating operators, well-founded fixpoint, stable fixpoints

1 INTRODUCTION

We study algebraic foundations of semantics of nonmonotonic knowledge representation formalisms. The algebraic framework we use is that of lattices, operators and fixpoints. The key tool is the theorem of Tarski and Knaster (Tarski, 1955) on fixpoints of monotone operators on complete lattices. Our work is motivated by the fact that all major semantics of knowledge representation formalisms such as logic programming, default logic and modal non-monotonic logics are defined by means of fixpoints of suitably chosen operators on lattices of interpretations and possible-world structures. We derive general algebraic principles that lie behind these semantics.

Our work can be viewed as an extension of an abstract approach to logic programming proposed by Fitting. In a series of papers culminating in (Fitting, 2000), Fitting demonstrated that stable, supported, well-founded and Kripke-Kleene semantics of logic programs can be studied in abstract terms of fixpoints of two operators on a bilattice of 4-valued interpretations. One of these operators is the 4-valued van Emden-Kowalski operator T_P that generalizes a 2-valued van Emden-Kowalski operator T_P introduced in (van Emden and Kowalski, 1976). Fixpoints of the operator T_P yield the partial supported model semantics and Kripke-Kleene semantics for logic programs. The other operator is a 4-valued stable operator Ψ'_P introduced in (Przymusinski, 1990). The operator Ψ'_P can be regarded as a multi-valued generalization of the Gelfond-Lifschitz operator GL_P (Gelfond and Lifschitz, 1988). Fixpoints of the operator Ψ'_P determine the partial stable model semantics and the well-founded semantics.

In (Denecker et al., 1998; Denecker et al., 2000) we observed that an operator-based approach to logic programming put forth by Fitting can be adapted to the case of two other nonmonotonic systems: autoepistemic logic (Moore, 1984; Moore, 1985) and default logic (Reiter, 1980). In the case of autoepistemic logic, this abstract approach resulted in several new semantics. First, it allowed us to introduce for autoepistemic logic a counterpart to the semantics of extensions. Second, it led to generalizations of Kripke-Kleene and well-founded semantics. Most importantly, it exhibited the existence of a unifying framework behind all major semantics for autoepistemic logic. In the case of default logic, the operator-based approach led to a generalization of the Kripke-Kleene semantics and resulted in a uniform semantic framework for default logic, surprisingly similar to that discovered in the case of autoepistemic logic. In fact, in (Denecker et al., 2000) we proved that both frameworks are isomorphic and we argued that under the translation proposed in (Konolige, 1988), default logic can be viewed as a fragment of autoepistemic logic.

In this paper we extract essential algebraic elements underlying unified semantic frameworks for logic programming, autoepistemic logic and default logic developed in (Fitting, 2000; Denecker et al., 1998; Denecker et al., 2000). In the abstract setting we develop, we consider lattices, bilattices, operators on lattices and their approximations, that is, operators on bilattices. Elements of lattices represent some “points of interest”. Operators describe ways in which one “point of interest” might be revised (updated) into another one. We are in-

terested in fixpoints of operators on lattices as they are precisely those elements that cannot be revised away.

With each lattice we associate a certain bilattice (the product of the lattice by itself). The elements of such a bilattice can be interpreted as approximations to elements of the underlying lattice. To study fixpoints of an operator on a lattice, we introduce the concept of an approximating operator, defined on the associated bilattice. We demonstrate that studying fixpoints of approximating operators can provide us with insights into the structure and properties of fixpoints of operators they approximate. In particular, by considering all fixpoints of an approximating operator we obtain an abstract version of the Kripke-Kleene semantics. Adding some minimization requirements results in an abstract version of the well-founded semantics.

In knowledge representation applications “points of interest” represented by elements of lattices might be interpretations or possible-world structures describing truths (beliefs, knowledge) about a world specified by a base theory. Operators are formal descriptions of constraints on truth or belief sets used in revising one set of truths or beliefs into another one. We argue that our abstract setting yields as special cases semantic frameworks for logic programming, autoepistemic logic and default logic. We also show that all three systems exhibit an amazing similarity in the structure of the families of their semantics. By far the most important contribution of the paper is a general algebraic construction assigning to an arbitrary approximating operator its stable version. For each of the knowledge representation formalisms discussed here: logic programming, autoepistemic logic and autoepistemic logic, this construction allows us to reduce the study of all major semantics to the study of properties of a single operator.

Our work is concerned with abstract principles underlying nonmonotonic reasoning and with unified approaches to nonmonotonicity. In this respect it is somewhat similar to the work by Bochman (Bochman, 1996; Bochman, 1998a; Bochman, 1998b), and by Brass and Dix (Brass and Dix, 1999). Bochman develops an abstract proof-theoretic approach to nonmonotonicity based on the notion of a biconsequence relation. Brass and Dix characterize semantics for nonmonotonic systems in terms of general abstract postulates on their properties.

The paper is organized as follows. In Section 2 we briefly review key concepts and definitions related to lattices, bilattices and operators on them. In Section 3 we formally introduce the notion of an approximating operator and establish a number of basic properties of these operators. We also discuss there an abstract version of the Kripke-Kleene semantics. Next, in Section 4 for every approximating operator we define its stable operator and an abstract form of the well-founded semantics. We discuss applications of our approach in knowledge representation in Section 5. The last section contains conclusions, open problems and a discussion of future work.

2 PRELIMINARIES FROM LATTICE THEORY

A *lattice* is a partially ordered set $\langle L, \leq \rangle$ such that every two element set $\{x, y\} \subseteq L$ has a *least upper bound*, $\text{lub}(x, y)$, and a *greatest lower bound*, $\text{glb}(x, y)$. A lattice $\langle L, \leq \rangle$ is *complete* if every subset of L has both least upper and greatest lower bounds. Consequently, a complete lattice has a least element (\perp) and a greatest element (\top).

An *operator* on a lattice $\langle L, \leq \rangle$ is any function from L to L . An operator O on L is *monotone* if for every pair of elements $x, y \in L$,

$$x \leq y \text{ implies } O(x) \leq O(y).$$

Similarly, an operator O on L is *antimonotone* if for every pair x, y of elements from L ,

$$x \leq y \text{ implies } O(y) \leq O(x).$$

The composition of two antimonotone operators is monotone, as stated in the following result.

Proposition 1 If the operators $O_1 : L \rightarrow L$, $O_2 : L \rightarrow L$ are antimonotone, then the operator $O_1 \circ O_2$ is monotone.

Another straightforward observation asserts that operators that are both monotone and antimonotone are constant.

Proposition 2 If an operator $O : L \rightarrow L$ is monotone and antimonotone then it is constant.

The basic tool to study fixpoints of operators on lattices is the celebrated theorem by Tarski and Knaster (Tarski, 1955).

Theorem 1 Let O be a monotone operator on a complete lattice $\langle L, \leq \rangle$. Then, O has a fixpoint and the set of all fixpoints of O is a complete lattice. The least fixpoint of this lattice (that is, the least fixpoint of O) can be obtained by iterating O over \perp . The greatest fixpoint of this lattice (the greatest fixpoint of O) can be obtained by iterating O over \top .

We denote the least and the greatest fixpoints of the operator O by $\text{lfp}(O)$ and $\text{gfp}(O)$, respectively.

In applications it is often useful, and sometimes necessary, to approximate elements of lattices. We say that an element $z \in L$ is approximated by a pair $(x, y) \in L^2$ if $x \leq z \leq y$. Approximations of the form (x, x) are especially interesting. They provide a complete description of an element they approximate and so, we refer to them as *complete*. There is a straightforward one-to-one correspondence between L and the set of complete elements of L^2 .

Since approximations are the key concept of our approach, in the paper, we study the set L^2 , operators on L^2 and fixpoints of these operators.

The set L^2 can be endowed with two natural orderings. The first of them is a generalization of an ordering \leq from L . We will refer to it as the *lattice* ordering and use the same symbol \leq to denote it. Formally, it is defined by

$$(x, y) \leq (x_1, y_1) \text{ if } x \leq x_1 \text{ and } y \leq y_1.$$

The second ordering, called the *information* ordering, captures the intuition of increased precision of the approximation. This ordering, denoted \leq_i , is defined by

$$(x, y) \leq_i (x_1, y_1) \text{ if } x \leq x_1 \text{ and } y_1 \leq y.$$

It is easy to see that L^2 with each of these two orderings induces a complete lattice. In addition, it can be shown that the twelve distributivity laws involving the meets and joins with respect to both orderings all hold. Such algebraic structures are known as *bilattices* (Ginsberg, 1988; Fitting, 2000). They were used by Fitting in his discussion of semantics of logic programs with negation.

Not all pairs $(x, y) \in L^2$ can be interpreted as approximations to elements of L . For that to be the case, it is necessary that $x \leq y$. Thus, we say that a pair $(x, y) \in L^2$ is *consistent* if $x \leq y$. Otherwise, it is called *inconsistent*. Clearly, consistent pairs can be viewed as descriptions of our, in general, incomplete knowledge about elements from L that they approximate. Inconsistent pairs can be viewed as describing the fact that our knowledge about some unknown elements from L is inconsistent. The information ordering when applied to inconsistent pairs can be regarded as an ordering measuring the “degree of inconsistency”.

Clearly, the collection of consistent pairs does not form a sublattice of L^2 . Indeed, each element of the form (x, x) is a maximal consistent element of L^2 . Thus, no two different elements of the form (x, x) have a consistent upper bound. By allowing inconsistent approximations into our considerations we get an intuitive duality between consistent and inconsistent pairs, and between the degree of precision and the degree of inconsistency. We deal with a much richer algebraic structure and obtain a more elegant theory. In the same time, all main constructions described in the paper are, in fact, restricted to the consistent part of a bilattice of approximations and both the Kripke-Kleene and well-founded fixpoints, that we define later, are consistent (however, dual constructions for the inconsistent part of the bilattice can also be considered).

The theorem by Tarski and Knaster talks about fixpoints of monotone operators. It implies also some important properties of antimonotone operators. A pair of elements $x, y \in L$ is an *oscillating pair* an operator O on L if $y = O(x)$ and $x = O(y)$. In other words, x and y form an oscillating pair if and only if x is a fixpoint of $O^2 = O \circ O$ and $y = O(x)$. An oscillating pair (x, y) is an *extreme* oscillating pair for O if for every oscillating pair (x', y') for O , $(x, y) \leq_i (x', y')$ and $(x, y) \leq_i (y', x')$ (or equivalently, $x \leq x'$, $y' \leq y$). In particular, if (x, y) is an extreme oscillating pair then $x \leq y$. It is also easy to see that if an extreme oscillating pair exists, it is unique.

Theorem 2 Let O be an antimonotone operator on a complete lattice $\langle L, \leq \rangle$. Then, O^2 has a least fixpoint and a greatest fixpoint and $(\text{lfp}(O^2), \text{gfp}(O^2))$ is the unique extreme oscillating pair of O .

In this paper, we study fixpoints of operators on lattices by considering fixpoints of associated operators on bilattices. These operators often satisfy some monotonicity properties. Thus, in the remainder of this section, we present results on operators on L^2 that are monotone or antimonotone with respect to the orderings \leq and \leq_i . Before we present our results, we need more terminology.

Let us consider an operator A on L^2 . Let us denote by A^1 and A^2 the functions from L^2 to L such that

$$A(x, y) = (A^1(x, y), A^2(x, y)).$$

We say that A is *symmetric* if $A^1(x, y) = A^2(y, x)$. Clearly, if an operator $A : L^2 \rightarrow L^2$ is symmetric then for every $x \in L$, $A^1(x, x) = A^2(x, x)$.

In our discussion in the remainder of this paper we will restrict our considerations to symmetric operators only. The motivation for this restriction is twofold. First, all operators that appear in knowledge representation applications (for instance, the 4-valued van Emden-Kowalski operator T_P) are symmetric. Second, the assumption of symmetry results in a much more elegant theory. In particular, symmetric operators are *extending*, an important property in our theory of approximations (we introduce this notion in the next section). However, we stress that the assumption of symmetry is not essential and all major concepts and constructions described in the paper can be developed without it.

Proposition 3 A symmetric operator $A : L^2 \rightarrow L^2$ is \leq_i -monotone if and only if for every $y \in L$, $A^1(\cdot, y)$ is monotone and for every $x \in L$, $A^1(x, \cdot)$ is antimonotone (or equivalently, if and only if for every $y \in L$, $A^2(\cdot, y)$ is antimonotone and for every $x \in L$, $A^2(x, \cdot)$ is monotone).

The next result provides a similar characterization of all symmetric operators on L^2 that are monotone with respect to the ordering \leq .

Proposition 4 A symmetric operator $A : L^2 \rightarrow L^2$ is \leq -monotone if and only if for every $x, y \in L$, $A^1(x, \cdot)$ and $A^1(\cdot, y)$ are monotone (or, equivalently, if and only if for every $x, y \in L$, $A^2(x, \cdot)$ and $A^2(\cdot, y)$ are monotone).

Propositions 3 and 4, together with Proposition 2, imply a characterization of symmetric operators that are both \leq_i -monotone and \leq -monotone.

Proposition 5 An operator $A : L^2 \rightarrow L^2$ is symmetric and monotone with respect to both \leq_i and \leq if and only if there is a monotone operator $O : L \rightarrow L$ such that for every $x, y \in L$, $A(x, y) = (O(x), O(y))$.

Next, we present a description of symmetric operators on L^2 that are \leq_i -monotone and \leq -antimonotone.

Proposition 6 An operator $A : L^2 \rightarrow L^2$ is symmetric, \leq_i -monotone and \leq -antimonotone if and only if there is an antimonotone operator $O : L \rightarrow L$ such that for every $x, y \in L$, $A(x, y) = (O(y), O(x))$.

Propositions 5 and 6 imply that there is a one-to-one correspondence between monotone (antimonotone, respectively) operators on L and \leq_i -monotone and \leq -monotone (\leq_i -monotone and \leq -antimonotone, respectively) operators on L^2 .

When L is a complete lattice, it follows by Knaster-Tarski Theorem and by Theorem 2 that an \leq_i -monotone and \leq -antimonotone operator $A : L^2 \rightarrow L^2$ has \leq_i -least and \leq_i -greatest fixpoints and a \leq -extreme oscillating pair. Let us denote the \leq_i -least fixpoint of A by q_A , and the \leq_i -greatest fixpoint of A by Q_A . Similarly, let us denote the \leq -extreme oscillating pair for A by (e_A, E_A) .

If $A : L^2 \rightarrow L^2$ is, in addition, symmetric, by Proposition 6, there is an antimonotone operator $O : L \rightarrow L$ such that $A(x, y) = (O(y), O(x))$. Let us denote by q the least fixpoint of O^2 and by Q the greatest fixpoint of O^2 (Tarski-Knaster Theorem applies as O^2 is monotone). The following theorem, due essentially to Fitting, summarizes the relations between the fixpoints and extreme pairs defined above.

Theorem 3 Let L be a complete lattice. Let $A : L^2 \rightarrow L^2$ be a symmetric \leq_i -monotone and \leq -antimonotone operator on L^2 . Then:

1. $q_A = (q, Q)$, $Q_A = (Q, q)$, $e_A = (q, q)$, $E_A = (Q, Q)$
2. $q_A = \text{glb}_{\leq_i}(e_A, E_A)$ and $Q_A = \text{lub}_{\leq_i}(e_A, E_A)$
3. $e_A = \text{glb}_{\leq}(q_A, Q_A)$ and $E_A = \text{lub}_{\leq}(q_A, Q_A)$.

Proof: Let $O : L \rightarrow L$ be an antimonotone operator such that $A(x, y) = (O(y), O(x))$ (Proposition 6) and let q and Q be the least and the greatest fixpoints of O^2 , respectively. Then, (q, Q) is the extreme oscillating pair of O (Theorem 2), $O(q) = Q$ and $O(Q) = q$. Thus, $A(q, Q) = (O(Q), O(q)) = (q, Q)$ or, equivalently, (q, Q) is a fixpoint of A . Let (x, y) be a fixpoint of A . Then, $(x, y) = A(x, y) = (O(y), O(x))$ and $x = O(y)$ and $y = O(x)$. Thus, (x, y) is an oscillating pair for O . Since (q, Q) is the extreme oscillating pair for O , $(q, Q) \leq_i (x, y)$. It follows that (q, Q) is the least fixpoint of A or, in other words, that $q_A = (q, Q)$. The proof that $Q_A = (Q, q)$ is similar.

Next, observe that $A(q, q) = (O(q), O(q)) = (Q, Q)$ and $A(Q, Q) = (O(Q), O(Q)) = (q, q)$. Thus, $((q, q), (Q, Q))$ is an oscillating pair for A . Let $((x, y), (x', y'))$ be an oscillating pair for A . Then, (x, y) and (x', y') are fixpoints of A^2 . Consequently, x, y, x' and y' are all fixpoints of O^2 . It follows that $q \leq x, y, x', y' \leq Q$ and so, $(q, q) \leq_i (x, y), (x', y') \leq_i (Q, Q)$. Thus, $((q, q), (Q, Q))$ is the extreme oscillating pair for A (or, equivalently, if $e_A = (q, q)$ and $E_A = (Q, Q)$, then $A(e_A) = E_A$ and $A(E_A) = e_A$).

The assertions (2) and (3) follow immediately from the assertion (1) and the fact that $q \leq Q$. \square

3 APPROXIMATING OPERATORS

Our paper is an attempt to identify basic algebraic principles behind semantics of nonmonotonic reasoning formalisms. The key concept to our approach is that of an approximating operator. Given an operator O on a lattice L the goal is to gain insights into its fixpoints and into constructive techniques to find them. To this end, we will consider operators on the bilattice L^2 .

Definition 1 An operator $A : L^2 \rightarrow L^2$ extends an operator $O : L \rightarrow L$ if for every $x \in L$, $A(x, x) = (O(x), O(x))$. An operator $A : L^2 \rightarrow L^2$ is extending if for every $x \in L$, there is $y \in L$ such that $A(x, x) = (y, y)$.

We define the *diagonal* of L^2 to be the set $\{(x, x) : x \in L\}$ (that is, the set of all complete approximations). If an operator $A : L^2 \rightarrow L^2$ extends $O : L \rightarrow L$ then the behavior of A on the diagonal fully describes the behavior of O . In particular, complete fixpoints of A correspond to fixpoints of O .

Proposition 7 Let O be an operator on a lattice L and let A be an operator on L^2 extending O . Then, x is a fixpoint of O if and only if (x, x) is a fixpoint of A .

If A is symmetric then for each lattice element x , $A^1(x, x) = A^2(x, x)$. Hence $A(x, x)$ is complete and, consequently, A is extending. This observation is stated in the following result. As we mentioned earlier, it is one of the motivations for restricting our discussion to symmetric operators only.

Proposition 8 If an operator $A : L^2 \rightarrow L^2$ is symmetric then A is extending.

It follows directly from the definition of an extending operator that to study fixpoints of an operator O one might construct an appropriate extending operator A and study its fixpoints instead. Clearly, complete fixpoints of the operator A would then provide a complete description of the fixpoints of O .

It seems that this new problem is essentially the same as the original one. There is, however, one difference. An extending operator A is defined on a bilattice. Consequently, all its fixpoints are approximated by the least element (\perp, T) of the bilattice (referred to as the *weakest approximation*). Two natural questions arise: are there better approximations to fixpoints of A than this trivial one, and can they be constructed. In general the answer is negative. However, the answer is positive if A is \leq_i -monotone. In such case, we can iterate A starting with the weakest approximation. In each iteration we improve the precision of the approximation. When no further improvement is possible the process terminates and results in the \leq_i -least fixpoint of A . This fixpoint approximates all fixpoints of A , it is often better than the weakest approximation (\perp, T) and it can be constructed! The possibility of constructing the least fixpoint of a \leq_i -monotone extending operator leads us to one of the key concepts of the paper (in view of our remarks, we introduce it with the stronger requirement of symmetry).

Definition 2 An operator $A : L^2 \rightarrow L^2$ approximates an operator $O : L \rightarrow L$ if A is symmetric, extends O and is \leq_i -monotone. An operator $A : L^2 \rightarrow L^2$ is approximating if it is symmetric and \leq_i -monotone.

We say that an operator $A : L^2 \rightarrow L^2$ is *consistent* if it maps consistent pairs to consistent pairs, that is whenever (x, y) is consistent, then also $A(x, y)$ is consistent. The following two results formally state basic properties of approximating operators.

Proposition 9 If $A : L^2 \rightarrow L^2$ is an approximating operator, then A is consistent.

Corollary 1 Let $A : L^2 \rightarrow L^2$ be an approximating operator for an operator $O : L \rightarrow L$. Then, A has a \leq_i -least fixpoint. This fixpoint is consistent and approximates every fixpoint of O .

The notion of \leq_i -least fixpoint of an operator A approximating operator O in lattice L is an important concept. The least fixpoint of A approximates all fixpoints of O . Speaking informally, it determines information that is common to all the fixpoints of O . Next, if the \leq_i -least fixpoint is complete, say it is of the form (x, x) , then x is the only fixpoint of O . Moreover, in such case, this unique fixpoint of O is based on a constructive principle of building it incrementally by iterating the approximating operator A . Since in the case of logic programming, the concept of the \leq_i -least fixpoint of an approximating operator can be specialized to the Kripke-Kleene semantics, we refer to the \leq_i -least fixpoint of an approximating operator A as the *Kripke-Kleene fixpoint* of A . We denote this fixpoint by α_A .

Clearly, an operator O on a lattice may have several approximating operators. Each gives rise to its Kripke-Kleene fixpoint and the corresponding approximation of all fixpoints of O . The problem of finding an approximation operator providing the best (in some sense) approximation is, in general, a challenging one. We do have some results that pertain to it. They will be a subject of another paper. Here we will only mention two simple special cases when an operator O is monotone or antimonotone.

Let O be a monotone operator on L . By Proposition 5, the operator $A_O(x, y) = (O(x), O(y))$ is \leq_i -monotone. It is also symmetric, consistent and extends the operator O . Hence, A_O is an approximating operator for O . By Proposition 5, A_O is \leq -monotone. In fact, Proposition 5 implies that A_O is a unique approximating operator for O that is \leq -monotone. The least \leq_i -fixpoint of A_O is $(\text{lfp}(O), \text{lfp}(O))$. We will call A_O the *trivial* approximating operator for a monotone operator O ¹.

Similarly, if O is an antimonotone operator on L then, by Proposition 6, the operator $A_O(x, y) = (O(y), O(x))$ is \leq_i -monotone. In addition, A_O is symmetric, consistent and it extends O . Hence, it is an approximating operator for O . By Proposition 6, A_O is \leq -antimonotone and, in fact, it is a unique approximating operator for O that is \leq -antimonotone. We will call A_O the *trivial* approximating operator for an antimonotone operator O . Theorem 3 characterizes the fixpoints and the extreme oscillating pair of the trivial approximating operator for an antimonotone operator O .

¹This algebraic property of monotone operators explains why all major nonmonotonic semantics coincide on the class of Horn theories (or programs) and are given by the least fixpoint construction.

4 STABLE OPERATOR AND WELL-FOUNDED FIXPOINT

In the case of logic programming, fixpoints of the van Emden-Kowalski operator T_P determine (2-valued) supported models of a program P . Supported model semantics (also known as Clark completion semantics) is often too weak for knowledge representation applications. The class of *stable* models was proposed in (Gelfond and Lifschitz, 1988) as the basis of an alternative semantics for programs with negation.

It is well-known that stable models form a subclass of the class of supported models. Our goal in this section is to study abstract principles relating supported and stable models. More generally, we search for principles that might allow us to identify interesting special subclasses in the class of all fixpoints of an operator O defined on a complete lattice L . Since, as argued in the previous section, fixpoints of O can be studied by considering approximating operators, our approach is to search for principles that allow us to narrow down the class of fixpoints of approximating operators. Approximating operators are symmetric and \leq_i -monotone. The results in this section rely only on these two assumptions (however, as mentioned earlier, the assumption of symmetry is not essential for our theory).

The fact that bilattices are also ordered by the (generalization of) lattice ordering suggests a possible approach. Minimizing truth is the key idea underlying commonsense reasoning and the process of jumping to conclusions. In our abstract setting, it boils down to minimization with respect to \leq and we focus our attention on those fixpoints of A which are \leq -minimal. However, the principle of \leq -minimality is in itself not sufficient. For instance, it is well known that not every minimal supported model of a logic program P is stable.

In this section we describe an algebraic construction that assigns to every \leq_i -monotone operator A on a bilattice L^2 its *stable* operator C_A defined also on L^2 . We demonstrate that every fixpoint of the operator C_A is a \leq -minimal fixpoint of A . Later in the paper we argue that fixpoints of stable operators appear naturally in several nonmonotonic reasoning formalisms such as logic programming, default logic and autoepistemic logics, thus validating our construction.

Definition 3 Let L be a complete lattice. Let an operator $A : L^2 \rightarrow L^2$ on a bilattice L^2 be symmetric and \leq_i -monotone.

1. The *complete stable operator* for A , $C_A : L \rightarrow L$, is defined by $C_A(y) = \text{lfp}(A^1(\cdot, y))$ (or, equivalently, by, $C_A(y) = \text{lfp}(A^2(y, \cdot))$).
2. The *stable operator* for A , $C_A : L^2 \rightarrow L^2$ is defined by $C_A(x, y) = (C_A(y), C_A(x))$.

Since for every $y \in L$ the operators $A^1(\cdot, y)$ and $A^2(y, \cdot)$ are monotone (Proposition 3), the operators C_A and C_A are well-defined.

The intuition behind the stable operator is as follows. We are given an operator $A : L^2 \rightarrow L^2$. This operator can be viewed as a description of a way to revise approximations (x, y) . Our goal is to derive from A a different (but

related) way to "revise" approximations. We proceed as follows. Given an approximation (x, y) , to construct the lower bound of a new approximation we use y — our current upper estimate. We consider the operator $A^1(\cdot, y)$ which models revisions of the lower bounds of those approximations with the upper bound fixed to y . Since $A^1(\cdot, y)$ is a monotone operator, there is a natural candidate for the intended new lower bound — the least fixpoint of $A^1(\cdot, y)$. To construct the new upper bound, we proceed similarly. We use the current lower bound x and consider the operator $A^2(x, \cdot)$. This operator is monotone and its least fixpoint is selected as the new intended upper bound. Since A is symmetric, the same operator, C_A , can be used to determine both the lower and the upper bound.

Let us consider an operator A that is both \leq_i - and \leq -monotone. Such operators are described in Proposition 5. They are of the form $A(x, y) = (O(x), O(y))$, where O is monotone. It follows that $C_A(y) = \text{lfp}(O)$ and does not depend on y . Thus, we get the following result.

Proposition 10 Let L be a complete lattice. Let $A : L^2 \rightarrow L^2$ be an operator monotone with respect to \leq_i and \leq . Then C_A is constant.

If an operator A is \leq_i -monotone and \leq -antimonotone then, by Proposition 6, there is an antimonotone operator O such that $A(x, y) = (O(y), O(x))$. Consequently, $A(\cdot, y) = O(y)$. It follows that $C_A(y) = O(y)$, that is, the stable operator for the operator A is A itself.

Proposition 11 Let L be a complete lattice. Let $A : L^2 \rightarrow L^2$ be an operator monotone with respect to \leq_i and antimonotone with respect to \leq . Then $C_A = A$.

We will now study properties of the stable operator C_A and its fixpoints. Our first result shows that fixpoints of C_A are \leq -minimal fixpoints of A (the converse statement in general does not hold).

Theorem 4 Let L be a complete lattice. Let an operator $A : L^2 \rightarrow L^2$ on a bilattice L^2 be \leq_i -monotone. Every fixpoint of the stable operator C_A is a \leq -minimal fixpoint of A .

Proof: In this proof we will use some additional basic properties of operators on lattices. An element x of a lattice L is a *pre-fixpoint* of an operator $O : L \rightarrow L$ if $O(x) \leq x$. The argument of Tarski and Knaster shows that if L is a complete lattice and O is a monotone operator on L then for every pre-fixpoint x of O , $\text{lfp}(O) \leq x$.

Let (x, y) be a fixpoint of C_A . It follows that $(x, y) = (C_A(y), C_A(x))$. By the definition of C_A , $x = \text{lfp}(A^1(\cdot, y))$, and hence $A^1(x, y) = x$. Similarly, $y = \text{lfp}(A^1(\cdot, x)) = \text{lfp}(A^2(x, \cdot))$. Thus, $A^2(x, y) = y$. Consequently, (x, y) is a fixpoint of A .

Next, assume that (x', y') is a fixpoint of A such that $(x', y') \leq (x, y)$. It follows that $x' \leq x$ and hence, by antimonotonicity of $A^2(\cdot, y')$ (Proposition 3), we have that $A^2(x, y') \leq A^2(x', y') = y'$. Thus, y' is a pre-fixpoint of the operator $A^2(x, \cdot)$. Since $A^2(x, \cdot)$ is monotone, and y is its least fixpoint, it

follows that $y \leq y'$. Since $(x', y') \leq (x, y)$, $y = y'$. Similarly, one can derive that $x = x'$. Thus, $(x', y') = (x, y)$ which, in turn, implies that (x, y) is a \leq_i -minimal fixpoint of A . \square

Theorem 4 shows, in particular, that if A is \leq_i -monotone, a fixpoint of C_A is also a fixpoint of A . We will call every fixpoint of the stable operator C_A a *stable* fixpoint of A .

Directly from the definition of the operators C_A and from Proposition 3 it follows that C_A is antimonotone. Consequently, by Proposition 6, C_A is \leq_i -monotone and \leq -antimonotone.

Proposition 12 Let L be a complete lattice. Let A be a symmetric \leq_i -monotone operator on L^2 . Then, C_A is an antimonotone operator on L and C_A is a \leq_i -monotone and \leq -antimonotone operator on L^2 .

Propositions 11 and 12 imply the following corollary that states that applying the stability construction to a stable operator does not lead to a new operator anymore.

Corollary 2 Let L be a complete lattice. Let A be a symmetric \leq_i -monotone operator on L^2 . Then $C_{C_A} = C_A$.

It is also easy to see that C_A is symmetric and extends the operator C_A . Thus, we obtain the following corollary to Proposition 12.

Corollary 3 Let L be a complete lattice. Let A be a \leq_i -monotone operator on L^2 . Then, the stable operator C_A is a trivial approximation of the complete stable operator C_A .

Since C_A is \leq_i -monotone and \leq -antimonotone, it has a \leq_i -least fixpoint, a \leq_i -greatest fixpoint and also a \leq -extreme oscillating pair. As explained in Theorem 3, these concepts are interrelated and can be expressed in terms of the fixpoints of the operator $C_A^2 = C_A \circ C_A$.

The \leq_i -least fixpoint of C_A is of particular interest as it provides an approximation to every stable fixpoint of A . We call the \leq_i -least fixpoint of C_A the *well-founded fixpoint* of a \leq_i -monotone operator A and denote it by β_A . The choice of the term is dictated by the fact that in the case of logic programming, the least fixpoint of the stable operator for the 4-valued van Emden-Kowalski operator T_P yields the well-founded semantics.

The following result gathers several properties of the well-founded fixpoint of an operator that generalize properties of the well-founded model of a logic program.

Theorem 5 Let L be a complete lattice. Let $A : L^2 \rightarrow L^2$ be a \leq_i -monotone symmetric operator.

1. The Kripke-Kleene fixpoint α_A and the well-founded fixpoint β_A satisfy $\alpha_A \leq_i \beta_A$
2. For every stable fixpoint x of A , $\beta_A \leq_i x$

3. If β_A is complete then it is the only consistent stable fixpoint of A .
4. The operator C_A is consistent and, consequently, β_A is consistent, too.

Proof: The assertion (1) follows from the fact that α_A is the \leq_i -least fixpoint of A and fixpoints of C_A are fixpoints of A (Theorem 4).

Stable fixpoints of A are precisely the fixpoints of C_A . Since β_A is the least fixpoint of C_A , the assertion (2) follows.

To prove (3), we first observe that since β_A is complete, it is a consistent stable fixpoint of A . Let us consider a consistent stable fixpoint of A , say x . Then x is a fixpoint of C_A . Thus, $\beta_A \leq_i x$. Since β_A is complete, it is a maximal consistent element of L^2 . Thus, $x = \beta_A$ and (3) follows.

Finally, C_A is an approximating operator (it approximates operator C_A). Thus, the assertion (4) follows from Proposition 9 and Corollary 1. \square

We will now assume that A is an approximating operator for an operator $O : L \rightarrow L$ and discuss the relationship between the fixpoints of C_A and fixpoints of O .

Proposition 13 Let L be a complete lattice. Let $A : L^2 \rightarrow L^2$ be an approximating operator for an operator $O : L \rightarrow L$. If (x, x) is a fixpoint of C_A then x is a \leq -minimal fixpoint of O .

Proof: The proposition follows immediately from theorem 4. \square

It follows from Proposition 13 that if A is an approximating operator for an operator O then fixpoints of O corresponding to complete fixpoints of the stable operator C_A form an antichain.

We will next consider the case when O is monotone. In this case we can use the trivial approximation of O , A_O . Using Proposition 10 and the discussion that precedes it, we obtain the following result.

Proposition 14 Let L be a complete lattice. If $O : L \rightarrow L$ is a monotone operator, then for every $x \in L$, $C_{A_O}(x, y) = (\text{lfp}(O), \text{lfp}(O))$ (that is, C_{A_O} is constant).

If O is monotone, its trivial approximation A_O may have many fixpoints in general and many complete fixpoints, in particular. However, by Proposition 14, the stable operator for A_O has only one fixpoint and it corresponds precisely to the least fixpoint of O . In the context of logic programming, this result says that a Horn logic program P has a unique stable model and that it coincides with the least Herbrand model of P .

Consider an operator O defined on a complete lattice L . How can we associate with this operator its well-founded fixpoint? In order to do so, we need to construct an approximation A of O and use the well founded fixpoint of A as the well-founded fixpoint of O . There may be several approximating operators and the well-founded fixpoints of these operators may have different properties. As mentioned earlier, a study of best approximations will be presented in another paper.

5 APPLICATIONS IN KNOWLEDGE REPRESENTATION

The results presented here provide us with a uniform framework for semantic studies of major knowledge representation formalisms: logic programming, autoepistemic logic and default logic. Namely, all major semantics for each of these formalisms can be derived from a single operator.

In the case of logic programming, our results extend an algebraic approach proposed in (Fitting, 2000). The lattice of interest here is that of 2-valued interpretations of the Herbrand base of a given program P . We will denote it by \mathcal{A}_2 . The corresponding bilattice $\mathcal{A}_2 \times \mathcal{A}_2$ is isomorphic with the bilattice \mathcal{A}_4 of 4-valued interpretations (in 4-valued Belnap logic). Our results imply that the central role in logic programming is played by the 4-valued van Emden-Kowalski operator T_P defined on the bilattice $\mathcal{A}_2 \times \mathcal{A}_2$ (or, equivalently, on bilattice \mathcal{A}_4). First, the operator T_P approximates the 2-valued van Emden-Kowalski operator T_P . Second, fixpoints of T_P represent 4-valued supported models, consistent fixpoints of T_P represent partial (3-valued) supported models and complete fixpoints of T_P describe supported models of P . The \leq_i -least fixpoint of T_P (it exists as T_P is approximating) defines the Kripke-Kleene semantics of P .

Perhaps most importantly, it turns out that our general construction assigning the stable operator to every approximating operator when applied to T_P yields the 4-valued Przymusinski operator Ψ'_P and the 2-valued Gelfond-Lifschitz operator GL_P . That is, the stable operator for T_P coincides with Ψ'_P and the complete stable operator for T_P coincides with GL_P . Thus, the semantics of 4-valued, partial (3-valued) and 2-valued stable models can also be derived from the operator T_P . The same is true for the well-founded semantics since it is determined by the \leq_i -least fixpoint of the stable operator of T_P . The structure of the family of operators and semantics for logic programming that can be derived from the operator T_P is presented in Figure 6.1.

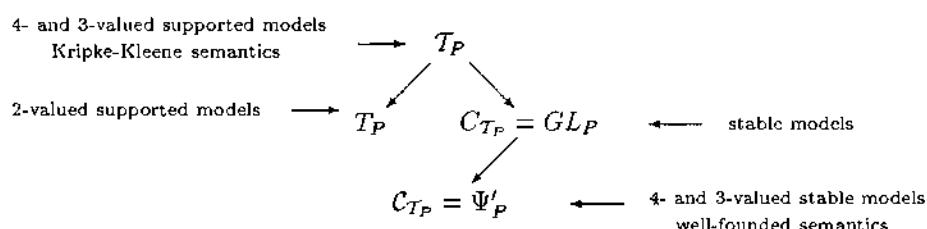


Figure 6.1 Operators and semantics associated with logic programming

In (Denecker et al., 1998; Denecker et al., 2000) we developed an algebraic approach to semantics for autoepistemic and default logics. In both cases, our approach can be regarded as a special case of the general approach presented here. In the investigations of autoepistemic and default logics we consider the lattice \mathcal{W} of possible-world structures (sets of 2-valued interpretations) and the corresponding bilattice \mathcal{B} of belief pairs (Denecker et al., 1998). In the case of autoepistemic logic, the central place is occupied by the operator D_T (T is

a given modal theory) defined on the bilattice of belief pairs and introduced in (Denecker et al., 1998). It turns out to be an approximating operator for the operator D_T used by Moore to define the notion of an expansion (Moore, 1984). Thus, the concepts of partial expansions and expansions can be derived from D_T . Similarly, the Kripke-Kleene semantics can be obtained from D_T as its least fixpoint. The stable operator for D_T and its complete counterpart lead to semantics for autoepistemic logic that to the best of our knowledge have not been studied in the literature: the semantics of extensions, partial extensions and the well-founded semantics, that are closely related to the corresponding semantics for default logic (Denecker et al., 2000). The emerging structure of operators and semantics for autoepistemic logic is depicted in Figure 6.2.

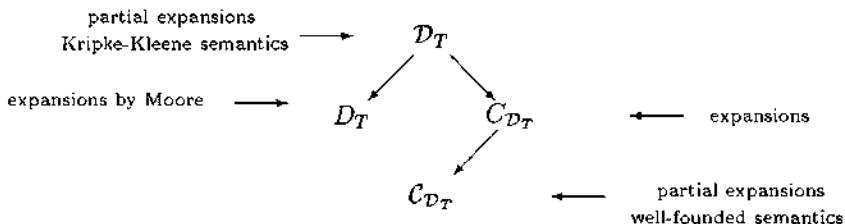


Figure 6.2 Operators and semantics associated with autoepistemic logic

A very similar picture emerges in the case of default logic, too. In (Denecker et al., 2000) we described an operator E_Δ on the bilattice of belief pairs and argued that all major semantics for default logic can be derived from it. Among them are the semantics of weak extensions (Marek and Truszczyński, 1989a), partial weak extensions and the corresponding Kripke-Kleene semantics for default logic. In addition, the complete stable operator for E_Δ coincides with the Guerreiro-Casanova operator characterizing extensions (Guerreiro and Casanova, 1990) and the \leq_i -least fixpoint of the stable operator C_{E_Δ} for E_Δ yields the well-founded semantics for default logic described by Baral and Subrahmanian in (Baral and Subrahmanian, 1991). The semantics landscape of default logic is depicted in Figure 6.3.

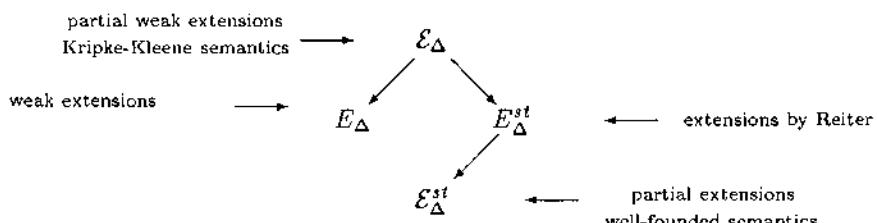


Figure 6.3 Operators and semantics associated with default logic

The similarity between the families of the semantics for logic programming, default logic and autoepistemic logic is striking. It has been long known that logic program clauses can be interpreted as default rules (Marek and Truszczyński, 1989b; Bidoit and Froidevaux, 1991). Namely, a logic program

clause

$$a \leftarrow b_1, \dots, b_m, \text{not}(c_1), \dots, \text{not}(c_n)$$

can be interpreted as a default

$$\frac{b_1 \wedge \dots \wedge b_m : \neg c_1, \dots, \neg c_n}{a}$$

It turns out that under this translation the operators T_P and $\mathcal{E}_{\Delta(P)}$ are very closely related ($\Delta(P)$ stands for the default theory obtained from the logic program P by means of the translation given above). Namely, let us observe that each interpretation I can be associated with the possible-world structure consisting of all interpretations J such that $I(p) = t$ implies $J(p) = t$. Thus, the lattice \mathcal{A}_2 can be viewed as a sublattice of W and the restriction of the operator $\mathcal{E}_{\Delta(P)}$ to this sublattice essentially coincides with T_P . It follows that all the derived operators are similarly related, and we obtain a perfect match between the semantics for logic programming and the semantics for default logic.

Similarly, in (Konolige, 1988) it was proposed to interpret a default

$$\frac{\beta_1 \wedge \dots \wedge \beta_m : \neg \gamma_1, \dots, \neg \gamma_n}{\alpha}$$

as a modal formula

$$K\beta_1 \wedge \dots K\beta_m \wedge \neg K\neg \gamma_1 \wedge \dots \wedge \neg K\neg \gamma_n \supset \alpha.$$

It turns out that under this translations the operators \mathcal{E}_Δ and $\mathcal{D}_{T(\Delta)}$ coincide (here $T(\Delta)$ is the modal image of a default theory Δ under Konolige's translation). As before, all corresponding pairs of derived operators also coincide. Thus, we obtain a perfect match between the semantics for default and autoepistemic theories².

6 CONCLUSIONS

In the paper we presented an algebraic theory of fixpoints of non-monotone operators. We argued that essentially all major semantics for logic programming, autoepistemic logic and default logic can be described in an elegant and uniform way by applying our algebraic fixpoint theory to a particular operator: T_P in logic programming, \mathcal{D}_T in autoepistemic logic, and \mathcal{E}_Δ in default logic. When, as our study appears to indicate, a number of different logics, developed from different perspectives, can be derived from a uniform principle, the question must be raised of the knowledge theoretic role and meaning of this principle.

²However, this correspondence does not align expansions by Moore and extensions by Reiter. These two semantics occupy different locations in the corresponding hierarchies. A more detailed discussion of this issue can be found in (Denecker et al., 2000).

We hypothesize that our theory provides a generalized algebraic account of non-monotone constructions and non-monotone induction in mathematics. Tarski's fixpoint theory can be considered as a general method for modeling monotone constructions and positive inductive definitions. It seems that the theory presented here extends this theory to the general case of non-monotone inductive definitions. The investigation of this hypothesis amounts to an empirical study of constructive techniques in mathematics and of logical formalizations of such techniques, including existing formalizations of non-monotone induction such as iterated inductive definitions and inflationary fixpoint logic. Early results in this direction are presented in (Denecker, 1998).

If we can validate our hypothesis, then the theory presented here elucidates new fundamental relationships between different scientific domains, including nonmonotonic reasoning, logic programming, database theory and inductive definitions. It may also shed more light on the role of different logics for knowledge representation. The discussion of these issues will be the subject of another publication.

Acknowledgments

This work was partially supported by the NSF grants CDA-9502645 and IRI-9619233.

References

- Baral, C. and Subrahmanian, V. (1991). Dualities between alternative semantics for logic programming and nonmonotonic reasoning (extended abstract). In A. Nerode, W. Marek, and V.S. Subrahmanian, editors, *Logic programming and non-monotonic reasoning (Washington, DC, 1991)*, pages 69–86, Cambridge, MA. MIT Press.
- Bochman, A. (1996). Biconsequence Relations for Nonmonotonic Reasoning. In L. Carlucci Aiello, J. Doyle, and S. Shapiro, editors, *Principles of Knowledge Representation and Reasoning, Proceedings of the Fifth International Conference (KR96)*, pages 482–492, Morgan Kaufmann Publishers.
- Bochman, A. (1998a). A Logical Foundation for Logic Programming I: Biconsequence Relations. *Journal of Logic Programming* 35:151–170.
- Bochman, A. (1998b). A Logical Foundation for Logic Programming II: Semantics of General Logic Programs. *Journal of Logic Programming* 35:171–194.
- Bidoit, N. and Froidevaux, C. (1991). Negation by default and unstratifiable logic programs. *Theoretical Computer Science*, 78(1, (Part B)):85–112.
- Brass, S. and Dix, J. (1999). Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming* 38:167–213.
- Denecker, M. (1998). The well-founded semantics is the principle of inductive definition. In J. Dix, L.F. Farinas del Cerro, and U. Furbach, editors, *Logics in Artificial Intelligence*, pages 1–16. Springer-Verlag.
- Denecker, M., Marek, V., and Truszczyński, M. (1998). Fixpoint 3-valued semantics for autoepistemic logic. In B.G. Buchanan and R. Uthurusamy, editors, *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 840 – 845. MIT Press.

- Denecker, M., Marek, V., and Truszczyński, M. (2000). Unified semantic treatment of default and autoepistemic logics. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Principles of Knowledge Representation and Reasoning, Proceedings of the Seventh International Conference (KR2000)*, pages 840 – 845. Morgan Kaufmann Publishers.
- Fitting, M. C. (1999). Fixpoint semantics for logic programming – a survey. *Theoretical Computer Science*. To appear. Available at:
<http://comet.lehman.cuny.edu/fitting/bookspapers/logprog.html>
- Gelfond, M. and Lifschitz, V. (1988). The stable semantics for logic programs. In R. Kowalski, and K. Bowen, editors, *Proceedings of the 5th International Symposium on Logic Programming*, pages 1070–1080, Cambridge, MA. MIT Press.
- Ginsberg, M. (1988). Multivalued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316.
- Guerreiro, R. and Casanova, M. (1990). An alternative semantics for default logic. Preprints for The Third International Workshop on Nonmonotonic Reasoning, South Lake Tahoe, pages 141–157.
- Konolige, K. (1988). On the relation between default and autoepistemic logic. *Artificial Intelligence*, 35(3):343–382.
- Marek, W. and Truszczyński, M. (1989a). Relating autoepistemic and default logics. In R.J. Brachman, H.J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning, Proceedings of the First International Conference (KR89)*, pages 276–288, Morgan Kaufmann Publishers.
- Marek, W. and Truszczyński, M. (1989b). Stable semantics for logic programs and default theories. In E. Lusk and R. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 243–256. MIT Press.
- Moore, R. (1984). Possible-world semantics for autoepistemic logic. In *Proceedings of the Workshop on Non-Monotonic Reasoning*, pages 344–354. Reprinted in: M. Ginsberg, editor, *Readings on nonmonotonic reasoning*, pp. 137–142, Morgan Kaufmann, 1990.
- Moore, R. (1985). Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94.
- Przymusinski, T. (1990). The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445–464.
- Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13(1–2):81–132.
- Tarski, A. (1955). Lattice-theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309.
- van Emden, M. and Kowalski, R. (1976). The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742.

v

LOGIC FOR CAUSATION AND ACTIONS

Chapter 7

GETTING TO THE AIRPORT: THE OLDEST PLANNING PROBLEM IN AI

Vladimir Lifschitz,

*Department of Computer Science
University of Texas at Austin
Austin, TX 78712
jvl@cs.utexas.edu*

Norman McCain,

Baker University

Emilio Remolina,

University of Texas at Austin

Armando Tacchella

Università di Genova

Abstract The problem discussed in this paper is described in a 1959 paper by John McCarthy as follows: *Assume that I am seated at my desk at home and I wish to go to the airport. My car is at my home also. The solution of the problem is to walk to the car and drive the car to the airport.* In the spirit of what is now known as the logic approach to AI, McCarthy proposed to address this problem by first giving “a formal statement of the premises” that a reasoning program would use to draw the relevant conclusions. Our goal here is to take a careful look at this episode from the early history of AI and to identify some of the logical and algorithmic ideas related to the airport problem that have emerged over the years.

Keywords: Action languages, C language, causation, Causal Calculator, deduction, frame problem, nonmonotonic reasoning, planning.

One way out of this difficulty is to replace this axiom by

$$at(I, neighborhood(desk)),$$

where *neighborhood* represents a function that maps every region to a slightly larger region—its “immediate neighborhood.” We would write, for instance,

$$at(scissors, desk)$$

to express that the scissors are in a drawer (or perhaps on top) of the desk, but $at(I, desk)$ would not be an acceptable postulate. The difference between $at(I, car)$ and $at(I, neighborhood(car))$ could be used to distinguish between being inside the car and standing next to it.

Alternatively, we can justify the first of axioms (7.1) as given above if we agree to understand the symbol *desk* to represent an immediate neighborhood of the desk. This convention would not apply to the other symbols: *car*, as before, denotes the part of space occupied by the car, rather than a neighborhood of the car, and similarly for *I*, *home*, *airport* and *county*.

The formalization given in Section 4 below uses this simpler (although admittedly ad hoc) solution.

2.2 ARRIVING AT A PLAN BY DEDUCTION

About the expected conclusion (7.6) Bar-Hillel said:

It sounds rather incredible that the machine could have arrived at its conclusion—which, in plain English, is “Walk from your desk to your car!”—by sound deduction. This conclusion surely could not possibly follow from the premise in any serious sense. Might it not be occasionally cheaper to call a taxi and have it take you over to the airport?

(See (McCarthy, 1990), pages 17–18.)

Indeed, the idea that a plan is a deductive consequence of the statement of the planning problem seems to presuppose the uniqueness of the plan: the problem is solved *only if this particular plan is executed*. In algebra, if 5 is a solution to a given equation, we do not expect $x = 5$ to be “deducible” from the equation in any sense—unless 5 is the only solution.

The relationship between a formal statement of the problem and a plan is not the provability relation. It is better to say that the plan “satisfies” the statement of the problem in some sense. Whatever the precise meaning of this word is, it’s essential that several different objects may “satisfy” the statement of a planning problem. (And there are no objects with this property if the problem is not solvable.)

In deductive planning, for instance, plans are represented by ground terms of the situation calculus (Green, 1969). A problem is described by specifying, first, an axiomatic theory *T* that describes the initial situation and the effects of actions, and, second, a goal condition *G(s)*. A situation term *t* “satisfies” $\langle T, G(s) \rangle$ if *T* entails *G(t)*.

In satisfiability planning (Kautz and Selman, 1992), plans are represented by propositional interpretations, and the satisfaction relation in question

is essentially the satisfaction relation of propositional logic. Answer set planning (Subrahmanian and Zaniolo, 1995; Dimopoulos et al., 1997; Lifschitz, 1999a) is similar, except that its underlying satisfaction relation is nonmonotonic.

2.3 THE TIME FACTOR

Further in Bar-Hillel's comments, we read:

Let me also point out that in the example the time factor has never been mentioned, probably for the sake of simplicity. But clearly this factor is here so important that it could not possibly be disregarded without distorting the whole argument. Does not the solution depend, among thousands of other things, also upon the time of my being at my desk, the time at which I have to be at the airport, the distance from the airport, the speed of my car, etc.

(See (McCarthy, 1990), page 18.)

Over the years, several temporal formalisms were proposed for use in formalizing actions, beginning with the situation calculus (McCarthy and Hayes, 1969). As to the description of the airport domain in (McCarthy, 1959), it is not correct to say that the time factor is ignored there altogether: the use of *did* in (7.5) hints at a time difference between action $go(x, y, z)$ and its effect $at(I, y)$. This syntactic mechanism helps us distinguish the effects of an action, as in (7.5), from its preconditions, as in (7.3). But it does not allow us to talk about the execution of several actions in a row. In this respect, it is similar to the language of STRIPS operators (Fikes and Nilsson, 1971), to ADL (Pednault, 1987) and to other action description languages (Gelfond and Lifschitz, 1998).

The temporal formalism used in this paper is action language \mathcal{C} (Giunchiglia and Lifschitz, 1998), reviewed in Section 4.1 below. This language has a simple syntax but is substantially more expressive than STRIPS.

2.4 IDENTIFYING THE RELEVANT PREMISES

Bar-Hillel said also in his comments:

A deductive argument, where you have first to find out what are the relevant premises, is something that many humans are not always able to carry out successfully. I do not see the slightest reason to believe that, at present, machines should be able to perform things that humans find trouble in doing. I do not think there could possibly exist a program which would, given any problem, divide all facts in the universe into those which are and those which are not relevant for that problem. Developing such a program seems to me by 10^{10} orders of magnitude more difficult than, say, the Newell—Simon problem of developing a heuristic for deduction in the propositional calculus. This cavalier way of jumping over orders of magnitude only tends to becloud the issue and throw doubt on ways of thinking for which I have a great deal of respect. By developing a powerful program language, you may have paved the way

for the first step in solving problems of the kind treated in your example, but the claim of being well on the way towards their solution is a gross exaggeration. This was the major point of my objections.

(See (McCarthy, 1990), page 19.)

Today, admittedly, AI researchers have about as little to say on this subject as they did in 1959. There is no need to explain that, in setting up the formalization of the airport example below, the task of identifying the relevant premises was not automated.

3 OTHER DIFFICULTIES

3.1 GOING TO A REGION

The destination of the action *go* may be a large “region” that has subregions (rather than a specific “location,” as in the case of the move action in the blocks world). This leads to an interesting question. If I go to a region *y* then, according to (7.5), fluent *at(I, y)* becomes true; but what about the effect of this action on fluent *at(I, y')*, where *y'* is a part of *y*? For instance, when I go home, how does this action affect fluent *at(I, desk)*?

We see here three choices.

One is to say that fluent *at(I, desk)* may remain false or may become true, depending on the particular way of executing the action of going home. The action becomes nondeterministic.

Second, we can say that this fluent always remains false. Going home is understood then as going to an unspecified location within the home that is disjoint from all the subregions of *home* that have names in the language (*desk* and *car*, in the initial state of the airport problem).

Finally, we can say that going home sometimes “involves” going to the desk, and sometimes it does not. In the first case, fluent *at(I, desk)* becomes true; in the second case, it does not. This view is adopted in the formalization below.

The idea that an execution of an action of type *A₁* “involves” an execution of an action of type *A₂* may be understood in two ways. We may think that there is a single event which instantiates both the *A₁* and *A₂* action types, or we may think that there are two events appropriately related—they at least must happen concurrently. Formally, we are not committed to either interpretation. When we say that the action *A₁* occurs, we mean that some event occurs that instantiates the *A₁* action type. When we say that the action *A₂* also occurs, we mean that some event occurs that instantiates the *A₂* action type. Whether these are the same event or two is not an issue that we address.¹ The view that a single event may instantiate distinct action types or descriptions has been articulated by the philosopher Donald Davidson ([Davidson, 1967]).

¹The issue is, however, of some importance. The number of distinct actions in a plan is one useful measure of its quality. In order to count actions we have to be able to tell when we have two and when we have one.

3.2 NEED FOR NONMONOTONIC REASONING

The discovery of the frame problem and the invention of the nonmonotonic formalisms that are capable of solving it may have been the most significant events so far in the history of research on reasoning about actions. A large part of this story is described in (Shanahan, 1997).

In simple cases, the frame problem can be solved monotonically (Schubert, 1990), but in the case of the airport domain the use of nonmonotonic reasoning seems almost imperative. The reason for this is that going to a region may have rather complicated ramifications. Axiom (7.5) describes the direct effect of going to y : in the resulting state, I am at y . But this action has also indirect effects. First, for every region y' that includes y , I am at y' . Second, for every region y' that is disjoint from y , I am not at y' . Formal nonmonotonic reasoning helps us describe these indirect effects in a concise way.

A nonmonotonic solution to the frame problem provides a formalization of the informal principle called the “commonsense law of inertia.” In application to a fluent f , this principle asserts that, as time goes by, f tends to keep the same value that it had in the past. The approach to expressing inertia adopted in action language \mathcal{C} is outlined in Section 4.1.

Another use of nonmonotonic reasoning in our formalization of the airport example has to do with the closed world assumption. Axioms (7.4) give little information about the extents of the predicates *walkable* and *drivable*; they do not tell us, for instance, whether the airport has any of these two properties, or whether the county is walkable. We can observe that if a region x is so small that one can walk (or drive), in principle, between any two locations within it, then any subregion of x has the same property. The axiom expressing this idea will allow us to derive, for instance, *drivable(airport)* from the axioms *drivable(county)* and *at(airport, county)*. It also will allow us to derive *drivable(home)*. (We understand *drivable(X)* to mean only that the distances within X are not too large, not necessarily that there are no obstacles against driving within X .) Even so, we will not be able to prove any negative facts about *walkable* or *drivable*. One way to overcome this difficulty is to postulate the closed world assumption for these two predicates: by default, a region is not walkable and not drivable. This idea, just like the idea of inertia, calls for the use of a nonmonotonic formalism.

4 AIRPORT DOMAIN IN ACTION LANGUAGE C

4.1 LANGUAGE

According to (Giunchiglia and Lifschitz, 1998) (see also (Gelfond and Lifschitz, 1998, Section 6)), a description of an action domain in action language \mathcal{C} is a set of propositions of two kinds: *static laws*

and *dynamic laws*

caused F if G after U .

Here F , G and U are logical formulas. The atomic subformulas of F and G are names of propositional fluents. U may contain, in addition, names of “elementary actions”; these names are used as atomic formulas also. An assignment of truth values to the names of elementary actions represents, intuitively, the composite action that consists in executing concurrently, during some fixed time interval, all elementary actions to which the value t is assigned. According to the semantics of \mathcal{C} , every action description represents a “transition system”—a directed graph whose edges correspond to the transitions caused by the execution of actions.

To illustrate the use of \mathcal{C} notation, consider a few examples.

1. The effect of $go(x, y, z)$ on $at(I, y)$ that McCarthy represented by formula (7.5) would be expressed in \mathcal{C} by

go(x, y, z) causes at(I, y).

This expression stands for the dynamic law

caused $at(I, y)$ if true after $go(x, y, z)$.

The general definition of the abbreviation **causes**, as well as the definitions of other abbreviations for \mathcal{C} propositions, can be found in (Gelfond and Lifschitz, 1998, Section 6).

2. Consider now the first of formulas (7.3), which expresses a sufficient condition for the executability of $go(y, z, walking)$. In \mathcal{C} , a similar condition might be written as

**nonexecutable $go(y, z, walking)$
if $\neg \exists x(walkable(x) \wedge at(y, x) \wedge at(z, x) \wedge at(I, y))$**

which stands for the dynamic law

**caused false if true
after $\neg \exists x(walkable(x) \wedge at(y, x) \wedge at(z, x) \wedge at(I, y)) \wedge go(y, z, walking)$.**

Actions described in \mathcal{C} are presumed to be executable if there is no evidence to the contrary, and the **nonexecutable** construct provides a way to express such evidence. The version of \mathcal{C} described in the papers mentioned above and used in this paper does not permit quantifiers. To comply with this limitation, $\exists x$ can be replaced by a finite disjunction over the regions represented in the language.

3. The closed world assumption for *walkable* (Section 3.2) can be expressed in \mathcal{C} by

default $\neg walkable(x)$

which stands for the static law

caused $\neg walkable(x)$ if $\neg walkable(x)$.

The role of this proposition in \mathcal{C} is similar to the role of the normal default

$$\begin{array}{c} \text{: } \neg\text{walkable}(x) \\ \hline \neg\text{walkable}(x) \end{array}$$

in default logic (Reiter, 1980). Intuitively, it says that, whenever x is not walkable, there is a cause for this; when x is walkable, however, this fact requires a special explanation provided by “positive” postulates such as

$$\text{caused walkable(home).}$$

This expression, similar to the first of postulates (7.4), stands for the static law

$$\text{caused walkable(home) if true.}$$

4. One possible counterpart of (7.2) in \mathcal{C} is

$$\text{always at}(x, y) \wedge \text{at}(y, z) \supset \text{at}(x, z)$$

which stands for the static law

$$\text{caused false if } \neg(\text{at}(x, y) \wedge \text{at}(y, z) \supset \text{at}(x, z)).$$

This proposition eliminates the states of the world in which at is nontransitive, so that any action whose effect is to make (7.2) false is nonexecutable. This version of (7.2) is a “qualification constraint” in the sense of (Lin and Reiter, 1994).

- Another possibility is to postulate the static law

$$\text{caused at}(x, z) \text{ if } \text{at}(x, y) \wedge \text{at}(y, z).$$

This is the version used in our formalization of the airport example. Besides eliminating the “bad” states, it allows us to draw conclusions about indirect effects of actions. It tells us, in particular, that any action which makes $\text{at}(x, y) \wedge \text{at}(y, z)$ true has also another effect: it makes $\text{at}(x, z)$ true. For instance, if I go to a subregion y of z then one ramification of this is that $\text{at}(I, z)$ becomes true; this was discussed in Section 3.2.

5. The commonsense law of inertia is not a “built-in” feature of \mathcal{C} , but the inertia assumption can be easily expressed in this language. For instance,

$$\text{inertial at}(x, y)$$

stands for dynamic law

$$\text{caused at}(x, y) \text{ if } \text{at}(x, y) \text{ after at}(x, y).$$

If $\text{at}(x, y)$ was true and remained true after executing an action then there is a cause for this. Whenever this fluent changes its value from t to f, a causal explanation is required. Such explanations are provided by other causal laws—the dynamic and static laws describing the direct and indirect effects of actions on $\text{at}(x, y)$.

4.2 FORMALIZATION

In our formalization of the airport example, action names are expressions of the form $go(X, M)$ where the metavariable X ranges over the symbols for regions

$$i, desk, car, home, airport, county \quad (7.7)$$

and M is either *walking* or *driving*. This is more concise than McCarthy's notation $go(x, y, z)$: we suppress the "source" argument x and keep only the "destination" argument and the "mode" argument.

We use fluent names of three kinds:

$$at(X, Y)$$

where each of the argument positions is occupied by one of symbols (7.7), and

$$walkable(U), drivable(U)$$

where U is one of the symbols in (7.7) that represent "big areas":

$$home, airport, county. \quad (7.8)$$

Although $walkable(U)$ and $drivable(U)$ are formally treated as fluents, their truth values are not going to be affected by any actions.

The first group of postulates consists of the static laws describing properties of *at*:

caused $at(X, Z)$ if $at(X, Y) \wedge at(Y, Z)$,
caused $\neg at(X, Z)$ if $at(X, Y) \wedge disjoint(Y, Z) \quad (Y \neq Z)$,
caused $\neg at(X, X)$.

Here X, Y, Z range over (7.7), and *disjoint*(Y, Z) stands for

$$\neg at(Y, Z) \wedge \neg at(Z, Y). \quad (7.9)$$

This abbreviation is motivated by the stipulation that the regions occupied by objects (7.7) never overlap: if two different regions Y, Z from that list satisfy (7.9) then they are disjoint.

The next two groups of postulates describe the direct effects of actions:

$go(X, M)$ causes $at(i, X)$,
 $go(X, driving)$ causes $at(car, X)$

and their preconditions:

nonexecutable $go(X, M)$ if $at(i, X)$,
nonexecutable $go(X, driving)$ if $\neg at(i, car)$,
nonexecutable $go(X, walking)$ if $\neg \forall_U (walkable(U) \wedge at(i, U) \wedge at(X, U))$,
nonexecutable $go(X, driving)$ if $\neg \forall_U (drivable(U) \wedge at(i, U) \wedge at(X, U))$.

Here U ranges over the names of big areas (7.8).

We also postulate two restrictions on the concurrent execution of actions. First, it is impossible to walk and to drive simultaneously:

nonexecutable $go(X, \text{walking}) \wedge go(Y, \text{driving})$.

Second, going to a region involves going to all supersets of that region (see Section 3.1):

nonexecutable $go(X, M) \wedge \neg go(Y, M)$ if $at(X, Y) \wedge \neg at(i, Y)$.

The next group of postulates provides information related to the sizes of objects:

always $\neg at(car, desk)$,

caused $walkable(home)$,

caused $walkable(U)$ if $at(U, V) \wedge walkable(V)$,

caused $drivable(county)$,

caused $drivable(U)$ if $at(U, V) \wedge drivable(V)$.

The first of them expresses that the car is too big to be fully contained in the immediate neighborhood of the desk. This fact is relevant because it explains why it would be impossible to get back from the airport to the desk in one step, by driving not followed by walking. (Alternatively, this could be explained by the impossibility of driving inside the home, in spite of the fact that the home is small enough to be “drivable.”)

Finally, we need to postulate that at satisfies the commonsense law of inertia

inertial $at(X, Y), \neg at(X, Y)$

and that $walkable$ and $drivable$ are normally false:

default $\neg walkable(U)$,

default $\neg drivable(U)$.

4.3 TRANSITION SYSTEM

The semantics of C (Giunchiglia and Lifschitz, 1998) defines how the propositions above describe a transition system. Every state of this transition system is characterized by an assignment of truth values to the fluent names

$at(X, Y)$, $walkable(U)$, $drivable(U)$.

One of the states, s_0 , is the initial state of the airport problem. The fluents that are true in state s_0 are

(7.10)

$$\begin{aligned} & walkable(home), \\ & drivable(airport), drivable(county), drivable(home), \\ & at(airport, county), at(home, county), \\ & at(desk, county), at(i, county), at(car, county), \\ & at(desk, home), \end{aligned}$$

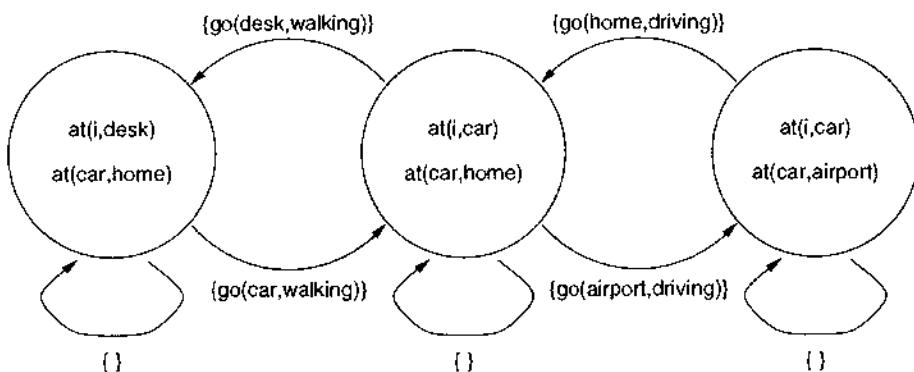


Figure 7.1 Part of the transition system describing the airport domain

$$\text{at}(i, \text{desk}), \text{ at}(i, \text{home}), \text{ at}(\text{car}, \text{home}). \quad (7.11)$$

Fluents (7.10) are not affected by any of the actions $\text{go}(X, M)$, so that they hold not only in state s_0 but also in all states that are reachable from s_0 in the transition system. Fluents (7.11) may become false after the execution of a sequence of actions.

Figure 7.1 shows state s_0 along with the two states that can be reached from it by executing actions. State s_0 is shown on the left. There are two actions that can be executed in this state: doing nothing (the loop labeled { }) and walking to the car. In the new state shown in the center, one can either do nothing, or walk back to the desk, or drive to the airport (the state on the right). In this last state, one can either do nothing or drive back home.

5 SOLVING THE PLANNING PROBLEM

5.1 CAUSAL CALCULATOR AND SATISFIABILITY PLANNING

The Causal Calculator² is an implementation of the propositional causal logic from (McCain and Turner, 1997). Domain descriptions in \mathcal{C} are translated by the Causal Calculator first into the language of causal theories and then into propositional logic. The models of the propositional theory correspond to paths in the transition system described by the original domain description in \mathcal{C} .

Planning is carried out by satisfiability checking, as proposed in (Kautz and Selman, 1992). The Causal Calculator conjoins the formulas describing an initial state and goal to the propositional theory, and then calls a propositional solver, such as SATO (Zhang, 1997), to find a model. If a model is found, it

²<http://www.cs.utexas.edu/users/tag/cc/>. See (Lifschitz, 2000) for other examples of the use of this system.

corresponds to a possible history of the domain, and the assignments it makes to the action symbols correspond to a plan. For deterministic domains, any plan found in this way is guaranteed to be valid (McCain and Turner, 1998).

A *C* input file for the Causal Calculator consists of declarations, propositions in language *C* (or, more often, schemas with metavariables whose instances are propositions in *C*), problems (for instance, planning problems) and comments. Among its declarations, a *C* input file usually contains a directive to include the "standard" file *C.t* which contains rewrite rules for translating from *C* into the language of causal theories, as well as various sorts, variables, constants, and domain independent causal laws that have been found useful in formalizing action domains.

5.2 AIRPORT DOMAIN IN THE CAUSAL CALCULATOR

The input file for the airport domain (*airport-domain.t*) shown below corresponds to the formalization of the domain in *C* described in Section 4.2. The sort names *inertialFluent*, *defaultFalseFluent*, and *action* that appear below in the *constants* declaration are declared in *C.t*. The declarations

```
: - constants
at(object,object) :: inertialFluent ;
walkable(object), drivable(object)
:: defaultFalseFluent.
```

are equivalent to the declarations

```
: - constants
at(object,object) :: fluent ;
walkable(object), drivable(object) :: fluent.
```

together with the schemas postulated at the end of Section 4.2:

```
inertial at(X,Y), -at(X,Y).
default -walkable(X).
default -drivable(X).
```

(*X* and *Y* are meta-variables of sort *object*). Hopefully, these remarks will suffice to enable the reader to understand the following file.³

```
%%% File 'airport-domain.t'

:- macros
disjoint(#1,#2) ->
(-((#1)=(#2)) && -at(#1,#2) && -at(#2,#1)).
```

³Both this file and the file *airport-problem.t* are available on-line at <http://www.cs.utexas.edu/users/tag/ccalc/ccalc.1.23/examples/airport/>.

```

:- include 'C.t'.

:- sorts
object >> region; mode.

:- variables
X,Y,Z      :: object;
U,V        :: region;
M          :: mode.

:- constants
i, car, desk           :: object;
home, airport, county :: region;
walking, driving       :: mode;
at(object,object)     :: inertialFluent;
walkable(region), drivable(region)
                      :: defaultFalseFluent;
go(object,mode)        :: action.

% Properties of at

caused at(X,Z) if at(X,Y) && at(Y,Z).
caused -at(X,Z) if at(X,Y) && disjoint(Y,Z).
caused -at(X,X).

% Effects of actions

go(X,M)      causes at(i,X).
go(X,driving) causes at(car,X).

% Action preconditions

nonexecutable go(X,M) if at(i,X).

nonexecutable go(X,driving) if -at(i,car).

nonexecutable go(X,walking)
  if - (\vee U: (walkable(U) && at(i,U) && at(X,U))). 

nonexecutable go(X,driving)
  if - (\vee U: (drivable(U) && at(i,U) && at(X,U))). 

% Restrictions on the concurrent execution of actions

nonexecutable go(X,walking) && go(Y,driving).

```

```

nonexecutable go(X,M) && -go(Y,M)
    if at(X,Y) && -at(i,Y).

% Sizes of objects

always -at(car,desk).

caused walkable(home).
caused walkable(U) if at(U,V) && walkable(V).

constant walkable(U).

caused drivable(county).
caused drivable(U) if at(U,V) && drivable(V).

constant drivable(U).

```

5.3 PLANNING

Besides the description of the airport domain shown above, the input given to the Causal Calculator includes the description of the planning problem:

```

%%% File 'airport-problem.t'

:- include 'airport-domain.t'.

:- plan
facts :: 
0: at(i,desk),
0: at(desk,home),
0: at(car,home),
0: at(home,county),
0: at(airport,county),
0: -at(desk,car),
0: disjoint(home,airport);
goal :: 
2: at(i,airport).

```

Each of the lines beginning with the time stamp 0 is an initial condition. These conditions express that s_0 (Section 4.3) is the initial state of the planning problem. The line beginning with the time stamp 2 tells us that the goal is to make $at(i, airport)$ true after the execution of 2 actions.⁴

Given this input file, the Causal Calculator produces the following output:

⁴If the number of steps required to solve a planning problem is unknown, the Causal Calculator can explore a range of possibilities. For instance, by writing `goal :: 5-20` we instruct the Causal Calculator to look first for a plan of length 5; if there is no such plan, try 6, etc., up to 20.

calling sato...

run time (seconds)

0.08

0. drivable(airport) drivable(county) drivable(home)
 walkable(home) at(airport,county) at(car,county)
 at(car,home) at(desk,county) at(desk,home)
 at(home,county) at(i,car) at(i,desk) at(i,home)

ACTIONS: go(car,walking)

1. drivable(airport) drivable(county) drivable(home)
 walkable(home) at(airport,county) at(car,county)
 at(car,home) at(desk,county) at(desk,home)
 at(home,county) at(i,car) at(i,county) at(i,home)

ACTIONS: go(airport,driving)

2. drivable(airport) drivable(county) drivable(home)
 walkable(home) at(airport,county) at(car,airport)
 at(car,county) at(desk,county) at(desk,home)
 at(home,county) at(i,airport) at(i,car) at(i,county)

The output shows the SATO runtime, the actions to be executed at times 0 and 1, and the fluents that hold at each of the time instants 0, 1, 2 when the plan is executed.

6 CONCLUSION

The solution to the airport problem presented in this paper is based on a number of ideas—some old, some new—coming from several areas of logic-based AI.

Nonmonotonic Reasoning. As a nonmonotonic formalism, the input language of the Causal Calculator can be viewed as a subset of default logic in the sense of Reiter [(Reiter, 1980)]. The process of “literal completion” (McCain and Turner, 1997) that translates rules of causal logic into propositional formulas is a modification of the completion semantics of negation as failure in logic programming (Clark, 1978) that treats positive and negative literals in a symmetric way, in the style of (Gelfond and Lifschitz, 1991). As discussed in (Lifschitz, 1997, Section 3), the formalization of the closed world assumption in causal logic can be viewed as the use of circumscription (McCarthy, 1986).

Frame Problem. The solution to the frame problem incorporated in \mathcal{C} and in the Causal Calculator is related to the method of (Reiter, 1980, Section 1.1.4). Claims to the opposite (Hanks and McDermott, 1987) notwithstanding, Reiter’s solution turned out to be completely satisfactory after the rest of the postulates were formulated in the right way ((Turner, 1997, Section 5.2); see also (Lifschitz, 1999b, Section 3)). The frame problem becomes more difficult in the presence of actions with indirect effects. This issue, known as the “ramification problem,”

was clarified over the last decade on the basis of the ideas of (Geffner, 1990) and (Lin and Reiter, 1994).

Action Languages. The idea of an action language as a language for describing transition systems was articulated by Pednault [(Pednault, 1994)]. After action language \mathcal{A} was defined and related to logic programming in (Gelfond and Lifschitz, 1993), it was extended and modified by several authors. Language \mathcal{C} is one of the most expressive action description languages introduced so far.

Satisfiability Planning. The success of this idea (due to Kautz and Selman ((Kautz and Selman, 1992), (Kautz and Selman, 2000)) is determined by the remarkable progress in the development of fast propositional solvers. "Between 1991 and 1996 the size of hard satisfiability problems that could be feasibly solved grew from ones involving less than 100 variables to ones involving over 10,000 variables" (Selman et al., 1997). On the other hand, the Causal Calculator demonstrates that the method is applicable to planning problems described in very expressive languages. This last fact is particularly essential in case of the airport problem whose difficulty is representational rather than algorithmic.

The approach to actions and planning described in this paper represents only one of several streams of research in this area. References to the work not mentioned here can be found in (Shanahan, 1997) and in recent issues of the *Electronic Transactions on AI: Reasoning about Actions and Change*.⁵

Acknowledgements

We are grateful to Esra Erdem, Jack Minker and Hudson Turner for constructive criticisms. This work was partially supported by National Science Foundation under grant IIS-9732744.

References

- Clark, K. (1978). Negation as failure. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York.
- Davidson, D. (1967). The logical form of action sentences. In *The Logic of Decision and Action*, pages 81–120. University of Pittsburgh Press.
- Dimopoulos, Y., Nebel, B., and Koehler, J. (1997). Encoding planning problems in non-monotonic logic programs. In Steel, S. and Alami, R., editors, *Proc. European Conf. on Planning 1997*, pages 169–181. Springer-Verlag.
- Fikes, R. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208.
- Geffner, H. (1990). Causal theories for nonmonotonic reasoning. In *Proc. AAAI-90*, pages 524–530. AAAI Press.
- Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385.

⁵<http://www.ida.liu.se/ext/etai/rac/>.

- Gelfond, M. and Lifschitz, V. (1993). Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322.
- Gelfond, M. and Lifschitz, V. (1998). Action languages.⁶ *Electronic Transactions on AI*, 3, pages 195–210.
- Giunchiglia, E. and Lifschitz, V. (1998). An action language based on causal explanation: Preliminary report. In *Proc. AAAI-98*, pages 623–630. AAAI Press.
- Green, C. (1969). Application of theorem proving to problem solving. In Walker, D. and Norton, L., editors, *Proc. IJCAI*, pages 219–240. The MITRE Corporation.
- Hanks, S. and McDermott, D. (1987). Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412.
- Kautz, H. and Selman, B. (1992). Planning as satisfiability. In *Proc. ECAI-92*, pages 359–363.
- Kautz, H. and Selman, B. (2000). Encoding domain and control knowledge for propositional planning. In Minker, J., editor, *Logic-Based Artificial Intelligence*, pages 169–186, Kluwer Academic Publishers, Norwell, Massachusetts 02061.
- Lifschitz, V. (2000). Missionaries and cannibals in the causal calculator. In *Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conf.*, pages 85–96.
- Lifschitz, V. (1997). On the logic of causal explanation. *Artificial Intelligence*, 96:451–465.
- Lifschitz, V. (1999a). Answer set planning. In *Proc. ICLP-99*, pages 23–37.
- Lifschitz, V. (1999b). Success of default logic. In Levesque, H. and Pirri, F., editors, *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 208–212. Springer-Verlag.
- Lin, F. and Reiter, R. (1994). State constraints revisited. *Journal of Logic and Computation*, 4:655–678.
- McCain, N. and Turner, H. (1997). Causal theories of action and change. In *Proc. AAAI-97*, pages 460–465.
- McCain, N. and Turner, H. (1998). Satisfiability planning with causal theories. In Cohn, A., Schubert, L., and Shapiro, S., editors, *Proc. Sixth Int'l Conf. on Principles of Knowledge Representation and Reasoning*, pages 212–223.
- McCarthy, J. (1959). Programs with common sense. In *Proc. Teddington Conf. on the Mechanization of Thought Processes*, pages 75–91, London. Her Majesty's Stationery Office. Reproduced in McCarthy, 1990.
- McCarthy, J. (1986). Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 26(3):89–116. Reproduced in McCarthy, 1990.
- McCarthy, J. (1990). *Formalizing Common Sense: Papers by John McCarthy*. Ablex, Norwood, NJ.
- McCarthy, J. and Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B. and Michie, D., editors,

⁶<http://www.ep.liu.se/ea/cis/1998/016>.

- Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh. Reproduced in McCarthy, 1990.
- Pednault, E. (1987). Formulating multi-agent, dynamic world problems in the classical planning framework. In Georgeff, M. and Lansky, A., editors, *Reasoning about Actions and Plans*, pages 47–82. Morgan Kaufmann, San Mateo, CA.
- Pednault, E. (1994). ADL and the state-transition model of action. *Journal of Logic and Computation*, 4:467–512.
- Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13:81–132.
- Schubert, L. (1990). Monotonic solution of the frame problem in the situation calculus: an efficient method for worlds with fully specified actions. In Kyburg, H., Loui, R., and Carlson, G., editors, *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer.
- Selman, B., Kautz, H., and McAllester, D. (1997). Ten challenges in propositional reasoning and search. In *Proc. IJCAI-97*, pages 50–54.
- Shanahan, M. (1997). *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press.
- Subrahmanian, V. and Zaniolo, C. (1995). Relating stable models and AI planning domains. In *Proc. ICLP-95*, pages 233–247.
- Turner, H. (1997). Representing actions in logic programs and default theories: a situation calculus approach. *Journal of Logic Programming*, 31:245–298.
- Zhang, H. (1997). An efficient propositional prover. In *Proc. CADE-97*, pages 272–275.

VI

PLANNING AND PROBLEM SOLVING

Chapter 8

ENCODING DOMAIN AND CONTROL KNOWLEDGE FOR PROPOSITIONAL PLANNING

Henry Kautz,
University of Washington
Seattle, Washington, 98195
kautz@cs.washington.edu

Bart Selman
Cornell University
Ithaca, NY 14853
selman@cs.cornell.edu

Abstract Propositional satisfiability checking is a powerful approach to domain-independent planning. In nearly all practical applications, however, there exists an abundance of domain-specific knowledge that can be used to improve the performance of a planning system. This knowledge is traditionally encoded as procedures or rules that are tied to the details of the planning engine. We present a way to encode domain knowledge in a purely declarative, algorithm independent manner. We demonstrate that the same heuristic knowledge can be used by completely different search engines, one systematic, the other using greedy local search. This approach enhances the power of planning as satisfiability: solution times for some problems are reduced from days to seconds.

Keywords: SATPLAN, BlackBox, planning, satisfiability, domain knowledge

* This is a revised and expanded version of a paper which appeared in the *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, Pittsburgh, PA, 1998.

1 INTRODUCTION

Planning is a notoriously hard combinatorial search problem. In (Kautz and Selman, 1996), we considered domain-independent, state space planning as a computational challenge. This style of planning is a core problem in AI; similar computational issues arise in many other models, such as reactive planning, planning with uncertainty and utilities, planning with metric time, and so on. We showed how planning problems could be converted into large propositional satisfiability problems, and then solved using highly-optimized search procedures. In particular, we showed that WalkSat (Selman et al., 1994) could be used to solve certain hard planning problems that were beyond the capabilities of previous specialized planning systems. Closely related to our planning as satisfiability approach is a new class of planners based on highly efficient nonmonotonic reasoning systems. Although these systems are not all strictly propositional, they make use of algorithms that are quite similar to those for propositional reasoning: the DLV system (Eiter et al., 2000), the Smodels system (Niemela and Simons, 2000), and the "causal calculator" (Lifschitz et al., 2000).

In our planning as satisfiability approach (Kautz and Selman, 1996), a planning problem is converted into a set of axiom schemas. These schemas are instantiated for plans of a fixed given length. The instantiated formulas are solved by a SAT engine, and the plan can be read off of the satisfying model. If no model is found, the length parameter is incremented and the problem is re-instantiated, until it can be proven to be consistent. The first instantiation of our satisfiability-based planning approach was called SATPLAN. A more recent implementation of the system is called BlackBox (Kautz and Selman, 1999). BlackBox uses the Graphplan planning graph representation (Blum and Furst, 1995) as an intermediate format. The experiments in this paper were performed using the original SATPLAN system, because the integration of new axiom schemas is most straightforward in this system. However, it would not be difficult to add the same kinds of additional axioms into the BlackBox system (Huang et al., 1999).

The benchmark planning testbed used in both (Kautz and Selman, 1996) and this paper is described in Table 8.1. There are a series of problems from a logistics domain, that involve moving packages between locations in various cities by truck and by airplane. Non-conflicting actions in this domain may occur in parallel, and each action takes one time step. For example, the best known solution to the problem logistics.a contains 54 actions, but they are parallelized in such a manner that they can be executed over a span of 11 time steps. The largest logistics problem solved in our 1996 paper involved 10^{10} possible configurations of packages and vehicles. In this paper we have solved a much harder instance involving 10^{16} configurations. (This corresponds to a logistics problem with 9 packages, 5 trucks, 2 airplanes, and 15 locations.)

Note that the state-space of the SAT encoding of these instances is much larger than the number of configurations. For example, the largest problems in this paper contain about 2,000 Boolean variables, for a state-space of size 2^{2000} . It has been interesting to discover that we can efficiently search this

problem	soln length	soln size	configurations
log.a	11	54	10^{10}
log.b	13	47	10^8
log.c	13	63	10^{10}
log.d	14	74	10^{16}

bw.a	6	12	10^6
bw.b	9	18	10^9
bw.c	14	28	10^{13}
bw.d	18	36	10^{19}

Table 8.1 Planning benchmark testbed. For logistics problems, the solution length in terms of time steps is optimal, and the number of actions per solution is the smallest that has been found using any search procedure so far. For the blocks world (one arm), the solution length and sizes are optimal (each time step includes a pickup/putdown pair).

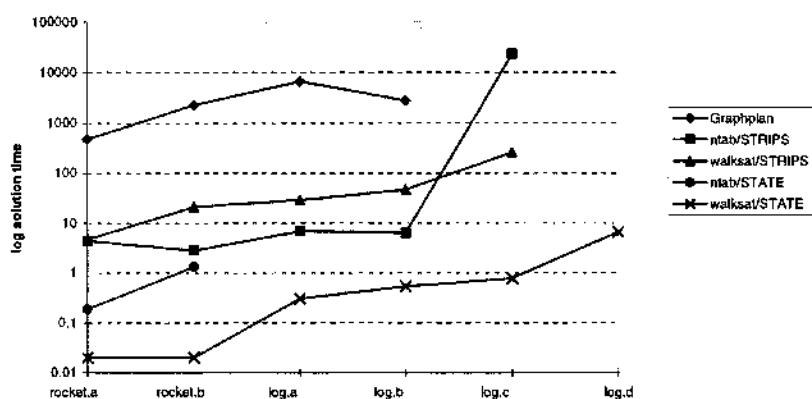


Figure 8.1 Comparison of search times on original logistics instances.

“exploded” state-space. We also worked on a series of traditional blocks world problems involving up to 19 blocks and 10^{19} configurations.

Figure 8.1 summarizes the main results from (Kautz and Selman, 1996). It compares the performance of the highly-efficient Graphplan system of Blum and Furst (1995) with the SATPLAN approach. The problems “rocket.a” and “rocket.b” are simpler versions of our logistics problems. The graph refers to the time required to search an instantiated data structure for a solution: in the case of Graphplan, a graph, and in the case of SATPLAN, a CNF formula. Times to instantiate and pre-process the problems for SATPLAN

ranged from 42 seconds (log.a) to 271 seconds (log.d). Because Graphplan does not separate out the instantiation and search times, we subtracted the SATPLAN instantiation times from the Graphplan numbers. (The Graphplan implementation in fact included a more efficient instantiation routine than did SATPLAN.)

The chart compares Graphplan with four variations of SATPLAN. The "ntab" variations employ an implementation (Crawford and Auton, 1993) of the classic Davis-Putnam-Loveland systematic search procedure, while the "WalkSat" variations use stochastic local search. Our specific version of local search is based on the GSAT / WalkSat algorithm (Selman et al., 1992; Selman et al., 1994). The "STRIPS" variations used SAT encodings automatically generated from a STRIPS-style problem specification (Fikes and Nilsson, 1971): the code to do this was based on a modified version of the Graphplan implementation. This conversion from STRIPS to propositional clauses can now be done more conveniently using our BlackBox system (Kautz and Selman, 1999). The "STATE" variations use a hand-crafted "state-based" encoding, as described in (Kautz et al., 1996). Table 8.2 describes the basic set of state-based axioms. Note that in this axiomatization no propositions that explicitly stand for actions are used, but only ones for fluents.

We see that in this domain the satisfiability-based planning approaches dominate. The best performance is obtained with stochastic local search and state-based encodings. Two caveats are in order. First, in other domains Graphplan or other planning systems can outperform SATPLAN; indeed, Graphplan is comparable with SATPLAN on the blocks world instances from the testbed. Second, the state-based encodings are quite different from STRIPS, and are not currently automatically generated from STRIPS operators. However, we have argued that state-based axioms are a natural and convenient way to specify many planning domains: there is no need to insist that STRIPS-notation is "primary". Furthermore, this paper will show that the general approach of specifying a problem domain by a set of axiomatic constraints on changes between states makes it straightforward to incorporate many different kinds of heuristic knowledge.

Given the computational difficulty of the planning problem, one might wonder why it has ever been possible to deploy traditional planning systems in real applications. The reason is that practical systems minimize search by using techniques such as domain-specific control rules (Bacchus and Kabanza, 1995; Bacchus and Kabanza, 2000; Doherty and Kvarnstrom, 1999; Nau et al., 1999; Carbonell et al., 1992; Penberthy and Weld, 1992; Sacerdoti, 1977; Slaney and Thiebaux, 1996; Veloso, 1992; Weld, 1994; Reiter, 1999), "compiled" reactive plans (Agre and Chapman, 1987; Williams and Nayak, 1997), and heuristics based on temporal and resource constraints (Vere, 1985; Muscettola, 1994). However, scaling remains problematic in many interesting domains.

All these techniques involve incorporating more domain specific knowledge into the planner. The natural question arises: can the power of knowledge also be incorporated into the planning as satisfiability framework, without sacrificing its elegance or generality?

Between adjacent time points i and $i + 1$:

- Packages stay at a location or are loaded into a vehicle.

$$at(pkg, loc, i) \supset at(pkg, loc, i + 1) \vee$$

$$\exists veh.in(pck, veh, i + 1)) \wedge at(veh, loc, i) \wedge at(veh, loc, i + 1)$$

- Packages stay in a vehicle or are unloaded.

$$in(pkg, veh, i) \supset in(pkg, veh, i + 1) \vee$$

$$\exists loc.at(pck, loc, i + 1)) \wedge at(veh, loc, i) \wedge at(veh, loc, i + 1)$$

- Trucks either stay in place or move to an adjacent location.

$$at(trk, loc1, i) \supset at(trk, loc1, i + 1) \vee$$

$$\exists loc2.adjacent(loc1, loc2) \wedge at(trk, loc2, i + 1)$$

For every time point i :

- Packages are at or in exactly one place.

$$\neg at(pkg, loc1, i) \vee \neg at(pkg, loc2, i)$$

$$\neg in(pkg, veh1, i) \vee \neg in(pkg, veh2, i)$$

$$\neg at(pkg, loc, i) \vee \neg in(pkg, veh, i)$$

$$at(pkg, loc1, i) \vee at(pkg, loc2, i) \vee \dots \vee in(pkg, vehk, i)$$

- Trucks are at one location.

$$\neg at(trk, loc1, i) \vee \neg at(trk, loc2, i)$$

$$at(trk, loc1, i) \vee \dots \vee at(trk, lock, i)$$

- Airplanes are at one airport.

$$\neg at(plane, airpt1, i) \vee \neg at(plane, airpt2, i)$$

$$at(plane, airpt1, i) \vee \dots \vee at(plane, airptk, i)$$

Table 8.2 Description of the basic axioms for the state-based encoding of logistics planning. Domain elements are time points, packages, locations (airports and non-airports), and vehicles (planes and trucks). Variables are implicitly typed (e.g. “pkg” is of type package) and are implicitly bound apart (e.g. “loc1” and “loc2” refer to different locations).

2 DOMAIN DEPENDENT PLANNING

We have seen that a satisfiability-based planning strategy can be quite effective in certain domains. One path to further improvement is to apply newer, faster satisfiability testing procedures. See (Kautz and Selman, 1999) for encouraging results for planning as satisfiability using new randomized versions of backtracking search algorithms (Gomes et al., 1998) based on lookahead (Li and Anbulagan, 1997) and dependency directed backtracking (Bayardo Jr. and Schrag, 1997). The other approach, which is the focus of this paper, is to encode more knowledge in the problem representations.

There are several kinds of knowledge that could be added to a problem representation. First, there is *knowledge about the domain itself*. For instance, a fact about the logistics domain is that "a truck is only in one location". Second, there is *knowledge about good plans*, for example, "do not return a package to a location from which it has been removed". Any plan violating this constraint is obviously suboptimal. Third, one could encode explicit control knowledge about search, such as "plan air routes before land routes".

Such information is traditionally incorporated in the planning algorithm itself or in a special programming language interpreted by the planner. Instead, we propose expressing domain knowledge as additional declarative axioms. Thus, a problem instance will be represented as the union of three sets of axioms: those for operators, those for the initial and goal states, and those for additional domain knowledge, which we will call "heuristic axioms". Domain information simply functions as a constraint on the search and solution spaces, and is independent of any search engine strategy. It is important to note that domain knowledge has exactly the same status as any of the other problem constraints. In particular, the domain knowledge is *not* a "meta-constraint" on the search algorithm, as is, for instance, the "cut" operator in logic programming.

Our approach is similar to that of Bacchus and Kabanza (1995), who also declaratively specify heuristic axioms for a planning system. It differs in that they used a detailed modal temporal logic specification to tightly control a simple forward-chaining planner, whereas we use weaker domain knowledge to assist general and powerful SAT engines. This paper builds on the observations of Ernst, Millstein, and Weld (1997) concerning the effect of domain-specific axioms on our original blocks-world testbed. They noted that including state-invariant axioms made the problems larger before simplification (as described below), but smaller and easier to solve after simplification.

3 THE LOGICAL STATUS OF HEURISTICS

The straightforward, logical representation of heuristic knowledge we have adopted allows us to determine the logical relationship between that knowledge and the original problem statement. It allows us to distinguish between heuristics that add entirely new constraints, and those that simply make *explicit* information that is implicit in other parts of the problem. In fact, the categories defined by the logical relationship of the heuristics to the rest of the problem

Invariant: A truck is at only one location

$$\begin{aligned} \textit{at}(truck, loc1, i) \wedge loc1 \neq loc2 \supset \\ \neg \textit{at}(truck, loc2, i) \end{aligned}$$

Optimality: Do not return a package to a location

$$\begin{aligned} \textit{at}(pkg, loc, i) \wedge \neg \textit{at}(pkg, loc, i + 1) \wedge i < j \supset \\ \neg \textit{at}(pkg, loc, j) \end{aligned}$$

Simplifying: Once a truck is loaded, it should immediately move

$$\begin{aligned} \neg \textit{in}(pkg, truck, i) \wedge \textit{in}(pkg, truck, i + 1) \wedge \\ \textit{at}(truck, loc, i + 1) \supset \\ \neg \textit{at}(truck, loc, i + 2) \end{aligned}$$

Table 8.3 Example axiomatic form of logistics heuristics.

instance create a natural classification of kinds of heuristics. Consider the following four classes:

1. Heuristics entailed by the operator axioms alone are those for *conflicts and derived effects*. For example, any two simultaneous *fly* actions for the same airplane conflict. For “pure” STRIPS operators, conflicts can be easily derived by simply comparing the precondition and delete lists of operators. When derived predicates and auxiliary axioms are included, however, determining action conflicts can become NP-complete. In such cases it can be helpful to add heuristic axioms that make the conflicts explicit.
2. Heuristics entailed by the operator and initial state axioms together include *state invariants*. For example, the typical STRIPS formulation of the operator for moving a vehicle from one location to another, *drive(truck, origin, dest)*, does not entail that a truck is always at a single location. However, if in the initial state every truck is at exactly one location, the operator will propagate that invariant to all future states.
3. Heuristics entailed by the operators and initial state axioms, together with the plan length n are a kind of *optimality condition*. Assuming that the given plan length is the minimum needed to solve the problem, then the fact that the plan contains no redundant or unnecessary steps logically follows. Optimality conditions can be hard to infer: as hard, in fact, as solving the problem instance, since the full set of optimality conditions would rule out all actions not in a solution. However, although difficult to automatically derive, at least some of the “obvious” optimality conditions are easy for a person to encode. The heuristic we noted earlier, “do not return a package to a location from which it has been removed”, is such an obvious condition.

4. Finally, new constraints on a problem instance that are not entailed by any of the previous axioms are *simplifying assumptions*. Since they are not entailed, they must actually rule out some of the valid solutions to the problem; thus, they introduce incompleteness into the heart of the planning system. However, computational considerations already make virtually all planners incomplete *in practice*, and in many cases one can prove the “lost” solutions never transform a solvable instance into an inconsistent one. In the logistics domain, the heuristic “once a truck is loaded, it should immediately move” is such a “safe” simplifying assumption, because any solution that violates it can be made into one that satisfies it by delaying *load* actions.

Examples of the axiomatic forms of some the heuristics used in the experiments described below appear in Table 8.3.

How can one go about developing heuristics for a problem domain? The traditional approach, and the one followed in this paper, is to simply employ introspection to capture both “obvious” inferences that are hard to deduce mechanically (McAllester, 1991) and simplifying assumptions that follow from abstracting the essence of the domain. It is clearly a matter of debate whether the introspective approach is feasible in the long run. The common complaint is that manually-created heuristics are hard to maintain as a complex system evolves over time. However, in previous work such heuristics are encoded either as procedures or as statements in a meta-language that directly controls the operation of a planning algorithm. By contrast, in our approach the heuristic axioms refer only to constraints on states and plans — not *how* plans are found. Thus, they may prove to be no harder to write or maintain than any other part of the domain specification.

A different (complementary) approach is to automatically generate heuristics. There has been a great deal of work on the problem of heuristic generation, including that on explanation-based learning (Minton, 1988; Kambhampati et al., 1996), static analysis of operator schemas (Smith, 1989; Etzioni, 1993), problem abstraction (Knoblock, 1994), and operator-graph analysis (Smith and Peot, 1996). All of this work has been aimed at producing explicit search control rules: for example, rules that would state when to prefer one operator application over another, or when to cut off a branch of recursive search. In our framework, however, we instead wish to generate constraints on admissible plans and sequences of events, in order to reduce and simplify the search space. It would be interesting to investigate how to modify techniques such as explanation-based learning so that the output is a set of such constraints.

There is also a relationship between heuristic generation and what has been called “theory compilation” (Selman and Kautz, 1996). For example, one could fix the operators and the goal state, generate a number of consequences, and add them back to the original theory. In the worse case, the compiled theory is exponentially large, but the general approach can be efficient in practice (Williams and Nayak, 1997). Another area that investigates potentially useful techniques is that of simplification of search problems by the discovery of “symmetries”. For example, (Joslin and Roy, 1997) adds “symmetry breaking predicates” to a theory based on the discovery of symmetries in operator

schema; in our vocabulary, this would be a case of a “safe simplifying assumption”. Finally, any system that automatically generates heuristics must amortize the cost of that generation over some set of problem instances. Heuristics such as type (1) above could be generated by analysis of the operators alone, and thus amortized over all problem instances in that domain. Heuristics based on both the operators and some information about the specific problem instance, such as types (2) and (3), could only be amortized over a subset of the domain.

4 EXPERIMENTS

We designed a series of experiments to test the feasibility of specifying declarative domain knowledge. It is not obvious that it would work: the heuristics might simply blow up the size of the problem with redundant axioms, and simply make the problem harder to solve.

The first series of experiments were based on the blocks world, where there is a single arm and no parallel actions. The number of “move” actions (a move is equivalent to a pickup and putdown pair) in the optimal solutions ranged from 6 to 19. We created three versions of each problem instance: The first included axioms for the move operator only. These instances are labeled bw.a, bw.b, bw.c, and bw.d. The second added axioms for state invariants of the predicate “on”; for example, that at most one block could be on another. These examples are labeled “bw.a.state”, etc. The third version also included optimality heuristics, namely: never move a block twice in a row; once a block is moved onto a block (not the table) it stays there; and never move a block more than twice. These examples are labeled “bw.a.opt”, etc. Because there is only a single predicate “on” and no parallel actions, there is no need for the class of heuristics for conflicts and derived effects. The blocks world problems that appear in (Kautz and Selman, 1996) are the “state” versions of this paper. These experiments expand on the ones performed in (Ernst et al., 1997), which considered only the “move” and “state” versions.

Each version of each problem was instantiated, simplified by unit propagation, and then solved by the “ntab” implementation of the Davis-Putnam-Loveland procedure (Crawford and Auton, 1993), and by WalkSat. The number of time steps was set to the smallest (optimal) value. Because WalkSat is a randomized procedure, we took the average timings for 100 runs with different random seeds. (Simplification by unit propagation can be done in linear time, and for problems of this size takes less than one second with a good C implementation; for convenience we used a simple shell-script that required about 20 seconds.) All times are on a 200 Mhz SGI Challenge.

The results are summarized in Table 8.4. First, let us consider the size of the problems. We see that before the formulas were simplified (“orig. no. vars/clauses”), adding the heuristic axioms increased the number of variables by about 10%, and increased the number of clauses by 300% — 600%. For example, bw.d contained 62,734 clauses, but bw.d.opt contained 388,755 clauses. However, simplification by unit propagation made the heuristic instances significantly smaller in the number of distinct variables. For the

wff	orig no. vars	orig no. clauses	final no. vars	final no. clauses	walksat time	ntab time
bw.a	804	5118	534	3060	1.84	0.26
bw.a.state	973	14582	459	4710	0.03	0.23
bw.a.opt	946	16391	294	2448	0.55	0.12
bw.b	1635	11091	1235	7457	13.08	5.13
bw.b.state	1865	35971	1087	13845	2.22	0.66
bw.b.opt	1876	42230	564	5759	0.75	0.38
bw.c	4258	30986	3526	22535		
bw.c.state	4723	126866	3016	50621	3.84	16.5
bw.c.opt	4738	155741	1225	17082	77	25.7
bw.d	8282	62734	7132	47098		
bw.d.state	9023	311862	6325	132257	688	
bw.d.opt	9042	388755	2174	40384	643	2651

Table 8.4 Effect of heuristic axioms on blocks world problems. Times in seconds. No entry means procedure failed to solve problem after more than 48 hours.

simplified versions of bw.d, the number of variables dropped from 7,132 (no heuristics) to 6,325 (state), and dramatically to 2,174 (opt). The final number of clauses was *highest* for the “state” versions, and *smallest* for the “opt” versions.

Turning to the timings, we see that our stochastic algorithm WalkSat was fastest on the “state” version, except for bw.b, where the “opt” version was slightly easier to solve. On the other hand, the “opt” versions were significantly easier to solve for the systematic algorithm ntab, except for bw.c, where “state” was slightly better, and bw.d, where for WalkSat only “state” was comparable. Problems bw.c and bw.d could not be solved in less than 48 hours by either routine without the addition of the heuristic axioms.

These results demonstrated that the general approach was sound, and could extend the SATPLAN approach to planning to classes of larger and harder problems. They also suggested that decreasing the number of variables after simplification was more important than decreasing the number of clauses, and that the point of diminishing returns from the addition of axioms would be sooner reached for stochastic search than for systematic search.

The second series of experiments were based on the logistics domain mentioned earlier. The first three instances also appeared in (Kautz and Selman, 1996), but for this paper we also created a much harder instance, logistics.d. As described in (Kautz and Selman, 1996), our axiomatization is “state based” in that it represents invariants on states and between pairs of

adjacent states, but does not explicitly represent the action operators. The actions in a plan can be derived in linear time from a satisfying model of the problem instance. Because there are no operators represented, there can be no additional axioms for conflict heuristics. Similarly, there can be no additional state invariant heuristics, because such invariants are already part of the representation. Therefore we studied the effect of optimality heuristics and simplifying assumptions in this domain.

There were a number of questions we wished to investigate:

- Are simplifying assumptions and optimality conditions useful heuristics for stochastic search? In the blocks world, the optimality heuristics helped systematic search, but not stochastic search. Furthermore, because stochastic search is particularly good on under-constrained problems, simplifying assumptions might hurt WalkSat rather than help.
- Would the overall positive impact of the additional axioms be greater on systematic or stochastic methods?
- Is problem size in terms of number of variables the more important factor, as in the blocks world? Would the optimal level of heuristics again differ for the two algorithms?

The question of the impact of the heuristics on systematic versus stochastic search was particularly intriguing, because there are intuitive grounds for either answer. One could argue that because the additional axioms enable more unit propagations during search, and systematic methods handle unit propagation more efficiently, the heuristics would be more beneficial to systematic search. Alternatively, one could argue that the additional axioms would *shorten* chains of unit propagation, and therefore be more helpful to stochastic algorithms, which handle long chains more slowly.

We began this set of experiments by writing down four sets of heuristics. We then performed a few preliminary experiments to see which set of heuristics, taken alone, usually provided the greatest improvement in solution time. We then ordered the heuristics from most useful to least useful. Finally, we made each set properly include all the previous (more useful) heuristics. The result gives five versions of each problem instance, labeled “none” (no extra axioms), h1, h2, h3, and h4. Each version is a strict superset of the clauses in the previous version.

The axioms that first appear in each set are:

h1 (optimality condition): Once a package leaves a location, it never returns to that location.

h2 (simplifying assumption): A package is never in any city other than its origin or destination cities. (This is not an optimality condition because it rules out solutions where packages are transferred between airplanes in an intermediate city.)

h3 (simplifying assumption): Once a vehicle is loaded, it should immediately move.

	log.a	log.b	log.c	log.d
base	828	843	1,141	2,160
h1	825	843	1,141	2,160
h2	666	603	781	1,232
h3	666	603	781	1,232
h4	666	603	781	1,232

Table 8.5 Number of variables in logistics formulas after simplification.

	log.a	log.b	log.c	log.d
base	6,718	7,301	10,719	27,242
h1	8,684	9,431	14,051	84,779
h2	5,933	4,871	7,083	11,683
h3	6,769	5,576	8,118	13,407
h4	7,856	6,551	9,645	17,075

Table 8.6 Number of clauses in logistics formulas after simplification.

	log.a	log.b	log.c	log.d
base	0.31	0.55	0.79	6.68
h1	0.14	0.12	0.21	1.97
h2	0.1	0.11	0.13	1.05
h3	0.27	0.08	0.20	2.30
h4	0.14	0.14	0.12	3.66

Table 8.7 Solution times in seconds for walksat on logistics problems with different sets of heuristic axioms.

h4 (optimality condition): A package never leaves its destination city.

Again the problems were instantiated, simplified, and solved by both WalkSat and ntab. Tables 8.5–8.8 summarize the results of the experiments.

	log.a	log.b	log.c
base			
h1	4,680	140,820	582
h2	655	3.6	468
h3	696	5.41	1,220
h4	106	10.7	38,520

Table 8.8 Solution times in seconds for ntab on logistics problems with different sets of heuristic axioms. A missing entry means a solution was not found after 48 hours.

We first consider the size of the formulas. Tables 8.5 and 8.6 show the number of variables and the number of clauses after simplification for each of the versions of problems log.a, log.b, and log.c.

Only the simplifying assumption h2 decreased the number of variables. This is because it rules out all propositions that represent a package being at a location that is not in the origin or destination cities. Considering the number of clauses after simplification, we see that h1 increased the size of the formula by about 30%, but when h2 is added, the formula is about 30% *smaller* than the original one. Increasing the heuristic set to h3 and h4 caused a gradual increase in the number of clauses. Thus, h2 represents a point at which both the number of variables and the number of clauses is minimized.

Table 8.7 is based on the timings from WalkSat. For log.a, log.c, and log.d the h2 variants were fastest to solve, and for log.b the h3 variant was fastest. We see that the greatest decrease in solution time came from the addition of h1; recall that h1 *increased* the number of clauses and did *not* decrease the number of variables. There was a further slight decrease at h2, where the problem size was minimized. However, the decrease at h1 over “none” indicates that problem size was not the only factor involved in making the problems easier to solve. Set h2 was the point of diminishing returns for WalkSat. There was a slight increase at h3, and either a slight increase or decrease at h4.

The timings for systematic search in Table 8.8 are more dramatic. None of the problems could be solved without the additional axioms, nor could any version of log.d. For the smallest problem, log.a, the fastest time was realized at h4, but log.b and log.c were solved most quickly at h2. Again the most important heuristic was the optimality condition h1, because it made unsolvable problems solvable. However, the simplifying assumption h2 was also important, particularly for log.b, where it reduced the solution time by a factor of 38,000. (In absolute terms, from about 39 hours to about 3 seconds.)

Let us now consider the questions we raised earlier. First, it is clear that simplifying assumptions as well as optimality conditions can enhance the power of stochastic search as well as systematic search. We saw a 6-fold improvement for WalkSat on the largest, hardest problem instances. At least on this

benchmark set, the worry that reducing the number of solutions would make the problems harder for local search was unfounded.

Second, the greatest impact of the additional axioms was again for the systematic search engine. We saw that for ntab problems that could not be solved in two days could then be solved in a few seconds. As described below, our future work will examine why this is the case. As we have mentioned, our current hypothesis is the additional axioms greatly increase the number of unit propagations at each branch-point.

Third, both algorithms were usually optimal on the variations of the problems that minimized both the number of variables and the number of clauses. However, problem size was not the only or most important factor in determining the difficulty of solution. This is notable in the h1 versions, which were strictly larger than the versions without any heuristic axioms, but were much easier to solve.

For both algorithms the point of diminishing returns was reached at h2. This may have been because few optimally-short plans would violate h3, and so it had little impact. It also appears that h4 follows with a fairly short proof from h1 and h2. It is an open question whether there are other natural heuristics that are more powerful than h2.

5 CONCLUSIONS AND FUTURE WORK

This paper reports on a promising new method of enhancing the planning as satisfiability framework with domain-specific knowledge. We have shown that it is easy to encode such knowledge in a purely declarative manner that is independent of the kind of algorithm used to search the space. The approach appears to be the key to the next order of magnitude scaling of state-space planning algorithms. Problems that could not be solved in days could be solved in seconds with the additional axiomatic knowledge.

Our framework makes clear the logical status of different kinds of heuristics, and in particular distinguishes invariants, optimality conditions, and simplifying assumptions. The role of the optimality conditions deserves further discussion. We observed that setting the length of the plan to the minimum value that keeps it consistent places a *global* optimality condition on the problem. This global optimality condition logically entails many *local* rules about the construction of the solution: for example, that the solution not contain such locally inefficient actions such as moving a block twice in a row. However, it can be quite difficult to deduce all the relevant local optimality conditions for a particular problem. The optimality heuristics we have described simply make *explicit* some of these local rules. Thus, less needs to be deduced by combinatorial search. In other words, it can be easier to satisfy a global constraint with the “advice” given by a subset of the local consequences of that constraint.

Finally, we have seen that on the hardest problems we examined local search continues to outperform systematic search, just as in (Kautz and Selman, 1996). However, the use of heuristic axioms does make systematic methods more competitive. The latest version of the SATPLAN system, called BlackBox

(Kautz and Selman, 1999), includes powerful new randomized systematic SAT solvers that meet or exceed the performance of WalkSat for some domains.

There are many avenues for future exploration. One task is to see exactly how the shape of the search space is changed by different kinds of heuristics. Frank et al. (1997) present a detailed picture of the search space for random 3SAT. We would like to create such an analysis of the space generated by a range of planning domains under different sets of heuristics.

Another task is to examine the tradeoff between declarative control knowledge and solution quality. As discussed above, simplifying assumptions can reduce number of solutions to a planning problem, and in the worst case can rule out all of the shortest (optimal) solutions. The heuristic axioms considered in this paper are all admissible in the sense that this can never occur. By contrast, the stronger control axioms used by (Bacchus and Kabanza, 1995) make planning very fast but are clearly non-admissible and lead to lower-quality solutions. Recently we have begun to experiment with the range of tradeoffs, and have found that we can obtain further speedups with stronger heuristic axioms while still obtain "good" (if non-optimal) solutions (Huang et al., 1999).

Perhaps the most important task is to devise ways of automatically learning or inferring control axioms from a basic action encoding (such as STRIPS). For some initial results on learning control rules in a planning as satisfiability approach, see (Huang et al., 2000). In addition to the older work on explanation-based learning cited above, there have been recent breakthroughs in practical algorithms for inferring state invariants by (Gerevini and Schubert, 1998) and (Fox and Long, 1998).

Acknowledgments

We thank Jim Hendler, Jack Minker, Martha Pollack, Dan Weld, and the anonymous reviewers for their valuable comments on this paper.

References

- Agre, P. and Chapman, D. (1987). Pengi: an implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 268–272. AAAI Press.
- Bacchus, F. and Kabanza, F. (1995). Using temporal logic to control search in a forward-chaining planner. In *Proceedings of the Third European Workshop on Planning*, pages 157–169. AAAI Press.
- Bacchus, F. and Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1–2):123–191.
- Bayardo Jr., R. and Schrag, R. (1997). Using CSP look-back techniques to solve real world SAT instances. In *Proc. of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 203–208. AAAI Press.
- Blum, A. and Furst, M. (1995). Fast planning through planning graph analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1636–1642. Morgan Kaufmann.

- Carbonell, J., J., B., Etzioni, O., Gil, Y., Joseph, R., Kahn, D., Knoblock, C., Minton, S., Perez, A., Reilly, S., Veloso, M., and Wang, X. (1992). *Prodigy 4.0: the Manual and Tutorial*. CMU, Pittsburgh, PA, cmu-cs-92-150 edition.
- Crawford, J. and Auton, L. (1993). Experimental results on the cross-over point in satisfiability problems. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 21–27. AAAI Press.
- Doherty, P. and Kvarnstrom, J. (1999). TALplanner: An Empirical Investigation of a Temporal Logic-based Forward Chaining Planner. In *Proceedings of the 6th International Workshop on Temporal Representation and Reasoning (TIME'99)*, pages 30–35.
- Eiter, T., Faber, W., Leone, N., and Pfeifer, G. (2000). Declarative problem-solving using the dlv system. In Minker, J., editor, *Logic-Based Artificial Intelligence*, pages 79–103. Kluwer Academic Publishers, Norwell, Massachusetts, 02061.
- Ernst, M., Millstein, T., and Weld, D. (1997). Automatic SAT-compilation of planning problems. In *Proc. of the Fourteenth Int. Joint Conf. on Artificial Intelligence (IJCAI-97)*, pages 1169–1176. Morgan Kaufmann.
- Etzioni, O. (1993). Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62(2):255–302.
- Fikes, R. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.
- Fox, M. and Long, D. (1998). The automatic inference of static invariants in tim. *Journal of AI Research*, 9:317–371.
- Frank, J., Cheeseman, P., and Stutz, J. (1997). When gravity fails: Local search topology. *Journal of AI Research*, 7:249–281.
- Gerevini, A. and Schubert, L. (1998). Inferring state constraints for domain-independent planning. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 905–912. AAAI Press.
- Gomes, C., Selman, B., and Kautz, H. (1998). Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 431–437. AAAI Press.
- Huang, Y.-C., Selman, B., and Kautz, H. (1999). Control knowledge in planning: Benefits and tradeoffs. In *Proc. of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 511–517. AAAI Press.
- Huang, Y.-C., Selman, B., and Kautz, H. (2000). Learning declarative control rules for constraint-based planning. In Langley, P., editor, *Proceedings of the Seventeenth International Conference on Machine Learning (ICML-2000)*, pages 415–422. Morgan Kaufmann.
- Joslin, D. and Roy, A. (1997). Exploiting symmetries in lifted CSPs. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 197–202, Providence, RI. AAAI Press.
- Kambhampati, S., Katukam, S., and Qu, Y. (1996). Failure driven dynamic search control for partial order planners: an explanation based approach. *Artificial Intelligence*, 88(1–2):253–315.
- Kautz, H., McAllester, D., and Selman, B. (1996). Encoding plans in propositional logic. In Arelllo, L., Doyle, J., and Shapiro, S., editors,

- Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning (KR-96)*, pages 374–385. Morgan Kaufmann.
- Kautz, H. and Selman, B. (1996). Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1194–1201. AAAI Press.
- Kautz, H. and Selman, B. (1999). Unifying SAT-based and Graph-based planning. In Dean, T., editor, *Proc. of the Sixteenth Int. Joint Conf. on Artificial Intelligence (IJCAI-99)*, pages 318–325. Morgan Kaufmann.
- Knoblock, C. (1994). Automatically generating abstractions for planning. *Artificial Intelligence*, 68(2):243–302.
- Li, C. M. and Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. In *Proc. of the Fifteenth Int. Joint Conf. on Artificial Intelligence (IJCAI-97)*, pages 366–371. Morgan Kaufmann.
- Lifschitz, V., McCain, N., Ramolino, E., and Tachella, A. (2000). Getting to the airport: The oldest planning problem in AI. In Minker, J., editor, *Logic-Based Artificial Intelligence*, pages 147–165. Kluwer Academic Publishers, Norwell, Massachusetts) 02061.
- McAllester, D. (1991). Observations on cognitive judgements. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 910–915. AAAI Press.
- Minton, S. (1988). Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 564–569. AAAI Press.
- Muscettola, N. (1994). On the utility of bottleneck reasoning for scheduling. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 1105–1110. AAAI Press.
- Nau, D., Cao, Y., Lotem, A., and Munoz-Avila, H. (1999). SHOP: Simple Hierarchical Ordered Planner. In Dean, T., editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 968–973. Morgan Kaufmann.
- Niemela, I. and Simons, P. (2000). Extending the smodels system with cardinality and weight constraints. In Minker, J., editor, *Logic-Based Artificial Intelligence*, pagex 491–521. Kluwer Academic Publishers.
- Penberthy, J. and Weld, D. (1992). Ucpop: a sound, complete, partial order planner for adl. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning (KR-92)*, pages 103–114. Morgan Kaufmann.
- Reiter, R. (1999). Knowledge in action: Logical foundations for describing and implementing dynamical systems. Unpublished draft, available at <http://www.cs.utoronto.ca/~cogrobo/>.
- Sacerdoti, E. D. (1977). *A Structure for Plans and Behavior*. Elsevier, NY.
- Selman, B. and Kautz, H. (1996). Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224.
- Selman, B., Kautz, H., and Cohen, B. (1994). Noise strategies for local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 337–343. AAAI Press.

- Selman, B., Levesque, H., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446. AAAI Press.
- Slaney, J. and Thiebaux, S. (1996). Linear time near-optimal planning in the blocks world. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1208–1214. AAAI Press.
- Smith, D. (1989). Controlling backward inference. *Artificial Intelligence*, 39:145–208.
- Smith, D. and Peot, M. (1996). Suspending recursion in causal link planning. In *Proceedings of the Third International Conference on AI Planning Systems (AIPS-96)*, pages 182–190. AAAI Press.
- Veloso, M. (1992). Learning by analogical reasoning in general problem solving. Technical Report CMU-CS-92-174, CMU, Pittsburgh, PA.
- Vere, S. (1985). Temporal scope of assertions and window cutoff. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 1055–1059. Morgan Kaufmann.
- Weld, D. (1994). An introduction to least commitment planning. *AI Magazine*, 14(4):27–60.
- Williams, B. and Nayak, P. (1997). A reactive planner for a model-based executive. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1178–1185. Morgan Kaufmann.

Chapter 9

FUNCTIONAL STRIPS: A MORE FLEXIBLE LANGUAGE FOR PLANNING AND PROBLEM SOLVING

Héctor Geffner

*Departamento de Computación
Universidad Simón Bolívar
Aptdo. 89000, Caracas 1080-A
Venezuela
hector@usb.ve*

Abstract Effective planning requires good modeling languages and good algorithms. The Strips language has shaped most of the work in planning since the early 70's due to its effective solution of the frame problem and its support for divide-and-conquer strategies. In recent years, however, planning strategies not based on divide-and-conquer and work on theories of actions suggest that alternative languages can make modeling and planning easier. With this goal in mind, we have developed Functional Strips, a language that adds first-class function symbols to Strips providing additional flexibility in the codification of planning problems. This extension is orthogonal and complementary to extensions accommodated in other languages such as conditional effects, quantification, negation, etc. Function symbols, unlike relational symbols, can be nested so objects need not be referred to by their explicit names and as a result more efficient encodings can be provided. For example, a problem like the 8-puzzle can be codified in terms of four actions with no arguments; Hanoi, can be codified with a number of ground actions independent of the number of disks; resources and constraints can be easily represented, etc.

Functional Strips is both an action and a planning language in the sense that actions are understood declaratively in terms of a state-based semantics and operationally in terms of efficient updates on state representations. In this paper, we present the language, the semantics and a number of examples, and discuss possible uses in planning and problem solving.

Keywords: Action languages, Strips, planning, modeling and problem solving

1 INTRODUCTION

The Strips language introduced by Fikes and Nilsson in 1971 has shaped most the work in Planning (Fikes and Nilsson, 1971). The appeal of Strips can be explained by two main reasons: first, it provides a compact representation of actions that avoids the frame problem, and second, it supports divide-and-conquer strategies that have been regarded as good for planning. Strips, however, is limited in several ways, and more recent work on action representation (Pednault, 1989; Reiter, 1991; Gelfond and Lifschitz, 1993; Sandewall, 1994; Shanahan, 1997) and planning algorithms (e.g., (Kautz and Selman, 1999; Bonet and Geffner, 1999)) suggests that other languages could facilitate both modeling and computation.

A number of extensions of the basic Strips language, including negation, conditional effects, state variables, and quantification have been considered in a number of proposals (e.g., (Wilkins, 1988; Pednault, 1989; McDermott, 1998b)). These extensions simplify the modeling task and in certain cases make plans shorter (Nebel, 1998). These planning languages, however, as well as the closely related action languages developed in the area of reasoning about change (in particular, (Gelfond and Lifschitz, 1993; Giunchiglia et al., 1997)) share with Strips an important restriction: the symbols that can be used to represent fluents are either constant or relational symbols (such as `on(a,b)` or `fluid_level`), but not function symbols. Functional symbols are excluded or they are accommodated with restrictions (e.g., they cannot appear nested in the head of action rules). Functional relations, however, are important in planning, and indeed, the recent PDDL language standard (McDermott, 1998b) makes room for terms involving non-boolean fluents, but largely in an ad-hoc fashion.

The goal of this paper is to extend Strips with first-class function symbols, generalizing and making explicit the logic underlying Strips-like languages.¹ In particular, we try to clarify the distinction between *state* and *state representations*, two notions that are often collapsed in planning. States are *logical interpretations* over the language providing a denotation to all constant, function, and relational symbols. States *representations* on the other hand, are encodings of those interpretations whose form depends on the language. In the case of Strips, states can be represented by sets of atoms, while in other languages, they can be represented by sets of literals or suitable assignments. In all cases, the representations encode interpretations that determine the denotation of all relevant expressions in the language.

One of the main advantages of a functional language over a purely relational one is that functions can be nested and thus can refer to objects without providing their explicit names. As a result, the number of action arguments and the number of possible *ground actions* can be reduced substantially, something that is crucial in modern planners (Blum and Furst, 1995; Kautz and Selman, 1999; Bonet and Geffner, 1999). For example, the 8-puzzle can be modeled in Functional Strips with 4 ground actions only, Hanoi can be modeled with a

¹For an earlier logical account of Strips with a different scope, see (Lifschitz, 1986).

number of ground actions that is independent of the number of disks, and so on. In other problems, like Gripper (McDermott, 1998a), the new language allows for entirely different formulations that exploit the high degree of symmetry in the domain (Fox and Long, 1999). In many cases, the representations obtained are as efficient as the representations used in specialized programs, something that is necessary, although not sufficient, for planning approaches to be competitive with specialized methods.

Two additional comments before we proceed. Once function symbols become 'first-class citizens', the number of possible atoms in the language becomes infinite. This does not mean that state representations become infinite as well. Indeed, the *representation* of states, while different from Strips, remains finite and compact as a consequence of the assumption that the domain of interpretation is given by a *finite* set of objects. At the same time, the *computation* of the representation of the next state remains efficient as well.

The second issue is that while functions in Strips help reduce the number of possible *ground actions*, they do not and cannot affect the *branching factor* of the problem; namely, the number of actions that are applicable in each state. Still, it's worth emphasizing that the performance of modern planners including Graphplan, SAT, and heuristic-based planners (Blum and Furst, 1995; Kautz and Selman, 1999; Bonet and Geffner, 1999) is influenced *both* by the branching factor of the problem *and* the number of possible ground actions. Even planners that rely on user-supplied control knowledge such as TLPLAN (Bacchus and Kabanza, 2000), are affected by the number of ground actions as they all must be tested for applicability in each state. In a Strips encoding of a problem like Hanoi, there may be up to N^3 ground actions, where N is the number of disks, but at most only 3 of these actions are applicable in each state. In Functional Strips, the number of ground actions for this problem can be reduced to 6, a number of actions that is closer to the branching factor of the problem and can more efficiently be tested at run time.²

The rest of the paper is organized as follows. We review Strips and State Models (Section 2), show how to accommodate functions in Strips (Section 3), and illustrate the resulting language over a number of examples (Sect. 4). We also illustrate the importance of choosing suitable representations by considering the so-called Gripper domain (Section 5). We then discuss possible uses of Functional Strips in actual planning systems (Section 6), and briefly consider a number of additional extensions and related work (Section 7).

²Models that reduce the number of possible ground actions towards the limit represented by the branching factor of the problem are likely to be more suitable for *learning* as well. Indeed, learning to apply actions that take few arguments is likely to be simpler than learning to apply actions that take many arguments. This is true for example in (Khordon, 1999), where general action strategies are learned for general domains expressed in Strips.

2 STRIPS

2.1 LANGUAGE

The Strips language comprises two parts: a language for describing the world and a language for describing how the world changes. The first is called the *state language* or simply the language, and the second, is called the *operator language*. We consider the Strips language as used currently in planning (c.g., the Strips subset of PDDL (McDermott, 1998b)), rather the original version of Strips that is slightly more complex (Fikes and Nilsson, 1971; Lifschitz, 1986).

The Strips language \mathcal{L}_S is made up of two types of symbols: relational and constant symbols. In the expression $on(a, b)$, on is a relational symbol of arity 2, and a and b are constant symbols. We refer to the (finite) sets of relational and constant symbols as \mathcal{R} and \mathcal{C} respectively. In the Strips language, there are no functional symbols and the constant symbols are the only *terms*. The *atoms* are defined in a standard way from the combination $p(t_1, \dots, t_k)$ of a relational symbol p and a tuple of terms t_i of the same arity as p . Similarly the Strips *formulas* are obtained by closing the set of atoms under the standard propositional connectives. In Strips, only conjunctions are used and they are identified with sets of atoms.

A main difference between relational and constant symbols in Strips is that the former are used to keep track of aspects of the world that may change as a result of the actions (e.g., the symbol *on* in $on(a, b)$), while the latter are used to refer to objects in the domain (e.g., the symbols *a* and *b* in $on(a, b)$). More precisely, actions in Strips affect the denotation of relational symbols but not the denotation of constant symbols. For this reason, the former are said to be *fluent symbols*, and the latter, *fixed symbols* or *constants*.

The *operators* are defined over the set of atoms \mathcal{A} in \mathcal{L}_S . Each operator op has a precondition, add, and delete lists $Prec(op)$, $Add(op)$, and $Del(op)$ given by sets of atoms. Operators are normally defined by means of *schemas*; here we assume that such schemas have been grounded.

A Strips planning problem $P = (\mathcal{L}_S, \mathcal{O}_S, \mathcal{I}_S, \mathcal{G}_S)$ consists of a tuple where \mathcal{L}_S stands for the state language (defined by the constant and relational symbols), \mathcal{O}_S is the set of operators defined over the atoms in \mathcal{L}_S , and \mathcal{I}_S and \mathcal{G}_S are sets of atoms defining the *initial* and *goal* situations.

2.2 STATE MODELS

The meaning of a Strips planning problem (as well as the meaning of problems expressed in many extensions of Strips) can be given in terms of *state-space models* (Newell and Simon, 1972; Nilsson, 1980; Pearl, 1983). A state model is a tuple $\langle S, s_0, S_G, A, next \rangle$ where

1. S is a finite set of states
2. $s_0 \in S$ is the initial state
3. $S_G \subseteq S$ is a non-empty set of goal states
4. $A(s)$ is the set of actions a applicable in state s , and

5. *next* is a transition function that maps a state s into a successor state $s_a = \text{next}(a, s)$ for any action $a \in A(s)$

A *solution* of a state-state model is a finite sequence of applicable actions a_0, a_1, \dots, a_n that maps the initial state s_0 into a goal state $s \in S_G$. That is, the action sequence a_0, a_1, \dots, a_n must generate a sequence of states s_i , $i = 0, \dots, n + 1$, such that $s_{i+1} = f(a_i, s_i)$, $a_i \in A(s_i)$, and $s_{n+1} \in S_G$. We say that two state spaces are *equivalent* when they have the same solutions. Often one is interested in solutions that are optimal in some sense; e.g., that minimize the number of actions. We'll say more about this in Section 7.

2.3 STRIPS STATE MODEL

A Strips planning problem $P = \langle \mathcal{L}_S, \mathcal{O}_S, \mathcal{I}_S, \mathcal{G}_S \rangle$ is given a precise meaning by mapping it into the state model $\mathcal{S}(P) = \langle S, s_0, S_G, A, \text{next} \rangle$ where

- A1. the states s are sets of atoms from \mathcal{L}_S
- A2. the initial state s_0 is \mathcal{I}_S
- A3. the goal states are the states s such that $\mathcal{G}_S \subseteq s$
- A4. $A(s)$ is the subset of operators $op \in \mathcal{O}_S$ such that $\text{Prec}(op) \subseteq s$
- A5. the transition function *next* is such that $\text{next}(a, s) = s + \text{Add}(a) - \text{Del}(a)$, for $a \in A(s)$

The solution of the planning problem P is the solution of the state model $\mathcal{S}(P)$; namely, a sequence of applicable actions that maps the initial state into a goal state in $\mathcal{S}(P)$.

For extensions of Strips such as those involving negated literals or conditional effects, slightly different state models are needed (Gelfond and Lifschitz, 1998; Nebel, 1998). These different models, however, have something in common: they are all *propositional* in the sense that the internal structure of atoms can be ignored. In particular, different atoms can be substituted by different propositional symbols, resulting in state models that are equivalent. For Functional Strips, this is no longer the case as different terms may denote the same object. For constructing a state-model for Functional Strips, we will thus have to consider the internal structure of atoms, and hence will find convenient to distinguish two notions that are collapsed in the model [A1]–[A5]: the notion of *state* and the notion of *state representation*. We will associate the former with the *logical interpretations* over the state language (Gelfond and Lifschitz, 1993; Sandewall, 1994) and the latter with suitable encodings of them. For the basic Strips language, states are conveniently *represented* by a set of atoms, but for other languages, including Functional Strips, states are represented in a different way. To make the transition from Strips to Functional Strips simpler, we will thus first reformulate the state model [A1]–[A5] making this distinction explicit.

2.4 NON-PROPOSITIONAL STRIPS MODEL

An *interpretation* s is a mapping that assigns a denotation x^s to each symbol, term, and formula x in the language. In Strips, the symbols are constant

or relational symbols. Constant symbols play the role of *object names*, with different names referring to different objects. We assume a finite set \mathcal{C} of such constant symbols n and a finite domain of interpretation D such that different names denote different objects and all objects have names. Furthermore, we consider a class of interpretations in which the denotation of names, as opposed to the denotation of fluents, is *fixed*. That is, if we write n^* to refer to the denotation of names $n \in \mathcal{C}$, we consider only interpretations s for which $n^s = n^*$ for all $n \in \mathcal{C}$. We call the denotation function $* : \mathcal{C} \mapsto D$ that establishes a 1-to-1 correspondence between names and objects, the *representation function for constants*. The choice of this representation function is arbitrary in Strips as they all lead to state models that are equivalent (i.e., that have the same solutions).

The denotation p^s of relational symbols p of arity k is a subset of D^k . The denotation of relational symbols, unlike the denotation of constant symbols, can change as a result of the actions. Relational symbols are thus *fluent symbols*, while constant symbols are *fixed*. The distinction between fluent and fixed symbols is *semantic*, and one could think of languages where relational symbols are fixed and constant symbols are fluents. Languages with *fluent ‘constant’ symbols* such as *level_of_fuel* are used in a number of planners (Jonsson and Bäckstrom, 1994; Currie and Tate, 1991; Wilkins, 1988; Koehler, 1998; Laborie and Ghallab, 1995; Penberthy and Weld, 1994) and action languages (Sandewall, 1994; Giunchiglia et al., 1997), where they are referred to as *state-variables, resources, or features*.

The denotation $[p(t)]^s$ of an atom $p(t)$, where p is a relational symbol in \mathcal{P} and t is a tuple of terms of the same arity as p , is **true** if $t^s \in p^s$ and **false** otherwise. In Strips, the finite set of object names $n \in \mathcal{C}$ are the only terms, and hence the resulting set of atoms is finite.

The interpretations s that result from a given representation function for constants can be encoded by the set $[s]$ of atoms $p(t)$ that they make true. Taking into account the distinction between s and its representation $[s]$, the model [A1]–[A5] associated to a Strips problem $P = \langle \mathcal{L}_S, \mathcal{O}_S, \mathcal{I}_S, \mathcal{G}_S \rangle$ can be reformulated as follows:

- B1. the states $s \in S$ are the *logical interpretations* over the language \mathcal{L}_S , and they are *represented* by the set $[s]$ of atoms that they make true
- B2. the initial state s_0 is the interpretation that makes the atoms in \mathcal{I}_S true and all other atoms false
- B3. the goal states $s \in S_G$ are the interpretations that make the atoms in \mathcal{G}_S true
- B4. the actions $a \in A(s)$ are the operators $op \in \mathcal{O}_S$ whose preconditions are true in s
- B5. the transition function *next* maps states s into states $s' = next(a, s)$ for $a \in A(s)$ such that the representation of s' is $[s'] = [s] - Del(a) + Add(a)$.

It is simple to show that for any Strips problem the state models [A1]–[A5] and [B1]–[B5] are equivalent (they possess the same solutions). This equivalence is independent of the representation function for constants used. Model [B1]–[B5],

however, is more flexible than model [A1]–[A5] as it can be easily modified to accommodate other languages. For example, *negation* can be accommodated by representing states by the *literals* they make true ([B1]) and by adjusting the representation of successor states ([B5]).³ As we will see below, similar modifications are needed to accommodate *functions*.

3 FUNCTIONAL STRIPS

The language of Functional Strips differs from Strips in the two main aspects: function symbols are allowed and they can be fluents. These are small changes but with interesting consequences for modeling and problem solving.

3.1 MOTIVATION

As an illustration, let us consider the Towers of Hanoi problem. This is a standard problem in AI (Newell and Simon, 1972; Nilsson, 1980; Pearl, 1983) that involves a number of disks of different sizes that have to be moved from one peg to another. Only disks at the top of a peg can be moved and they can never be placed on top of smaller disks.

The formulation of this problem in Strips requires relations like *on*(*i*, *j*), *clear*(*i*), and *smaller*(*i*, *j*), and actions like *move*(*i*, *j*, *k*) with *i*, *j*, and *k* ranging over the disk names.⁴ If the number of disks is *N*, this implies in the order of N^3 ground actions. For *N* = 10, this means 1000 ground actions. These ground actions can be described very conveniently by means of schemas or quantification, yet most modern planners including Graphplan, SAT and heuristic planners (Blum and Furst, 1995; Kautz and Selman, 1999; Bonet and Geffner, 1999) require the substitution of schemas by their ground instances. This causes a real computational problem if *N* is large. Interestingly, the number of actions that are ever *applicable* in a state is given by the number of *pegs* and not the number of *disks*. In particular, with 3 pegs, there can never be more than 3 ground actions applicable in any state.

The same problem appears in slightly different form in the extensions of Strips that accommodate conditional effects and quantification such as ADL (Pednault, 1989). In an ADL language, it's possible to formulate the problem so that the number of *ground actions* can be kept small but at the expense of a large number of *ground conditional effects*.⁵ For example, using the relations *top*(*p_i*, *d_k*) to state that the top disk in peg *p_i* is *d_k*, and *on*(*d_k*, *d_l*) to express that disk *d_k* is on disk *d_l*, then the actions *move*(*p_i*, *p_j*) can be defined over *pegs* so that when *top*(*p_i*, *d_k*), *top*(*p_j*, *d_l*), and *on*(*d_k*, *d_m*) are all true, *top*(*p_i*, *d_m*), *top*(*p_j*, *d_k*), and \neg *top*(*p_i*, *d_k*) become true in the successor state. Yet again, there will be N^3 *ground conditional effects* associated with each action.

³In addition, the initial state in [B2] has to be fully determined without closed world assumptions; see (Nebel, 1998).

⁴This is a slight simplification as the bottom disks are not on top of other disks. This, however, is not relevant to our discussion.

⁵The recent ADL planners are based on Graphplan and also replace schemas by their ground instances; see (Gazen and Knoblock, 1997; Koehler et al., 1997; Anderson et al., 1998).

- Domains:** $Peg : p_1, p_2, p_3$; the pegs
 $Disk : d_1, d_2, d_3, d_4$; the disks
 $Disk* : Disk, d_0$; the disks and a dummy bottom disk 0
- Fluents:** $top : Peg \mapsto Disk*$; denotes top disk in peg
 $loc : Disk \mapsto Disk*$; denotes disk below given disk
 $size : Disk* \mapsto Integer$; represents disk size
- Action:** $move(p_i, p_j : Peg)$; moves between pegs
- Prec:** $top(p_i) \neq d_0$, $size(top(p_i)) < size(top(p_j))$
- Post:** $top(p_i) := loc(top(p_i))$; $loc(top(p_i)) := top(p_j)$; $top(p_j) := top(p_i)$
- Init:** $loc(d_1) = d_0$, $loc(d_2) = d_1$, $loc(d_3) = d_2$, $loc(d_4) = d_3$,
 $top(p_1) = d_4$, $top(p_2) = d_0$, $top(p_3) = d_0$,
 $size(d_0) = 4$, $size(d_1) = 3$, $size(d_2) = 2$, $size(d_3) = 1$, $size(d_4) = 0$
- Goal:** $loc(d_1) = d_0$, $loc(d_2) = d_1$, $loc(d_3) = d_2$, $loc(d_4) = d_3$, $top(p_3) = d_4$

Figure 9.1 Formulation of 3-towers-of-hanoi in Functional Strips

The problem with the large number of ground instances in Strips and ADL formulations is a consequence of the demand that objects be referred to by their unique names. Indeed, once function symbols are allowed as ‘first-class citizens’ and compound terms can be used to name objects, this problem can be avoided.⁶ Functional Strips is based on this idea and replaces the *relational* fluents in Strips with *functional* fluents. Provided with fluent *terms* like $top(p_i)$ and $top(p_j)$ representing the top disks in pegs p_i and p_j , the effects of the action $move(p_i, p_j)$ can be expressed as affecting the disks $top(p_i)$ and $top(p_j)$ directly, without having to appeal to the explicit names of those disks.

A complete formulation of Towers of Hanoi in Functional Strips is shown in Fig. 9.1. The most significant change from Strips is the use of postconditions of the form

$$f(t) := w$$

for terms $f(t)$ and w , in place of add and delete lists. A postcondition of that form says that in the state $s_a = next(a, s)$ that results from doing action a in state s , the denotation f^{s_a} of fluent f must become such that the equation

$$f^{s_a}(t^s) = w^s$$

holds, where t^s and w^s refer to the denotations of t and w in the state s .

⁶ It must be noted that the original formulation of ADL accommodates function symbols but not as ‘first-class citizens’; in particular, they cannot appear nested in the head of actions rules; see (Pednault, 1989).

The formulation in Fig. 9.1 uses the function $loc(d_k)$ to denote the disk below d_k and $size(d_k)$ to encode the size of disk d_k . A postcondition of the action $move(p_i, p_j)$ like $loc(top(p_i)) := top(p_j)$ therefore says that the action makes $loc(d_k) = d_l$ true when $d_k = top(p_i)$ and $d_l = top(p_j)$ are both true. This follows from the semantics sketched above and explained below in further detail. Note that the number of ground actions for the problem depends on the number of pegs but not on the number of disks.

3.2 LANGUAGE

The state language in Functional Strips (FStrips) is a first-order language with no quantification, involving *constant*, *function* and *relational symbols* but no variable symbols.

For simplicity, we assume that all *fluent* symbols are encoded as *function* symbols. Constant fluent symbols can be encoded as function symbols of arity 0, while relational fluent symbols can be encoded as function symbols of the same arity plus equality. This guarantees that any Strips representation can be easily translated into FStrips, even if FStrips often provides alternative encodings. For example, atoms in the blocks-world expressed as $on(a, b)$ can be encoded more conveniently in Functional Strips as $loc(a) = b$, where loc is a function symbol that makes explicit that blocks are at a single location.

As before, we call the non-fluent symbols in the language, the *fixed* symbols. They include all constant and relational symbols, as well as the function symbols that are not fluents. As in Strips, we assume that the denotation x^* of fixed symbols x is *fixed* by a *representation function* and consider only the interpretations s for which $x^s = x^*$. Among the fixed symbols we have a finite set \mathcal{C} of *object names* that are assumed to refer to different objects, and constant, function, and relational symbols such as ‘3’, ‘+’, ‘=’ that have a standard interpretation. The denotation of *fixed terms* t , i.e., terms involving no fluent symbols, does not depend on the state and is expressed as t^* . We call the terms that involve fluent symbols, *fluent terms*.

For the *representation* of states to be compact and finite, the language of Functional Strips is typed. The formulation of Hanoi, for example, involves the types *Peg*, *Disk*, and *Disk**. The use of types is common in planning languages (McDermott, 1998b) where they are used to delimit the range of action schemas. In FStrips, they also define the domains over which fluents are interpreted. For example the declaration $Disk : d_1, d_2, d_3, d_4$ says that the constant symbols d_i , $i = 1, 2, 3, 4$ denote objects d_i^* of type *Disk* and that there are no other *Disk* objects. Similarly, the declaration $Disk* : Disk, d_0$ says that the *Disk** objects are given by the *Disk* objects and the object denoted by d_0 .

The arguments of functional fluents must range over domains that are *finite*. Examples of such domains are booleans, finite integer intervals, and enumerated domains like $Peg = \{p_1, p_2, p_3\}$. The representation function ‘*’ for constants maps the standard symbols ‘=’, ‘+’, ‘3’, etc, into their standard interpretation, and object names like p_1 , p_2 , etc, into different integers p_1^* , p_2^* , etc. Under suitable syntactic conditions, it can be proved that this mapping is irrelevant as long as different names are mapped into different objects.

3.3 OPERATORS

In FStrips, an operator op is described by the type of its arguments and two sets: the *precondition* and the *postcondition* lists, referred to as $Prec(op)$ and $Post(op)$. The precondition list is a set of *formulas*, while the postcondition list is a set of *updates* of the form:

$$f(t) := w \quad (9.1)$$

where $f(t)$ and w are terms of the same type, and f is a fluent symbol. Updates like (9.1) express how fluent f changes when an action is taken. Such postcondition says that the denotation f^{s_a} of f in the successor state $s_a = next(a, s)$ must become such that the following equation is satisfied:

$$f^{s_a}(t^s) = w^s \quad (9.2)$$

For example, an update like $h := h + 1$ means that h is incremented by 1, while an update like $loc(top(p_1)) := top(p_2)$ says that $loc(d_4) = d_3$ must become true in s_a when $top(p_1) = d_4$ and $top(p_2) = d_3$ are true in s .

Note that the terms t and w in (9.1) are interpreted in the state s in which the action a is taken, and affect the denotation of the fluent f in the next state. Postconditions, thus, do not interact. Also Equation 9.2 says nothing about the persistence of fluents; this is treated below.

3.4 FUNCTIONAL STRIPS STATE MODEL

The state-model for Functional Strips defines the semantics of the operators and the planning task. The model is similar to the non-propositional Strips model [B1]–[B5] where states are interpretations over the language and operators stand for updates on state representations. The differences arise from the differences in the language.

3.4.1 States and State Transitions. The states s are interpretations over the language \mathcal{L}_F defined by the fluents and constants. Given the representation function of constants, states s can be represented by the interpretation f^s of fluent symbols f only. We refer to the domain of the function f^s denoted by f as D_f . This domain is finite and is independent of the state s .⁷

Given a state s and an action a applicable in s , the successor state $s_a = next(a, s)$ is defined by the denotation f^{s_a} of each fluent symbol f in s_a . This denotation is given by the equation

$$f^{s_a}(v) = \begin{cases} w^s & \text{if } f(t) := w \text{ in } Post(a) \text{ and } v = t^s \\ f^s(v) & \text{otherwise} \end{cases} \quad (9.3)$$

⁷In general D_f is a tuple of domains of the same arity as f and the arguments taken by f are tuples as well. The presentation is simplified assuming that the arity of f is 1 but the generalization is straightforward.

where v ranges over the objects in D_f . This equation extends (9.2) with the standard assumption of fluent persistence (Sandewall, 1994; Shanahan, 1997).

A domain description is *inconsistent* when for some state s and action a applicable in s , there is no state s_a satisfying equation (9.3). For example, a domain with an action a with conflicting postconditions $x := x+1$ and $x := x-1$ will be inconsistent. When a domain is not inconsistent, the successor state s_a given by equation (9.3) is uniquely defined.

3.4.2 State Representation and Updates. Equation 9.3 provides the *declarative semantics* of actions in Functional Strips. The *operational semantics* is defined in terms of state *representations*.

A *state representation* in Functional Strips is a an assignment of *values* to a finite number of *state variables*. For each fluent f denoting functions with domain D_f and range R_f , we create a finite set of state variables $f[v]$, one for each $v \in D_f$. A state s is represented by an assignment in which each state variable $f[v]$ is assigned a value $f_s[v]$ in R_f that stands for $f^s(v)$. Thus, the denotation f^s of f is encoded by the value of the finite set of state variables $f[v]$ which are implemented as an array.

The representation of the state s_a that follows an action a in the state s is obtained from (9.3) as

$$f_{s_a}[v] = \begin{cases} w^s & \text{if } f(t) := w \text{ in } Post(a) \text{ and } v = t^s \\ f_s[v] & \text{otherwise} \end{cases} \quad (9.4)$$

This computation is linear in the number of effects as in Strips. The overhead comes from the evaluation of the terms t and w in the state s , but this is negligible in general.

3.4.3 State Variables and Terms. We say that a term $f(t)$, where f is a fluent symbol and t is a tuple of fixed terms, *refers* to a state variable $f[v]$, when t denotes v ; i.e., $t^* = v$. When no confusion arises we also say that $f(t)$ is the state variable. For example, we say that the state in the Hanoi problem is represented by the values of the state variables $top(p_1), \dots, top(p_3)$, $loc(d_0), \dots, loc(d_N)$, and $size(d_0), \dots, size(d_N)$.

A term $f(t)$ where t involves fluent symbols will normally refer to different state variables in different states. E.g., $loc(top(p_1))$ refers to the state variable $loc(d_1)$ in states where $top(p_1) = d_1$ and to $loc(d_2)$ in states where $top(p_1) = d_2$. It's precisely this treatment of fluent function symbols as 'first-class citizens' that distinguishes Functional Strips from other planning and action languages that accommodate state-variables such as (Currie and Tate, 1991; Wilkins, 1988; Jonsson and Bäckstrom, 1994; Laborie and Ghallab, 1995; Penberthy and Weld, 1994; Koehler, 1998; Sandewall, 1994; Giunchiglia et al., 1997).

3.4.4 Stating Problems. A planning problem in Functional Strips is a tuple $P = \langle \mathcal{L}_F, \mathcal{O}_F, \mathcal{I}_F, \mathcal{G}_F \rangle$ where \mathcal{L}_F is the language, \mathcal{O}_F the operators, and \mathcal{I}_F and \mathcal{G}_F are formulas standing for the initial and goal situations. The language \mathcal{L}_F is defined by declaring the fluents and their domains,

while operators are defined by means of suitable schemas. In addition, a representation function '*' that maps fixed symbols into their denotation is assumed. The representation function for standard symbols like '=', '+', 3, etc, is assumed to be provided by the underlying programming language, while the representation function for object names maps different names into different objects (integers).

The formulas I_F defining the initial situation must define a unique state that should be easy to compute. For this reason, we assume that the formulas in I_F must be of the special form $f(t) = w$, where f is fluent symbol and t and w are *fixed* terms (i.e., terms involving no fluent symbols). The initial state s_0 then is such that $f^{s_0}(t^*) = w^*$.⁸

3.4.5 State Model. All the ingredients are in place to define the state model associated with a problem $P = \langle \mathcal{L}_F, \mathcal{O}_F, I_F, G_F \rangle$ in Functional Strips. The state model is such that

- C1. the states $s \in S$ are the logical interpretations over the language \mathcal{L}_F , and are represented by assigning a value $f_s[v]$ to each state variable $f[v]$ for each fluent f and value v in D_f
- C2. the initial state s_0 satisfies the equations $f(t) = w$ in I_F
- C3. the goal states $s \in S_G$ are the interpretations that satisfy the goal formula G_F
- C4. the actions $a \in A(s)$ are the operators $op \in \mathcal{O}_F$ whose preconditions are true in s
- C5. the representation of the next state $s_a = \text{next}(a, s)$ for $a \in A(s)$ is such that for each fluent symbol f and $v \in D_f$

$$f_{s_a}[v] = \begin{cases} w^s & \text{if } f(t) := w \text{ in } \text{Post}(a) \text{ and } v = t^s \\ f_s[v] & \text{otherwise (persistence)} \end{cases}$$

A number of examples will be used to illustrate the language.

4 EXAMPLES

4.1 BLOCKS

The blocks world domain is a convenient testbed for modeling and planning. The most common encoding in Strips involves an action $\text{move}(x, y, z)$ for moving a block x from a block y onto a block z , as well as actions for moving blocks to the table and from the table. In the presence of N blocks, this encoding leads to three action schemas with more than N^3 ground instances. As with Towers of Hanoi, ADL (Pednault, 1989) can model the problem with

⁸A partial characterization of the initial situation gives rise to a slightly different planning task that is usually referred to as *planning with incomplete information*. See (Smith and Weld, 1998; Bonet and Geffner, 2000).

Domains :	<i>Block : a, b, c, ... ; the blocks</i>
	<i>Loc : Block, table ; the locations: blocks + table</i>
Fluents:	<i>loc : Block \mapsto Loc</i>
	<i>clear : Loc \mapsto Bool</i>
Action:	<i>move(x : Block, y : Loc)</i>
Prec:	<i>clear(x); clear(y); x \neq y</i>
Post:	<i>loc(x) := y; clear(y) := (y = table); clear(loc(x)) := true</i>
Init:	<i>loc(a) = table; loc(b) = a; ..., clear(b), clear(table)</i>
Goal:	<i>loc(a) = b, loc(b) = table</i>

Figure 9.2 Formulation of **Blocks-world** in Functional Strips

a single conditional schema, but the number of ground conditional effects remains N^3 . An alternative often used in planning is to decompose the actions $move(x, y, z)$ into two actions with two arguments each: $unstack(x, y)$ and $stack(y, z)$. Similar actions are defined for placing and removing blocks from the table. Such decomposition reduces the number of ground actions to N^2 but makes planning harder by adding more choice points in the search.

In Functional Strips, it is possible to model this problem with N^2 ground actions and a single schema (Fig. 9.2). The action $move(x, y)$ moves a block x to a location y that can be another block or the table. The block on which x was located, denoted by $loc(x)$, becomes *clear* after the action, but its explicit name is not needed. We use *Bool* as the domain given by the boolean terms *true* and *false*, and assume functions analogous to ' $=$ ', ' \neg ', etc. that return boolean terms. Such functions can be easily defined in the underlying programming language. The postcondition $clear(y) := (y = table)$, for example, says that y becomes not *clear* if y is not the table. We also abbreviate terms of the form $t = \text{true}$ and $t = \text{false}$ by t and $\neg t$ respectively.

4.2 LOGISTICS

Logistics is a more recent benchmark in planning that deals with the transportsations of packages (Veloso, 1992; Kautz and Selman, 1996; McDermott, 1998a). Packages are transported in trucks among locations in the same city (including airports) and by planes among airports in different cities. In the Strips formulation one needs schemas for actions like

```

load(pkg, transpt, loc)
unload(pkg, transpt, loc)
drive_truck(truck, loc1, loc2)
fly_plane(plane, loc1, loc2)

```

where *transpt* refers to trucks and planes. In Functional Strips, the presence of functional fluents allows for a more concise representation in which the

Domains: $Pkg : o_1, \dots, o_{10}$
 $Truck : t_1, t_2, \dots, t_4$
 $Plane : p_1, p_2, \dots, p_3$
 $City : bos, pgh, lax, \dots$
 $Airpt : abos, pgh, alax, \dots$
 $Locn : bos_1, bos_2, pgh_1, lax_1, \dots$
 $Site : Airpt, Locn ; Transp : Truck, Plane$
 $Thing : Transp, Pkg ; Loc : Transp, Site$

Fluents: $loc : Thing \leftarrow Loc$
 $city : Site \leftarrow City$

Action: $\text{load}(pkg : Pkg, target : Transp)$
Prec: $loc(pkg) = loc(target)$
Post: $loc(pkg) := target$

Action: $\text{unload}(pkg : Pkg)$
Prec: $transp?(loc(pkg))$
Post: $loc(pkg) := loc(loc(pkg))$

Action: $\text{drive_truck}(t : Truck, dest : Site)$
Prec: $city(loc(t)) = city(dest)$
Post: $loc(t) := dest$

Action: $\text{fly_plane}(p : Plane, dest : Airpt)$
Prec: $airport?(loc(p))$
Post: $loc(p) := dest$

Init: $city(abos) = bos, city(bos_1) = bos, \dots,$
 $loc(t_1) = abos, loc(t_2) = apgh, loc(t_3) = lax_1,$
 $loc(p_1) = abos, loc(p_2) = alax, \dots$
 $loc(o_1) = bos_1, loc(o_2) = bos_2, loc(o_3) = pgh_1$
Goal: $loc(o_1) = pgh, loc(o_2) = pgh, loc(o_3) = alax, \dots$

Figure 9.3 Formulation of Logistics in Functional Strips

number of action arguments can be reduced (e.g., `unload` requires the package argument only). This encoding is shown in Fig. 9.3.

The function `city` is defined as a fluent even though it is fixed in the initial situation and doesn't change. A straightforward extension would allow us to declare such symbols as parameters rather than fluents (actually such extension is supported in the PDDL standard (McDermott, 1998b)). We also make use of type predicates like `airport?(t)` to test if t denotes an object of type `Airport`.

Domain:	$\text{Pos} : 1, \dots, 9$
	$\text{Tile} : 0, \dots, 8$
	$\text{Pos}^* : \text{Pos}, 0$
Fluent:	$\text{tile} : \text{Pos} \mapsto \text{Tile}$
	$\text{bp} : \text{Pos}$
Fixed:	$u, l, r, d : \text{Pos} \mapsto \text{Pos}^*$
Action:	up
Prec:	$u(\text{bp}) \neq 0$
Post:	$\text{bp} := u(\text{bp}) ; \text{tile}(\text{bp}) := \text{tile}(u(\text{bp})) ; \text{tile}(u(\text{bp})) := \text{tile}(\text{bp})$
Action:	down, left, right, ...
Init:	$\text{tile}(1) = 2, \text{tile}(2) = 0, \text{tile}(3) = 2, \dots, \text{tile}(9) = 5, \text{bp} = 2$
Goal:	$\text{tile}(1) = 1, \text{tile}(2) = 2, \text{tile}(3) = 3, \dots, \text{tile}(9) = 8$

Figure 9.4 Formulation of 8-puzzle in Functional Strips

4.3 8-PUZZLE

The 8-puzzle is a standard problem in heuristic search (Nilsson, 1980; Pearl, 1983). In Functional Strips, the description of the problem is very compact due to the ability to nest fluents and attach user defined functions to symbols (Fig. 9.4). There are two main *fluent* symbols: $\text{tile} : \text{Pos} \mapsto \text{Tile}$ that maps each grid position to the tile that occupies the position, and bp that keeps track of the position of the ‘blank’ tile. In addition, there are *four* symbols, u, l, \dots that represent functions that map a position into each one of its four neighboring position (positions outside the grid are denoted by 0). For example, assuming that the top row positions are 1, 2 and 3, the second row positions are 4, 5, and 6, and so on, we must have $u(5) = 2, u(2) = 0$, etc. This interpretation of the symbols u, l, \dots can be defined *extensionally* by modeling them as fluents and enumerating these equations in the initial situation, or *intensionally* by modeling these symbols as fixed symbols with an interpretation provided in the underlying language. In such case, a function u^* must be defined in the underlying language such that $u^*(x)$ returns 0 if $x \leq 3$ and returns $x - 3$ otherwise. Something similar has to be done for the other functions d^*, r^* , and l^* . The declaration that u, d, r , and l are **fixed** symbols in Fig. 9.4 indicates that their denotation is defined in the underlying programming language.

In the resulting model, the actions **up**, **left**, ..., have no arguments. Indeed, the number of ground actions matches exactly the branching factor of the problem. In addition, the resulting state representation is in close correspondence with the representation used in specialized programs. It’s worth emphasizing that these are features that are necessary (although not sufficient) for making planning approaches competitive with specialized solvers.

5 RESOURCES

The word ‘resources’ in planning and scheduling refers to objects that can be produced, consumed, or ‘borrowed’ during the execution of plans, constraining the possible actions (Wilkins, 1988; Currie and Tate, 1991; Laborie and Ghallab, 1995; Penberthy and Weld, 1994; El-Kholy and Richards, 1996; Koehler, 1998). E.g., driving a car requires and consumes fuel, building a wall requires and consumes bricks, etc. An implicit assumption when a collection of objects is represented as a resource is that the identity of the objects in the set does not matter. This is important as actions and states that would be different if objects were named individually are collapsed. Namely, actions like ‘grabbing brick1’, ‘grabbing brick2’, and so on, are replaced by the single action ‘grabbing a brick’. Such simplified representations can have a significant impact on planning (Wilkins, 1988; Currie and Tate, 1991).

Resources are usually represented as real or integer state variables. In certain cases, such state variables can depend on one or more arguments (e.g., *level_fuel(car₁)*). Functional Strips, by making function symbols ‘first-class citizens’, allows these and other uses of state-variables. In particular, state-variables can be nested, allowing for terms like *B(loc)* where both symbols *loc* and *B* are fluents. We illustrate the uses of such constructs by considering a variation of the ‘Gripper’ domain used in the AIPS-98 Planning Competition (McDermott, 1998a).

‘Gripper’ involves a robot with a number of grippers that can move between rooms, picking up and dropping balls from its grippers (one ball per gripper at most). In the competition, this problem proved to be hard, and three out of the four competing planners solved a few instances only. The Strips formulation of Gripper involves names for each ball and each gripper, along with predicates for keeping track of the status of each one of them (the location of balls, whether a gripper is free or not, etc). As discussed in (Fox and Long, 1999), this representation produces a number of symmetries that if exploited at run-time can improve planner performance significantly. Alternatively, these symmetries can often be exploited at *modeling time*. For example, if we do not care about the identity of the individual balls and grippers, balls and grippers can be modeled as *resources*. This leads to substantial simplifications in both the branching factor of the problem and the state representation. The resource formulation of Gripper in Functional Strips is shown in Fig. 9.5. While in the Strips formulation, the actions *move*, *pick* and *drop* involve 2 or 3 arguments, in the ‘resource’ formulation only the action *move* needs an argument. The fluent *loc* keeps track of the room where the robot is, *B* keeps track of the number of balls in each room, and *G* stands for the number of grippers. A term like *B(loc)* thus denotes the number of balls in the room where the robot is located. The denotation of this term changes when either the number of balls in the room or the position of the robot changes. The state representation that results from the formulation in Fig. 9.5 is once again as economical as the representation that can be obtained in a specialized program.

Domain: $Room : a, b, c, \dots$; the rooms

Fluents: $loc : Room$; the room where the robot is
 $G : Int$; number of grippers
 $B : Room \mapsto Int$; number of balls in each room
 $H : Int$; number of balls being held

Action: $move(dest : Room)$

Prec: —

Post: $loc := dest$

Action: $pick$

Prec: $H < G ; B(loc) > 0$

Post: $H := H + 1 ; B(loc) := B(loc) - 1$

Action: $drop$

Prec: $H > 0$

Post: $H := 0 ; B(loc) := B(loc) + H$

Init: $G = 2, H = 0, loc = a, B(a) = 20, B(b) = 0$

Goal: $B(b) = 15$

Figure 9.5 Formulation of Gripper in Functional Strips

6 PLANNING AND PROBLEM SOLVING

In this section we discuss briefly how Functional Strips can be used in planning and problem solving. We distinguish two cases. In *domain-independent planning*, the domain descriptions are assumed to encode all the knowledge needed to solve the problem, including the dynamics of the domain and the control knowledge. In *specialized problem solving*, on the other hand, domain descriptions encode the dynamics of the domain but the control knowledge can be provided separately (e.g., in the form of an heuristic function or a set of control rules). We focus first on the latter case.

6.1 PROBLEM SOLVING WITH FUNCTIONAL STRIPS

Consider writing a program for solving a problem like Rubik's cube. One option is to write the program in a programming language such as C. One would then write some routines for modeling the dynamics of the problem and other routines for capturing the control; namely, which action to try next, when and where to backtrack, etc. This is actually the most common option for solving combinatorial problems and it's the approach taken for example in (Korf, 1998). The advantage of this approach is that it can be very efficient at *run time*; the disadvantage, is that it may be quite inefficient at *modeling time*. That is, building a good specialized program takes time, and usually involves a tedious process of debugging and tuning.

A modeling language such as Functional Strips can be used in this setting to reduce *modeling time* without incurring a substantial overhead at *run time*. For that, Functional Strips can be used for describing the dynamics of the domain which can then be automatically compiled into efficient run-time procedures (i.e., procedures for testing when an action is applicable in a state and for computing successor states). These compiled procedures can take the place of the routines written by hand. They will impose a minimal overhead if the encoding of the problem is such that the resulting state representation is in correspondence with the state representation used by a specialized program. As argued above, this can often be achieved in Functional Strips but is more difficult to achieve in Strips or ADL languages where the number of ground actions or ground conditional effects often explodes and state representations have often little to do with specialized representations.

We have actually implemented a tool that accepts descriptions of problems in Functional Strips and compiles them into state space search procedures. The heuristic function used by these procedures is supplied by the user in the form of a C++ routine. This tool can be seen as a domain-*dependent* planner in the style of TLPLAN (Bacchus and Kabanza, 2000) but while in TLPLAN the control knowledge is expressed declaratively in a logical language, in this tool, the control knowledge is expressed procedurally in the form of an heuristic function. Recent heuristic search planners like HSP are based on a similar idea but rather than relying on a user-supplied heuristic function, they extract the heuristic function automatically from the problem description. We elaborate on this below.

6.2 PLANNING WITH FUNCTIONAL STRIPS

HSP (Bonet and Geffner, 1999) is a planner that maps Strips problems into state-space search problems that are solved with an heuristic extracted from the Strips encodings. For a state s , the heuristic value $h(s)$ that estimates the distance from s to the goal is obtained by computing the cost of achieving each atom p from s under some simplifying assumptions. The value $h(s)$ is then set to the sum of the costs of the atoms p in the goal. While the heuristic values $h(s)$ are not admissible (they may overestimate the true cost to the goal), they can be computed reasonably fast and are often quite informative (they drive the search in a good direction). From the results in the recent AIPS Planning Competition (McDermott, 1998a), HSP appears competitive with the state of the art Graphplan and SAT planners (Long and Fox, 1999; Koehler et al., 1997; Kautz and Selman, 1999).

The ideas of HSP can be used in the context of Functional Strips (see (Bonet et al., 1997) for related results). The key point is the automatic extraction of the heuristic from Functional Strips encodings. We have been exploring a number of ways for doing this, but coming up with an efficient implementation that can be competitive with HSP on similar problems has been difficult. The problem is that effects of the form $f(t) := w$ in Functional Strips are *state dependent* when the terms t or w involve fluent symbols. Such

state dependent effects can be compiled into *conditional* but *state-independent* effects of the form $C \rightarrow f(t') := w'$, where t' and w' have no fluent symbols and $C = (t = t') \wedge (w = w')$. These conditional effects are easier to deal with but, on the other hand, their number grows up with the product of the domains of the terms t and w . The resulting theory is actually in ADL format (Pednault, 1989).

Approaches that do not involve the extraction of an heuristic function are also possible. For example, (Van Beek and Chen, 1999) maps planning problems into *constraint satisfaction problems* that are then solved by domain-independent methods. This approach may prove suitable for Functional Strips where the state is represented by a finite set of state variables taking a finite set of values. On the other hand, SAT approaches (Kautz and Selman, 1996) that require all variables to be boolean may prove less convenient.

Finally, while it is not clear how to extend Graphplan (Blum and Furst, 1995) for dealing best with Functional Strips encodings, one possibility is to compile them into ADL encodings, while extracting useful mutual exclusivity relations from the functional relations. Graphplan approaches that handle ADL encodings have been reported in (Koehler et al., 1997; Anderson et al., 1998). Better results could actually be obtained by using extensions of Graphplan that handle non-boolean variables (Koehler, 1998). In any case, further work is needed to evaluate how the different planning approaches can benefit from the facilities provided by a fully functional action modeling language such as Functional Strips.

7 DISCUSSION

The work we have presented is motivated by a perspective in which *planning is general problem solving*. As such, planning should offer a general language for expressing problems and general algorithms for solving them. For such an approach to be useful though, the time required to model problems and find the solutions has to be competitive with the time required to set up and solve specialized models. The quality of the solutions has to be competitive as well. This will be possible only by a suitable combination of general and effective languages and algorithms as shown by the closely related work in the area of Constraint Programming (Hentenryck et al., 1992; Marriot and Stuckey, 1999).

Due to the similarity between Functional Strips, Strips, and the \mathcal{A} action representation language (Gelfond and Lifschitz, 1993), the extensions found in ADL (conditional effects, negation, and quantification (Pednault, 1989)) and languages such as \mathcal{AR} (ramifications) (Giunchiglia et al., 1997) can easily be integrated with the extensions accommodated in Functional Strips. Among other extensions that are likely to be necessary in a good modeling language for planning, we mention the following:⁹

⁹Some current systems that handle expressive action languages that go well beyond Strips are CCALC (McCain and Turner, 1998), GOLOG (Levesque et al., 1997), and TLPLAN (Bacchus and Kabanza, 2000). The first relies on a SAT formulation while the latter two on declarative, user-supplied, control knowledge.

- **Constraints.** Constraints represented as formulas can be used to provide implicit action preconditions (e.g., (Giunchiglia et al., 1997)). Namely, the set $A(s)$ of actions applicable in a state s will exclude all actions a that lead to states $s_a = \text{next}(a, s)$ that violate a constraint. Such constraints can be used to express capacity constraints (e.g., that the number of balls being held cannot exceed the number of grippers) or control knowledge (Bacchus and Kabanza, 2000). With constraints, any Constraint Satisfaction Problem (CSP) can be expressed as a Planning problem. The question is how to make those constraints play an active role in the search. So far, only approaches based *exclusively* on constraint techniques are able to do that (e.g., (Kautz and Selman, 1999; Van Beek and Chen, 1999)). A combination of constraint-directed and heuristic search techniques may also prove useful.
- **Costs.** The state models discussed above can be extended to take into account action costs. Traditionally, the focus in planning has been on *uniform* action costs, but more generally costs may depend on actions and states. For example, a *fluent* t standing for *time* can be included so that actions increase t by their time duration. Then, defining $c(a, s)$ as the duration of a in s , i.e., $c(a, s) = (t^{s_a} - t^s)$ where $s_a = \text{next}(a, s)$, a formulation can be obtained in which optimal plans stand for plans with minimum completion times. In a different way, optimal *parallel plans*, such as those computed by Graphplan and SAT planners, can be defined as well. The issue is how to plan effectively with such flexible cost structures.
- **Data Structures.** The state representation of the 8-puzzle is close to the representation one could find in a specialized program. However, the same cannot be said for Towers of Hanoi. In a specialized program, the state would not be represented by a set of equalities $\text{loc}(\text{disk}_i) = \text{disk}_j$, but by three lists $(\text{disk}_1, \text{disk}_2, \dots)$ one for each peg, meaning that disk_1 is on disk_2 which is on disk_3 , etc. Then the action of moving a disk from one peg to another would be implemented as operations on the heads of these lists. One way to achieve such efficient representations in FStrips is by allowing Lists as primitive types. Then functions like *car*, *cdr*, *cons*, etc. defined in the underlying language can be made available in the planning language. This would allow high-level representations of Tower of Hanoi and other problems that are as efficient as specialized representations.

We have developed a tool that supports some of these extensions. The tool also accommodates probabilistic actions and partial observability; see (Bonet and Geffner, 1998; Bonet and Geffner, 2000) for details.

Acknowledgements: The tool discussed above has been built by Blai Bonet. Blai has also provided valuable feedback on a number of topics related to this paper. Part of this work was done while I was visiting IRIT, Toulouse, and Linköping University. I thank J. Lang, D. Dubois, H. Prade, and H. Farreny in

Toulouse, and E. Sandewall and P. Doherty in Linköping for their hospitality and for making these visits possible. I also thank Chitta Baral for useful comments.

References

- Anderson, C., Smith, D., and Weld, D. (1998). Conditional effects in Graphplan. In Simmons, R., Veloso, M., and Smith, S., editors, *Proceedings of the Fourth International Conference on AI Planning Systems (AIPS-98)*, pages 44–53. AAAI Press.
- Bacchus, F. and Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123–191.
- Blum, A. and Furst, M. (1995). Fast planning through planning graph analysis. In Mellish, C., editor, *Proceedings of IJCAI-95*, pages 1636–1642. Morgan Kaufmann.
- Bonet, B. and Geffner, H. (1998). High-level planning and control with incomplete information using POMDPs. In Giacomo, G. D., editor, *Proceedings AAAI Fall Symp. on Cognitive Robotics*, pages 52–60. AAAI Press.
- Bonet, B. and Geffner, H. (1999). Planning as heuristic search: New results. In *Proceedings of ECP-99*, pages 359–371. Springer.
- Bonet, B. and Geffner, H. (2000). Planning with incomplete information as heuristic search in belief space. In Chien, S., Kambhampati, S., and Knoblock, C., editors, *Proc. of the Fifth International Conference on AI Planning and Scheduling (AIPS-2000)*, pages 52–61. AAAI Press.
- Bonet, B., Loerincs, G., and Geffner, H. (1997). A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97*, pages 714–719. MIT Press.
- Currie, K. and Tate, A. (1991). O-Plan: the open planning architecture. *Artificial Intelligence*, 52(1):49–86.
- El-Kholy, A. and Richards, B. (1996). Temporal and resource reasoning in planning: the parcPLAN approach. In Wahlster, W., editor, *Proc. ECAI-96*, pages 614–618. Wiley.
- Fikes, R. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120.
- Fox, M. and Long, D. (1999). The detection and exploitation of symmetry in planning domains. In Dean, T., editor, *Proc. IJCAI-99*, pages 956–961. Morgan Kaufmann.
- Gazen, B. and Knoblock, C. (1997). Combining the expressiveness of UCPOP with the efficiency of Graphplan. In Steel, S. and Alami, R., editors, *Recent Advances in AI Planning. Proc. 4th European Conf. on Planning (ECP-97). Lect. Notes in AI 1348*, pages 221–233. Springer.
- Gelfond, M. and Lifschitz, V. (1993). Representing action and change by logic programs. *J. of Logic Programming*, 17:301–322.
- Gelfond, M. and Lifschitz, V. (1998). Action languages. *Electronic Transactions on AI*, 3(16):301–322. At www.ep.liu.se/ea/cis/1998/016.

- Giunchiglia, E., Kartha, N., and Lifschitz, V. (1997). Representing action: indeterminacy and ramifications. *Artificial Intelligence*, 95:409–443.
- Hentenryck, P. V., Simonis, H., and Dincbas, M. (1992). Constraint satisfaction using constraint logic programming. *Artificial Intelligence*, 58(1–3):113–159.
- Jonsson, P. and Bäckstrom, C. (1994). Tractable planning with state variables by exploiting structural restrictions. In *Proc. AAAI-94*, pages 998–1003. AAAI Press / MIT Press.
- Kautz, H. and Selman, B. (1996). Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, pages 1194–1201. AAAI Press / MIT Press.
- Kautz, H. and Selman, B. (1999). Unifying SAT-based and Graph-based planning. In Dean, T., editor, *Proceedings IJCAI-99*, pages 318–327. Morgan Kaufmann.
- Khardon, R. (1999). Learning action strategies for planning domains. *Artificial Intelligence*, 113:125–148.
- Koehler, J. (1998). Planning under resource constraints. In *Proc. of the 13th European Conference on AI (ECAI-98)*, pages 489–493. Wiley.
- Koehler, J., Nebel, B., Hoffman, J., and Dimopoulos, Y. (1997). Extending planning graphs to an ADL subset. In Steel, S. and Alami, R., editors, *Recent Advances in AI Planning. Proc. 4th European Conf. on Planning (ECP-97). Lect. Notes in AI 1348*, pages 273–285. Springer.
- Korf, R. (1998). Finding optimal solutions to Rubik's cube using pattern databases. In *Proceedings of AAAI-98*, pages 1202–1207. AAAI Press MIT Press.
- Laborie, P. and Ghallab, M. (1995). Planning with sharable resources constraints. In Mellish, C., editor, *Proc. IJCAI-95*, pages 1643–1649. Morgan Kaufmann.
- Levesque, H., Reiter, R., Lespérance, Y., Lin, F., and Scherl, R. (1997). GOLOG: A logic programming language for dynamic domains. *J. of Logic Programming*, 31:59–83.
- Lifschitz, V. (1986). On the semantics of STRIPS. In Georgeff, M. and Lansky, A., editors, *Proc. Reasoning about Actions and Plans*, pages 1–9. Morgan Kaufmann.
- Long, D. and Fox, M. (1999). The efficient implementation of the plan-graph. *JAIR*, 10:85–115.
- Marriot, K. and Stuckey, P. (1999). *Programming with Constraints*. MIT Press.
- McCain, N. and Turner, H. (1998). Fast satisfiability planning with causal theories. In T. Cohn, L. S. and Shapiro, S., editors, *Proceedings of the Sixth Int. Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, pages 121–130. Morgan Kaufmann.
- McDermott, D. (1998a). AIPS-98 Planning Competition Results. At <http://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html>.
- McDermott, D. (1998b). PDDL – the planning domain definition language. Available at <http://ftp.cs.yale.edu/pub/mcdermott>.
- Nebel, B. (1998). On the compilability and expressive power of propositional planning formalisms. Technical Report 101, Freiburg University. At <http://www.informatik.uni-freiburg.de/~nebel>.

- Newell, A. and Simon, H. (1972). *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ.
- Nilsson, N. (1980). *Principles of Artificial Intelligence*. Tioga.
- Pearl, J. (1983). *Heuristics*. Morgan Kaufmann.
- Pednault, E. (1989). ADL: Exploring the middle ground between Strips and the situation calculus. In Brachman, R., Levesque, H., and Reiter, R., editors, *Proc. KR-89*, pages 324–332. Morgan Kaufmann.
- Penberthy, J. and Weld, D. (1994). Temporal planning with continuous change. In *Proc. AAAI-94*, pages 1010–1015.
- 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.
- Sandewall, E. (1994). *Features and Fluents. The Representation of Knowledge about Dynamical Systems*. Oxford Univ. Press.
- Shanahan, M. (1997). *Solving the Frame Problem*. MIT Press.
- Smith, D. and Weld, D. (1998). Conformant graphplan. In *Proceedings AAAI-98*, pages 889–896. AAAI Press.
- Van Beek, P. and Chen, X. (1999). CPlan: a constraint programming approach to planning. In *Proc. National Conference on Artificial Intelligence (AAAI-99)*, pages 585–590. AAAI Press/MIT Press.
- Veloso, M. (1992). *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, Computer Science Department, CMU. Tech. Report CMU-CS-92-174.
- Wilkins, D. (1988). *Practical Planning: Extending the classical AI paradigm*. Morgan Kaufmann.

VII

**LOGIC, PLANNING AND
HIGH LEVEL ROBOTICS**

Chapter 10

PLANNING WITH NATURAL ACTIONS IN THE SITUATION CALCULUS

Fiora Pirri,

Dipartimento di Informatica e Sistemistica

Università degli Studi di Roma "La Sapienza"

Via Salaria 113, 00198 Roma, Italy

pirri@assi.dis.uniroma1.it

Raymond Reiter

Department of Computer Science

University of Toronto

Toronto, Canada

reiter@cs.toronto.edu

Abstract We describe the theory and implementation of a deductive planner in the situation calculus for domains with two kinds of actions:

1. "Free will" actions on the part of agents with the ability to perform or withhold their actions, like choosing to pick up an object, or deciding to walk to some location.
2. Natural actions whose occurrence times are predictable in advance, in which case they must occur at those times unless something happens to prevent them, for example, objects moving under Newtonian laws, or trains arriving and departing in accordance with known schedules.

The theoretical basis for our planner is an extension of the situation calculus to accommodate continuous time and natural actions. The planner itself is patterned after that proposed by (Bacchus and Kabanza, 1995; Bacchus and Kabanza, 2000); it is a forward reasoning planner that filters out partial plans using domain and problem-specific information supplied by the user. The planner is implemented in ECLIPSE Prolog, and exploits that system's built-in linear constraint solver to do temporal reasoning. We illustrate the planner's workings on a space platform example that we fully axiomatize in the situation calculus.

Keywords: Deductive planning, natural actions, situation calculus, continuous time, Golog, constraint logic programming.

1 INTRODUCTION

The earliest formulation of the planning problem in artificial intelligence was as a deduction problem in the situation calculus (Green, 1969). On this view, actions and their effects are axiomatized for a given domain of application, as well as the domain's initial state, and the task for a given goal is to find a sequence of actions (a situation) such that these axioms entail the goal. For a variety of reasons having to do with the frame problem, and inefficiencies associated with automated theorem proving, this approach was largely abandoned during the 1970's in favour of STRIPS-style representations and planning algorithms (Fikes and Nilsson, 1971; Fikes and Nilsson, 1993). One purpose – not the only purpose – of this paper is to suggest that this rejection of logic and the situation calculus for planning was perhaps a bit hasty.

The technical contribution of this paper is the theory and implementation of a deductive planner in the situation calculus for domains with two kinds of actions:

1. "Free will" actions on the part of agents with the ability to perform or withhold their actions, like choosing to pick up an object, or deciding to walk to some location.
2. Natural actions whose occurrence times are predictable in advance, in which case they must occur at those times unless something happens to prevent them, for example, objects moving under Newtonian laws, or trains arriving and departing in accordance with known schedules.

The theoretical basis for our planner is an extension of the situation calculus to accommodate continuous time and natural actions. The planner itself is patterned after that proposed by (Bacchus and Kabanza, 1995; Bacchus and Kabanza, 2000); it is a forward reasoning planner that filters out partial plans using domain and problem-specific information supplied by the user. The planner is implemented in ECLIPSE Prolog, and exploits that system's built-in linear constraint solver to do temporal reasoning. We illustrate the planner's workings on a space platform example that we fully axiomatize in the situation calculus.

2 PRELIMINARIES

2.1 THE SITUATION CALCULUS

The situation calculus (McCarthy, 1963) is a first order language for axiomatizing dynamic worlds. In recent years, it has been considerably extended beyond the "classical" language to include concurrency, continuous time, etc, but in all cases, its basic ingredients consist of *actions*, *situations* and *fluents*.

2.1.1 Actions. Actions are first order terms consisting of an action function symbol and its arguments. In the approach to representing time in the situation calculus of (Reiter, 1996), one of the arguments to such an action function symbol – typically, its last argument – is the time of the action's occurrence. For example, $\text{startMeeting}(\text{Susan}, 3.14)$ might denote the action of Susan starting a meeting at time 3.14. We shall follow Reiter's approach to time here. That will mean, in particular, that all actions are viewed as instantaneous – they have zero duration. Despite this, it is possible to treat actions with positive duration by introducing processes into the situation calculus, as Example 1 below will show.

2.1.2 Situations. A *situation* is a first order term denoting a sequence of actions. These sequences are represented using a binary function symbol do : $\text{do}(\alpha, s)$ denotes the sequence resulting from adding the action α to the sequence s . So $\text{do}(\alpha, s)$ is like LISP's $\text{cons}(\alpha, s)$, or Prolog's $[\alpha \mid s]$. The special constant S_0 denotes the *initial situation*, namely the empty action sequence, so S_0 is like LISP's $()$ or Prolog's $[]$. Therefore, the situation term

$$\text{do}(\text{endWalk}(A, B, 7.3), \text{do}(\text{startChewGum}(2.4), \\ \text{do}(\text{startWalk}(A, B, 2.4), S_0)))$$

denotes the sequence of actions

$$[\text{startWalk}(A, B, 2.4), \text{startChewGum}(2.4), \text{endWalk}(A, B, 7.3)].$$

Notice that the action sequence is obtained from a situation term by reading the term from right to left.

Foundational axioms for situations without time are given in (Pirri and Reiter, 1999). These introduce the concept of subsequences of situations, which we shall need in this paper: $s \sqsubset s'$ means that the sequence of actions of s is a proper subsequence of those of s' . $s \sqsubseteq s'$ means that the subsequence need not be proper.

Foundational axioms for situations with time are given in (Reiter, 1998). These introduce two new function symbols, that we need in this paper: $\text{time}(a)$, denoting the occurrence time of action a , and $\text{start}(s)$, denoting the start time of situation s . The latter is easily characterized by:

$$\text{start}(\text{do}(a, s)) = \text{time}(a),$$

i.e. the start time of a situation is the occurrence time of the last action in its sequence. $\text{time}(a)$ is defined, for each action term a , to be its temporal argument, for example, $\text{time}(\text{startMeeting}(\text{person}, t)) = t$.

2.1.3 Fluents. Relations/functions whose values vary from state to state are called *fluents*, and are denoted by predicate/function symbols with last argument a situation term. For example, $\text{closeTo}(x, y, s)$ might be a relational fluent, meaning that in that state of the world reached by performing the action sequence s , x will be close to y . $\text{pos}(x, s)$ might be a functional fluent, denoting x 's position in that state of the world reached by performing the action sequence s .

2.2 AXIOMATIZING A DOMAIN THEORY

A domain theory is axiomatized in the situation calculus with four classes of axioms (More details in (Levesque et al., 1998)):

1. **Action precondition axioms.** There is one for each action function $A(\vec{x})$, with syntactic form $\text{Poss}(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$. Here, $\Pi_A(\vec{x}, s)$ is a formula with free variables among \vec{x}, s . These characterize the preconditions of the action A .
2. **Successor state axioms.** There is one for each relational fluent $F(\vec{x}, s)$, with syntactic form $F(\vec{x}, \text{do}(a, s)) \equiv \Phi_F(\vec{x}, a, s)$, where $\Phi_F(\vec{x}, a, s)$ is a formula with free variables among a, s, \vec{x} . These characterize the truth values of the fluent F in the next situation $\text{do}(a, s)$ in terms of the current situation s , and they embody a solution to the frame problem for deterministic actions (Reiter, 1991). There are similar axioms for functional fluents, but we shall not be using these in this paper, so we omit them.
3. **Unique names axioms for actions.** These state that the actions of the domain are pairwise unequal.
4. **Initial database.** This is a set of first order sentences whose only situation term is S_0 ; it specifies the initial state of the domain.

Example 1 We present axioms for a space platform domain that we use as an ongoing example throughout this paper. This consists of a single space arm, a dock, initially containing a collection of payloads, and a rotating platform divided into palettes. The arm can move payloads to and from palettes. As the platform rotates, its palettes periodically enter the shade and the sun. Each payload has associated with it an experiment that can be performed only while the payload is located on one of the palettes; moreover, the experimental condition for a given payload is one of *shade* or *sun*, meaning that the experiment must be performed while the payload is in the shade or sun, respectively, and this illumination must remain the same throughout the duration of the experiment. We axiomatize this domain using the following actions, fluents and predicates:

Primitive Actions:

All actions are understood to be instantaneous – they have no duration – and take, as their final argument, their occurrence time (Reiter, 1996).

- *startMove(pay, l₁, l₂, t)*: At time t , start the process of the space arm moving payload *pay* from location l_1 to location l_2 .
- *endMove(pay, l₁, l₂, t)*: At time t , terminate the process of the space arm moving payload *pay* from location l_1 to location l_2 .
- *enterSun(pal, t)*: At time t , palette *pal* enters the sun.
- *enterShade(pal, t)*: At time t , palette *pal* enters the shade.
- *startExpt(pay, t)*: At time t , start the process of performing the experiment for payload *pay*.

- $\text{endExpt}(\text{pay}, t)$: At time t , terminate the process of performing the experiment for payload pay .

Notice that some of these actions are associated with *processes*, and they come in pairs (e.g. $\text{startExpt}(\text{pay}, t)$ and $\text{endExpt}(\text{pay}, t)$), one of which initiates its process (e.g. an experiment in progress) and the other terminates the process. This device allows us to represent concurrent processes in the situation calculus using interleaving of instantaneous process initiating and terminating actions (Reiter, 1998). Processes themselves are represented as fluents, as we now see.

Fluents:

- $\text{inShade}(\text{pal}, t, s)$: Palette pal is in the shade in situation s , and t is the time at which it will next enter the sun.
- $\text{inSun}(\text{pal}, t, s)$: Palette pal is in the sun in situation s , and t is the time at which it will next enter the shade.
- $\text{moving}(\text{pay}, l_1, l_2, t_1, t_2, s)$: The arm is in the process of moving payload pay from location l_1 to location l_2 ; t_1 is the time this process was initiated, and t_2 is the time this process will terminate.
- $\text{exptInProgress}(\text{pay}, t_1, t_2, s)$: The experiment on payload pay is in progress; t_1 is the time this process was initiated, and t_2 is the time this process will terminate.
- $\text{location}(\text{pay}, \text{loc}, s)$: Payload pay is located at loc in situation s .

Action Precondition Axioms¹

$$\begin{aligned} \text{Poss}(\text{startMove}(\text{pay}, l_1, l_2, t), s) &\equiv \neg \text{armInUse}(s) \wedge \\ &\quad \text{location}(\text{pay}, l_1, s) \wedge (l_2 = \text{Dock} \vee \text{palette}(l_2) \wedge \neg \text{occupied}(l_2, s)). \\ \text{Poss}(\text{endMove}(\text{pay}, l_1, l_2, t), s) &\equiv (\exists t') \text{moving}(\text{pay}, l_1, l_2, t', t, s). \\ \text{Poss}(\text{enterSun}(\text{pal}, t), s) &\equiv \text{inShade}(\text{pal}, t, s). \\ \text{Poss}(\text{enterShade}(\text{pal}, t), s) &\equiv \text{inSun}(\text{pal}, t, s). \\ \text{Poss}(\text{startExpt}(\text{pay}, t), s) &\equiv \neg \text{exptCompleted}(\text{pay}, s) \wedge \\ &\quad \neg (\exists t_1, t_2) \text{exptInProgress}(\text{pay}, t_1, t_2, s) \wedge \\ &\quad (\exists \text{pal}). \text{palette}(\text{pal}) \wedge \text{location}(\text{pay}, \text{pal}, s) \wedge \\ &\quad (\text{needsSun}(\text{pay}) \wedge (\exists t')[\text{inSun}(\text{pal}, t', s) \wedge \\ &\quad \text{start}(s) + \text{exptDuration}(\text{pay}, s) \leq t'] \vee \\ &\quad \text{needsShade}(\text{pay}) \wedge \text{inSun}(\text{pal}, t, s) \vee \\ &\quad \text{needsShade}(\text{pay}) \wedge (\exists t')[\text{inShade}(\text{pal}, t', s) \wedge \\ &\quad \text{start}(s) + \text{exptDuration}(\text{pay}, s) \leq t'] \vee \\ &\quad \text{needsSun}(\text{pay}) \wedge \text{inShade}(\text{pal}, t, s)). \\ \text{Poss}(\text{endExpt}(\text{pay}, t), s) &\equiv (\exists t') \text{exptInProgress}(\text{pay}, t', t, s). \end{aligned}$$

Successor State Axioms

¹Throughout this paper, a character string beginning with a lower case Roman will denote a variable in a formula; a character string beginning with an upper case Roman denotes a constant. All formulas are implicitly universally quantified over their free variables.

$$\begin{aligned}
inShade(pal, t, do(a, s)) &\equiv (\exists t') [a = enterShade(pal, t') \wedge \\
&\quad t = t' + platformCycleTime/2] \vee \\
&\quad inShade(pal, t, s) \wedge \neg(\exists t') a = enterShade(pal, t'). \\
inSun(pal, t, do(a, s)) &\equiv (\exists t') [a = enterSun(pal, t') \wedge \\
&\quad t = t' + platformCycleTime/2] \vee \\
&\quad inSun(pal, t, s) \wedge \neg(\exists t') a = enterShade(pal, t'). \\
moving(pay, l_1, l_2, t_1, t_2, do(a, s)) &\equiv a = startMove(pay, l_1, l_2, t_1) \wedge \\
&\quad t_2 = t_1 + timeToPositionArm(l_1, s) + \\
&\quad timeToMovePayload(pay, l_1, l_2, s) \vee \\
&\quad moving(pay, l_1, l_2, t_1, t_2, s) \wedge a \neq endMove(pay, l_1, l_2, t_2). \\
location(pay, l, do(a, s)) &\equiv (\exists l', t) a = endMove(pay, l', l, t) \vee \\
&\quad location(pay, l, s) \wedge \neg(\exists l', t) a = startMove(pay, l, l', t). \\
exptInProgress(pay, t_1, t_2, do(a, s)) &\equiv a = startExpt(pay, t_1) \wedge \\
&\quad t_2 = t_1 + exptDuration(pay, s) \vee \\
&\quad exptInProgress(pay, t_1, t_2, s) \wedge a \neq endExpt(pay, t_2).
\end{aligned}$$

Initial Database

$$\begin{aligned}
start(S_0) &= 0. \\
(\forall p).location(p, Dock, S_0) &\equiv payload(p). \\
inShade(pal(0), 30, S_0). \quad inSun(pal(1), 20, S_0). \\
inShade(pal(2), 10, S_0). \\
(\forall p).payload(p) &\equiv p = Pay_1 \vee p = Pay_2 \vee p = Pay_3 \vee p = Pay_4. \\
(\forall p).palette(p) &\equiv p = pal(0) \vee p = pal(1) \vee p = pal(2). \\
needsSun(Pay_1). \quad needsSun(Pay_2). \quad needsShade(Pay_3). \\
needsShade(Pay_4). \quad platformCycleTime &= 60.
\end{aligned}$$

Abbreviations

$$\begin{aligned}
armInUse(s) &\stackrel{\text{def}}{=} (\exists pay, l, l', t, t') moving(pay, l, l', t, t', s). \\
occupied(l, s) &\stackrel{\text{def}}{=} (\exists pay) location(pay, l, s). \\
exptCompleted(pay, s) &\stackrel{\text{def}}{=} (\exists s', t) do(endExpt(pay, t), s') \sqsubseteq s. \\
timeToMovePayload(pay, l_1, l_2, s) &= 7. \\
timeToPositionArm(l, s) &= 5. \quad exptDuration(pay, s) = 15.^2
\end{aligned}$$

²To keep the axioms manageable for the purposes of this paper, we have simply specified constant times for positioning the arm and moving a payload, as well as for performing experiments on payloads. In general, these will depend on the payload, and the arm location, and therefore will be situation dependent. Nothing in the story we tell precludes richer axiomatizations for these quantities.

3 NATURAL ACTIONS AND EXECUTABLE SITUATIONS

Our focus in this paper is with planning in the presence of two kinds of actions:

1. Agent-performed actions, which are under the free will of an agent who can decide whether or not to perform an action, e.g. choosing whether or not to catch a falling ball.
2. *Natural* actions, under nature's control, which obey known physical laws. These must occur at their predicted times, provided no earlier actions (natural or free-will-agent initiated) prevent them from occurring. Nature has no free will; her actions must occur provided the time and circumstances for their occurrence are right.

Example 2 We continue with Example 1. The primitive action $startMove(pay, l_1, l_2, t)$ will be a free-will-agent action; the space arm gets to decide whether or not to initiate a payload moving process, and when. The remaining actions are all natural. Thus, $enterSun(pal, t)$ is natural because we know the "laws" of platform rotation, and can therefore predict exactly when each palette will enter the sun. $startExpt(pay, t)$ and $endExpt(pay, t)$ are also natural; a payload's experiment initiates (terminates) at predictable times, provided the payload is on a palette. Finally, $endMove(pay, l_1, l_2, t)$ is natural; provided the arm is in the process of moving pay from l_1 to l_2 , it is predictable, given the time this process initiated, exactly when the process will terminate.

3.1 EXECUTABLE SITUATIONS

A situation is a finite sequence of actions. There are no constraints on the actions entering into such a sequence, so that it may not be possible to actually execute these actions one after the other. For example, suppose the precondition for performing the action $putdown(x)$ in situation s is that the agent is holding x : $holding(x, s)$. Suppose, moreover, that in situation S_0 , the agent is not holding A : $\neg holding(A, S_0)$. Then $do(putdown(A), S_0)$ is a perfectly good situation, but its action is not executable; the precondition for performing $putdown(A)$ is violated in S_0 . Moreover, the actions in $do(pickup(B), do(putdown(A), S_0))$ cannot be executed in sequence either. In fact, no actions in a situation whose first action is $putdown(A)$ can be executed in sequence. Similarly, an action A_1 may be executable in S_0 , but the action A_2 may not be possible in $do(A_1, S_0)$, in which case no sequence of actions beginning with these two would be executable.

The fact that actions can have occurrence times places further constraints on whether an action sequence is executable. Consider the situation $do(bounce(B, W, 4), do(startMeeting(Susan, 6), S_0))$, in which the time of the second action precedes that of the first. We do not want to consider such an action sequence possible; it violates our intuitions about the forward progress of time.

We emphasize that action sequences in which an action violates its precondition, or in which the occurrence time of an action is less than the time of its preceding action, are perfectly good situations. They simply are not physically realizable; they are “ghost” situations.

Usually, we are interested only in non-ghostly situations. To characterize these, we introduce an abbreviation:

$$\text{feasible}(s) \stackrel{\text{def}}{=} (\forall a, s'). \text{do}(a, s') \sqsubseteq s \supset \text{Poss}(a, s') \wedge \text{start}(s') \leq \text{time}(a).$$

$\text{feasible}(s)$ means that all the actions in s can be executed one after the other, and their occurrence times are nondecreasing. Notice that we allow concurrency in the temporal domain; an action’s occurrence time is permitted to be the same as the time of its preceding action in a situation. This is desirable, for example, when one action can enable another, as in $\text{do}(\text{startFalling}(A, 10), \text{do}(\text{release}(A, 10), S_0))$.

So far, we have not considered natural actions; their presence adds a further constraint on the feasible situations. If the axioms for the natural action *bounce* predict that A will bounce at time 15 if A is released at time 10, then we should be suspicious of the following situation:

$$\text{do}(\text{startMeeting}(Susan, 20), \text{do}(\text{release}(A, 10), S_0)).$$

Somehow, the predicted natural action *bounce*($A, 15$) did not occur. So we want also to treat situations like this as ghosts. The following abbreviation captures the requirement that a natural action must occur at its predicted times whenever the conditions are right for it to occur:

$$\text{respectsNatOccurrences}(s) \stackrel{\text{def}}{=} \neg(\exists a, a', s'). \text{do}(a', s') \sqsubseteq s \wedge \text{natural}(a) \wedge \text{Poss}(a, s') \wedge \text{time}(a') > \text{time}(a).$$

Finally, we put these constraints together to capture formally our intuitive concept of the executable situations:

$$\text{executable}(s) \stackrel{\text{def}}{=} \text{feasible}(s) \wedge \text{respectsNatOccurrences}(s).$$

3.2 LEAST NATURAL TIME POINTS

The following concept of *least natural time point* plays a central role in theorizing about natural actions:

$$\begin{aligned} \text{lntp}(s, t) \stackrel{\text{def}}{=} & (\exists a)[\text{natural}(a) \wedge \text{Poss}(a, s) \wedge \text{time}(a) = t] \wedge \\ & (\forall a)[\text{natural}(a) \wedge \text{Poss}(a, s) \supset \text{time}(a) \geq t]. \end{aligned}$$

Intuitively, the least natural time point is the earliest time during situation s at which a natural action can occur.

Remark 1 *The following sentence is valid:³*

$$\text{lntp}(s, t) \wedge \text{lntp}(s, t') \supset t = t'.$$

³That is, valid over structures that give the real numbers and their ordering their standard interpretation.

So, when it exists, the least natural time point is unique. The least natural time point need not exist, for example, when $(\forall a).natural(a) \equiv (\exists x, t)a = B(x, t)$, where x ranges over the nonzero natural numbers, and $Poss(B(x, t), s) \equiv t = start(s) + 1/x$.

In view of the possibility of “pathological” axiomatizations, for which the least natural time point may not exist, we introduce the following sentence:

$$(\forall s).natActionPossible(s) \supset (\exists t).lntp(s, t). \quad (LNTPC)$$

Here,

$$natActionPossible(s) \stackrel{\text{def}}{=} (\exists a).natural(a) \wedge Poss(a, s).$$

Normally, it will be the responsibility of the axiomatizer to prove that her axioms entail *LNTPC*.

3.3 CHARACTERIZING THE EXECUTABLE SITUATIONS

Our purpose now is to characterize the executable situations in a way that leads naturally to a procedure for computing them.

Theorem 1 (Characterizing Executable Situations)

The following sentence is derivable from the foundational axioms for the temporal situation calculus (Reiter, 1998):

$$\begin{aligned} LNTPC \supset executable(do(a, s)) &\equiv executable(s) \wedge Poss(a, s) \wedge \\ &start(s) \leq time(a) \wedge \\ &[natActionPossible(s) \supset (\exists t).lntp(s, t) \wedge time(a) \leq t]. \end{aligned}$$

Theorem 1 is our central computational result. It tells us exactly how to extend an executable situation s to obtain an executable situation $do(a, s)$, provided the *LNTPC* is true: Pick any action a (natural or agent) with the following properties:

1. a 's preconditions are true in situation s .
2. a 's occurrence time is at least the same as the start time of s (which guarantees that the action sequence $do(a, s)$ will be monotone non-decreasing).
3. If a natural action is possible in s , then a 's occurrence time must not exceed the least natural time point of s (which exists, by *LNTPC*, and is unique by Remark 1).

Since S_0 is executable, we can inductively construct all executable situations in this way.

4 PLANNING WITH NATURAL ACTIONS IN THE SITUATION CALCULUS

The classical definition of planning is due to (Green, 1969).

Definition 1 Plans

Let \mathcal{D} be a background situation calculus axiomatization for some domain, and $G(s)$ a situation calculus formula – the *goal* – with one free situation variable s . A situation term $do(\alpha_n, do(\alpha_{n-1}, \dots, do(\alpha_1, S_0) \dots))$ that mentions no free variables is a *plan for G* iff

$$\mathcal{D} \models \text{feasible}(do(\alpha_n, do(\alpha_{n-1}, \dots, do(\alpha_1, S_0) \dots))) \wedge \\ G(do(\alpha_n, do(\alpha_{n-1}, \dots, do(\alpha_1, S_0) \dots))).^4$$

So on this definition, planning is a theorem-proving task: Determine a sequence $\alpha_1, \dots, \alpha_n$ of variable free action terms such that both

$$\text{feasible}(do(\alpha_n, do(\alpha_{n-1}, \dots, do(\alpha_1, S_0) \dots)))$$

and

$$G(do(\alpha_n, do(\alpha_{n-1}, \dots, do(\alpha_1, S_0) \dots)))$$

are provable from the background axioms \mathcal{D} . Green proposed this definition of planning for settings in which all agents have free will; it was not designed for natural actions. Fortunately, it is easy to modify it to provide for both free will and natural actions as follows:

A situation term $do(\alpha_n, do(\alpha_{n-1}, \dots, do(\alpha_1, S_0) \dots))$ that mentions no free variables is a *plan for G* iff

$$\mathcal{D} \models \text{executable}(do(\alpha_n, do(\alpha_{n-1}, \dots, do(\alpha_1, S_0) \dots))) \wedge \\ G(do(\alpha_n, do(\alpha_{n-1}, \dots, do(\alpha_1, S_0) \dots))).$$

This is the formal foundation for our planner.

4.1 GOLOG

Our planner is implemented in the situation calculus-based programming language GOLOG (Levesque et al., 1997), a language for defining complex actions in terms of a set of primitive actions axiomatized in the situation calculus according to Section 2.2. It has the standard – and some not so standard – control structures found in most Algol-like languages.

1. *Sequence*: $\alpha ; \beta$. Do action α , followed by action β .
2. *Test actions*: $p?$ Test the truth value of expression p in the current situation.
3. *Nondeterministic action choice*: $\alpha | \beta$. Do α or β .
4. *Nondeterministic choice of arguments*: $(\pi x)\alpha$. Nondeterministically pick a value for x , and for that value of x , do action α .
5. Conditionals (*if-then-else*) and *while* loops.
6. *Procedures, including recursion*.

GOLOG's formal semantics is specified by introducing an abbreviation $Do(\delta, s, s')$, where δ is a program and s and s' are situation terms. $Do(\delta, s, s')$

⁴Actually, Green's formulation of the planning problem was for the situation calculus without time, but our generalization here is minor.

is best viewed as a macro that expands into a situation calculus formula; moreover, *that formula says that situation s' can be reached from situation s by executing some sequence of actions specified by δ* . Note that our programs may be nondeterministic, that is, may have multiple executions terminating in different situations.

Do is defined inductively on the structure of its first argument as follows:

1. *Primitive actions*: If α is a primitive action term,

$$Do(\alpha, s, s') \stackrel{\text{def}}{=} Poss(\alpha, s) \wedge start(s) \leq time(\alpha) \wedge s' = do(\alpha, s).$$

2. *Test actions*: When ϕ is a situation-suppressed logical expression,

$$Do(\phi?, s, s') \stackrel{\text{def}}{=} \phi[s] \wedge s = s'.$$

Here, $\phi[s]$ denotes the result of restoring the situation argument s to all of the situation-suppressed fluents mentioned by ϕ .

3. *Sequence*:

$$Do(\delta_1; \delta_2, s, s') \stackrel{\text{def}}{=} (\exists s^*). Do(\delta_1, s, s^*) \wedge Do(\delta_2, s^*, s').$$

4. *Nondeterministic choice of two actions*:

$$Do(\delta_1 \mid \delta_2, s, s') \stackrel{\text{def}}{=} Do(\delta_1, s, s') \vee Do(\delta_2, s, s').$$

5. *Nondeterministic choice of action arguments*:

$$Do((\pi x) \delta, s, s') \stackrel{\text{def}}{=} (\exists x) Do(\delta, s, s').$$

There is also an abbreviation for procedures; it is rather more complicated than those just given, and we omit it. See (Levesque et al., 1997) for its details. Conditionals and while loops are definable with procedures and the above abbreviations.

$Do(program, s, s')$ is an abbreviation for a situation calculus formula saying that s' is one of the situations reached by evaluating the GOLOG *program*, beginning in situation s . Therefore, to execute *program*, one *proves*, using the situation calculus axiomatization of some background domain (e.g. the axioms of Example 1), the situation calculus sentence $(\exists s) Do(program, S_0, s)$. Any binding for s obtained by a constructive proof of this sentence is an execution trace, in terms of the primitive actions, of the *program*. A GOLOG interpreter in Prolog for the situation calculus with time, is surprisingly easy to implement (Levesque et al., 1997; Reiter, 1998).⁵ The following is the heart of our implementation:

Top-Level Prolog Clauses for a Golog Interpreter

```
do(E1 : E2, S, S1) :- do(E1, S, S2), do(E2, S2, S1).      % Sequence.
do(?(P), S, S) :- holds(P, S).                          % Test actions.
```

⁵With the very important caveat that the implementation inherits Prolog's closed world assumption to the effect that the background application domain axioms specify complete information about that domain's initial database. See (Reiter, 1999) for details.

```

do(E1 # E2,S,S1) :- do(E1,S,S1) ; do(E2,S,S1). % Nondeterministic choice.
do(if(P,E1,E2),S,S1) :- do(?(P) : E1 # ?(-P) : E2,S,S1). % Conditionals.
do(pi(V,E),S,S1) :- sub(V,_,E,E1), do(E1,S,S1). % Nondeterministic
    % choice of action argument. Generate a new Prolog variable,
    % and substitute it into the action E.
do(E,S,S1) :- proc(E,E1), do(E1,S,S1).           % Procedures.
do(E,S,do(E,S)) :- primitiveAction(E), poss(E,S), % Primitive actions.
    start(S,T1), time(E,T2), T1 <= T2.

```

4.2 A DEPTH-FIRST FORWARD PLANNER

The point of departure for implementing a planner for natural and free-will actions is Theorem 1, which characterizes the executable situations when *LNTPC* holds. This translates directly into the following GOLOG program for computing the executable situations of length n :

```

proc ex(n)
  n = 0? |
  n > 0? ;
  ( $\pi a$ )[(primitiveAction(a)  $\wedge$  [natActionPossible  $\supset$ 
    ( $\exists t$ )[lntp(t)  $\wedge$  time(a)  $\leq$  t]])? ; a] ;
  ex(n - 1)
endProc

```

Here, *primitiveAction* is a predicate characterizing what are all the primitive actions of the domain (natural, or under the agent's control). Like any GOLOG program, this is executed by proving

$(\exists s)Do(ex(n), S_0, s)$,

in our case, using the background axioms of Example 1. Therefore, we start with S_0 as the current situation. In general, if σ is the current situation, $ex(n)$ terminates in situation σ if $n = 0$; if $n > 0$, a primitive action a is selected (nondeterministically) to satisfy the program's big test action, the GOLOG interpreter checks that $Poss(a, \sigma) \wedge start(\sigma) \leq time(a)$ ⁶ and if so, a is “performed”, meaning that $do(a, \sigma)$ becomes the new current situation, and $ex(n - 1)$ is evaluated with $do(a, \sigma)$ as the current situation.

How can we turn this into a planner? The generalization of Green's Definition 1 of a plan requires us to find an executable situation that also satisfies the goal. The above program computes executable situations; to make it into a planner for some *goal* expression, modify it to return if *goal* is established, or the depth bound n is exceeded:

```

proc plan(n)
  goal? |
  n > 0? ;
  ( $\pi a$ )[(primitiveAction(a)  $\wedge$  [natActionPossible  $\supset$ 
    ( $\exists t$ )[lntp(t)  $\wedge$  time(a)  $\leq$  t]])? ; a] ;

```

⁶See the first abbreviation for *Do* in Section 4.1.

```

-badSituation? ;
plan(n - 1)
endProc

```

The only mystery in this planner is the test action $\neg\text{badSituation?}$. $\text{plan}(n)$ is meant to be a Bacchus-Kabanza style planner (Bacchus and Kabanza, 1995; Bacchus and Kabanza, 2000), so it expects the user to provide an axiomatization of a domain dependent predicate badSituation , used to filter out partial plans that are known a priori to be fruitless. In the next section we give examples of bad situations for the planning problem we treat in the space platform domain.

5 IMPLEMENTATION

We now describe how to implement the above planner and its supporting Golog interpreter and domain axioms (Example 1). As we shall see, these have a straightforward Prolog implementation with one exception: To execute the planner, the Golog interpreter must have a temporal reasoning component. For example, it must be able to infer that $T_1 = T_2$ when given that $T_1 \leq T_2 \wedge T_2 \leq T_1$. While such a special purpose temporal theorem prover could be written and included in the Golog interpreter, this would involve a great deal of effort. Instead, we rely on a logic programming language with a built-in constraint solver. Specifically, we appeal to the ECRC Common Logic Programming System ECLIPSE 3.5.2, which is equipped with a Simplex algorithm for solving linear equations and inequalities over the reals. So we shall assume, as is the case for the space platform Example 1 that our planner and domain axioms make use of linear temporal relations like $2 * T_1 + T_2 = 5$ and $3 * T_2 - 5 \leq 2 * T_3$, and we shall rely on ECLIPSE to perform the reasoning for us in the temporal domain. ECLIPSE provides a special syntax for those relations over the reals for which it provides a built-in theorem prover. These relations are: $=$, \neq , \geq , $>$, \leq , $<$, which are represented in ECLIPSE by the infix $\$=$, $\$<>$, $\$>=$, $\$>$, $\$<=$, $\$<$, respectively. So, in ECLIPSE, the above modification of the Golog interpreter of Section 4.1 to include time is:

```

do(E,S,do(E,S)) :- primitiveAction(E), poss(E,S),
    start(S,T1), time(E,T2), T1 \$<= T2.

```

All other clauses of the earlier Golog interpreter as given in Section 4.1 will work correctly under ECLIPSE.

We can now present the implementation of our planner.

A Sequential Planner for Free Will Agents with Natural Actions

```

proc(plan(N,
      ?(goal) : ?(reportStats) #
      ?(N > 0) :
      pi(a, ?(natActionPossible => some(t, lntp(t) &
                                              some(t1, time(a,t1) & t1 \$<= t))) :
      ?(primitiveAction(a)) : a) :
      ?(-badSituation) :

```

```

pi(n, ?(n is N - 1) : plan(n)).

% Least natural time point.

lntp(S,T) :- setof(N, (natural(N), poss(N,S)), Nacts), member(A,Nacts),
            time(A,T), leqAllNaturalActionTimes(T,Nacts), !.

leqAllNaturalActionTimes(T,[]).
leqAllNaturalActionTimes(T,[A | R]) :- time(A,T1), T $=< T1,
                                         leqAllNaturalActionTimes(T,R).

natActionPossible(S) :- natural(A), poss(A,S).

% Utilities.

prettyPrintSituation(S) :- makeActionList(S,Alist), nl, write(Alist), nl.

makeActionList(s0,[]).
makeActionList(do(A,S), L) :- makeActionList(S,L1), append(L1, [A], L).

askForMore :- write('More? '), read(n).

reportStats :- nl, cputime(T), write(' CPU time (sec): '),
              getval(cpu,T1), T2 is T - T1, write(T2), nl.

initializeCPU :- cputime(T), setval(cpu,T).

% Top-level call.

plan(N) :- initializeCPU, do(plan(N),s0,S), % Call GOLDG interpreter.
           prettyPrintSituation(S), askForMore.

```

The next task is to show how to implement the domain axioms of Example 1. But that part is easy; all axioms are in if-and-only-if form, so we simply appeal to the Clark completion semantics for Prolog (Lloyd, 1987) to justify translating each of these axioms into the corresponding Prolog clause for its if-half.⁷ In what follows, we give examples of this translation for some of the axioms of Example 1.

Examples of Clauses for the Axioms of the Space Platform Domain

```

poss(startMove(Pay,L1,L2,T),S) :- not armInUse(S), location(Pay,L1,S),
                                         (L2 = dock ; palette(L2), not occupied(L2,S)).

poss(startExpt(Pay,T),S) :- payload(Pay), not exptCompleted(Pay,S),
                           not exptInProgress(Pay,T1,T2,S), location(Pay,pal(N),S),

```

⁷The precise justification for this translation into Prolog takes a little bit of work; see (Reiter, 1999) for the details.

```

exptDuration(Pay,D,S),
( needsSun(Pay), inSun(pal(N),T1,S), start(S,T2), T2 + D $=< T1 ;
  needsShade(Pay), inSun(pal(N),T,S) ;
  needsShade(Pay), inShade(pal(N),T,S), start(S,T2), T2 + D $=< T1 ;
  needsSun(Pay), inShade(pal(N),T,S) ) .

inShade(Pal,T,do(A,S)) :- A = enterShade(Pal,T1),
                           platformCycleTime(K), T $= T1 + K/2 ;
                           inShade(Pal,T,S), not A = enterSun(Pal,T1).

exptInProgress(Pay,T1,T2,do(A,S)) :- A = startExpt(Pay,T1),
                                         exptDuration(Pay,D,S), T2 $= T1 + D ;
                                         exptInProgress(Pay,T1,T2,S), not A = endExpt(Pay,T2).

```

Finally, we need to specify the *badSituation* predicate used by the planner to filter out partial plans that are known to be useless. This is domain and problem specific. The planning goal we have in mind is that all payloads are on the dock, with their experiments completed:

$$(\forall p). payload(p) \supset location(p, Dock, s) \wedge exptCompleted(p, s).$$

The following are the bad situations we found useful for this goal:

Bad Situations for the Space Platform Problem

```

% Don't move a payload from the dock to the dock.
% Don't move a payload from a palette to a palette.

badSituation(do(startMove(Pay,dock,dock,T),S)).
badSituation(do(startMove(Pay,pal(M),pal(N),T),S)).

% Don't move a payload from the dock to a palette if the payload's
% experiment is completed.

badSituation(do(startMove(Pay,dock,pal(M),T),S)) :- exptCompleted(Pay,S).

% Don't move a payload from a palette if the payload's experiment is
% not completed.

badSituation(do(startMove(Pay,pal(M),L,T),S)) :- not exptCompleted(Pay,S).

% Heuristic rule: keep the arm busy.

badSituation(do(A,S)) :- not A = startMove(Pay,L1,L2,T),
                           poss(startMove(Pay,L1,L2,T),S).

% Don't move a payload to a palette if its experiment can't complete.

badSituation(S) :- lastAction(startMove(Pay,dock,pal(N),T),S),
                  simulateNatActionsUntil(endMove(Pay,dock,pal(N),T1),S,Sf), !,
                  not poss(startExpt(Pay,T2),Sf).

```

```

simulateNatActionsUntil(A,S1,S2) :- % Call Golog interpreter.
    do(simulateUntil(A),S1,S2).

proc(simulateUntil(A),
    if(lastAction(A), ?(true),
        /* ELSE */ pi(t, ?(lntp(t)) :
            pi(a, ?(natural(a) & time(a,t)) : a)) :
        simulateUntil(A)).

```

lastAction(A,do(A,S)).

For the most part, these bad situations are self-explanatory. The implementation of the last one is of interest for revealing some of the power of the situation calculus in a planning setting. s is judged a bad situation if its last action was to initiate a move process that, on termination, places the payload on a palette at such a time that its experiment cannot terminate. The problem is to determine what will be the state of the platform at the time the move process terminates. To find that out, we simulate the effect of all natural action occurrences (with the help of the GOLOG program *simulateUntil*) beginning with situation s , and ending when the move process terminates; the resulting situation determines the required platform state.

We can now exhibit the result of an execution of the planner.^{8,9} Recall that the planning goal is that all payloads are on the dock, with their experiments completed.

A Run of the Planner for the Initial Database of Example 2.1¹⁰

[eclipse 2]: plan(50).

CPU time (sec): 1.65

```

[ startMove(pay1,dock,pal(0),_359), enterSun(pal(2),10),
  endMove(pay1,dock,pal(0),_1646), startMove(pay2,dock,pal(1),_2252),
  enterShade(pal(1),20), startExpt(pay1,30), enterSun(pal(0),30),
  endMove(pay2,dock,pal(1),_6175), startMove(pay3,dock,pal(2),38),
  enterShade(pal(2),40), endExpt(pay1,45), endMove(pay3,dock,pal(2),50),
  startMove(pay1,pal(0),dock,50), startExpt(pay2,50), startExpt(pay3,50),
  enterSun(pal(1),50), enterShade(pal(0),60), endMove(pay1,pal(0),dock,62),
  startMove(pay4,dock,pal(0),62), endExpt(pay3,65), endExpt(pay2,65),
  enterSun(pal(2),70), endMove(pay4,dock,pal(0),74),
  startMove(pay3,pal(2),dock,74), startExpt(pay4,74),
  enterShade(pal(1),80), endMove(pay3, pal(2),dock,86),
  startMove(pay2,pal(1),dock,86), endExpt(pay4,89), enterSun(pal(0),90),

```

⁸All code necessary to run this example is available from the authors on request.

⁹Because the planner relies on Theorem 1, it presupposes that *LNTPC* holds for the space platform domain. That it does hold is straightforward to verify, and we omit the details.

¹⁰CPU times here are for a SUN Enterprise (Ultra) 450, with four 400MHZ processors and 4GB of RAM.

```

endMove(pay2,pal(1),dock,98), startMove(pay4,pal(0),dock,98),
enterShade(pal(2),100), endMove(pay4,pal(0),dock,110)]
More? n.

```

Linear Store:

```

_6175 $<= 32 + -1 * _S608
_2252 $<= 20 + -1 * _S608
_1646 $<= 20 + -1 * _S608 + -1 * _2325
_359 $<= 8 + -1 * _S608 + -1 * _2325

```

Notice that the plan contains uninstantiated temporal variables, whose possible values are subject to the inequalities in ECLIPSE's linear constraint store. To obtain a fully specified plan, one must determine instances of these, for example, by using the ECLIPSE `rmin` predicate.

The plan's duration is 110. This is not minimal; we know of several with duration 104, which we found by adding the following bad situation:

```
badSituation(S) :- start(S,T), not T $=< 105.
```

We also know of one with duration 96 that we found by running the planner on a suitable reordering of the payloads in the initial database. Even for such a small problem as this (4 payloads, 3 palettes), the search space is too large to explore completely, so we do not know whether 96 is minimal.

6 DISCUSSION

The problem we have considered – temporal planning with natural actions – seems not to have been extensively treated in the planning literature. See (Smith and Weld, 1999) for a discussion of their Graphplan-based TGP planner with exogenous events, and for a survey of other existing approaches. In contrast to these approaches, our planner is based on the conceptually straightforward ideas of Bacchus and Kabanza, and because it is also firmly grounded in mathematical logic, it has a transparency and conciseness lacking in most other planning formalisms. Moreover, it admits a simple implementation in ECLIPSE Prolog that allows one to exploit that system's constraint solving capabilities for temporal reasoning.

The particular space-platform problem on which we exhibited our planner is an instance of the general *resource constrained project scheduling problem* (RCPSP) (Brucker et al., 1999), which is concerned with allocating resources to dependent activities over time. In our problem, the resources are the palettes, sun, shade and the dock. The activities consist of moving a payload from the dock to a palette, starting an experiment with the right resource (sun or shade), and moving a payload from its palette to the dock on termination of its experiment. Examples of constraints between activities are that a payload cannot start its experiment if it is not on the palette with the right resource, that it cannot be put on a palette if the experiment cannot be concluded, and that it cannot be returned to the dock if the experiment is not finished. The general problem is to minimize the total activity time, subject to the constraints.

The RCPSP problem is a generalization of the classical job shop scheduling problem and therefore is *NP-hard* (Blazewicz et al., 1983). Accordingly, the literature proposes mixed heuristic approaches (See (Hartmann and Kolisch, 1998b) for a review). Most of these heuristic mechanisms can be naturally encoded in our *badSituation* predicate, and many of them are highly domain dependent (Hartmann and Kolisch, 1998a). Therefore, it seems that our Bacchus-Kabanza style planner is potentially well suited to solving scheduling problems of this kind. We, however, are not in the scheduling business, and we did not attempt to provide very sophisticated heuristic guidance to our planner for the space-platform problem. Accordingly, it returns feasible plans, but it doesn't try very hard to obtain optimal, or near optimal, solutions. Our primary purpose in this paper was to provide the theoretical and implementation foundations for a natural action planner, and we leave it to future research to determine whether planners like ours can be of service to the operations research community, or - which is closer to our interests - to the planning community.

Finally, we should point out that a wide variety of situation calculus planners have been implemented in the style of Bacchus and Kabanza. An open world planner for incomplete initial situations that uses regression and theorem proving is described in (Finzi et al., 2000). Several closed world planners have also been implemented (Reiter, 1999). These include planners for "classical" (totally ordered) problems, as well as for concurrency, and temporally ordered processes, as exemplified by a multi-handed blocks world agent.

References

- Bacchus, F. and Kabanza, F. (1995). Using temporal logic to control search in a forward chaining planner. In *Proceedings of the Third European Workshop on Planning*.
- Bacchus, F. and Kabanza, F. (2000). Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123–191.
- Blazewicz, J., Lenstra, J., and Kan, A. R. (1983). Scheduling subject to resource constraints. Classification and complexity. *Discrete Applied Mathematics*, 5:11–24.
- Brucker, P., Drexl, A., Mehring, R., Neumann, K., and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 29:262–273.
- Fikes, R. and Nilsson, N. (1971). STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4):189–208.
- Fikes, R. and Nilsson, N. (1993). STRIPS, a retrospective. *Artificial Intelligence*, 59(1/2):227–232.
- Finzi, A., Pirri, F., and Reiter, R. (2000). Open world planning in the situation calculus. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'00)*. To appear.
- Green, C. (1969). Theorem proving by resolution as a basis for question-answering systems. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 4*, pages 183–205. American Elsevier, New York.

- Hartmann and Kolisch, R. (1998a). Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. Invited paper for the 1998 INFORMS spring conference, Cincinnati, 1998.
- Hartmann and Kolisch, R. (1998b). Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In Weglarz, J., editor, *Recent Advances in Project Scheduling*. Kluwer, Amsterdam.
- Levesque, H., Pirri, F., and Reiter, R. (1998). Foundations for the situation calculus. *Linköping Electronic Articles in Computer and Information Science*, 3(18). <http://www.ep.liu.se/ea/cis/1998/018/>.
- Levesque, H., Reiter, R., Lespérance, Y., Lin, F., and Scherl, R. (1997). GOLOG: a logic programming language for dynamic domains. *J. of Logic Programming, Special Issue on Actions*, 31(1-3):59–83.
- Lloyd, J. (1987). *Foundations of Logic Programming*. Springer Verlag, second edition.
- McCarthy, J. (1963). Situations, actions and causal laws. Technical report, Stanford University. Reprinted in Semantic Information Processing (M. Minsky ed.), MIT Press, Cambridge, Mass., 1968, pp. 410-417.
- Pirri, F. and Reiter, R. (1999). Some contributions to the metatheory of the situation calculus. *Journal of the ACM*, 46(3):261–325.
- 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, San Diego, CA.
- Reiter, R. (1996). Natural actions, concurrency and continuous time in the situation calculus. In Aiello, L., Doyle, J., and Shapiro, S., editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96)*, pages 2-13. Morgan Kaufmann Publishers, San Francisco, CA.
- Reiter, R. (1998). Sequential, temporal GOLOG. In Cohn, A. and Schubert, L., editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, pages 547–556. Morgan Kaufmann Publishers, San Francisco, CA.
- Reiter, R. (1999). *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. In preparation. Draft available at <http://www.cs.toronto.edu/~cogrobo/>.
- Smith, D. and Weld, D. (1999). Temporal planning with mutual exclusion reasoning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 326–333, Stockholm, Sweden.

Chapter 11

REINVENTING SHAKEY

Murray Shanahan

Imperial College

Department of Electrical and Electronic Engineering

Exhibition Road

London SW7 2BT

England.

m.shanahan@ic.ac.uk

Abstract This paper describes the logical foundations of an implemented system which employs resolution-based theorem proving techniques for high-level robot control. The paper offers complementary logical characterizations of perception and planning, and shows how sensing, planning and acting are interleaved to control a real robot.

Keywords: Cognitive robotics, reasoning about action

1 INTRODUCTION

In the late Sixties, when the Shakey project started (Nilsson, 1984), the vision of robot design based on logical representation seemed both attractive and attainable. Through the Seventies and early Eighties, however, the desire to build working robots led researchers away from logic to more practical but *ad hoc* approaches to representation. This movement away from logical representation reached an extreme in the late Eighties and early Nineties when Brooks jettisoned the whole idea of representation, along with the so-called sense-model-plan-act architecture epitomized by Shakey, (Brooks, 1991).

However, the Shakey style of architecture, having an overtly logic-based deliberative component, seems to offer researchers a direct path to robots with high-level cognitive skills, such as planning, reasoning about other agents, and communication with other agents. Accordingly, a number of researchers have instigated a Shakey revival, and are aiming to achieve robots with these sorts of high-level cognitive skills by using logic as a representational medium (Lespérance et al., 1994).

This paper describes one such robot. The paper concentrates on logical foundations, but everything described has been implemented, deployed, and

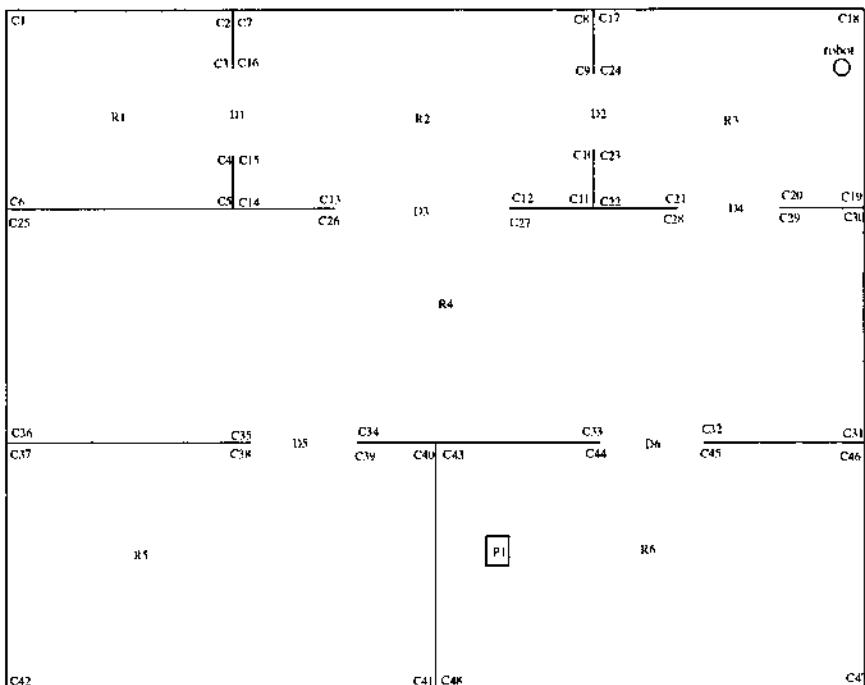


Figure 11.1 The Robot's Environment

tested on a real robot, namely a Khepera, a miniature robot with two drive wheels and a suite of eight infra-red proximity sensors around its circumference. The robot inhabits a miniaturized office-like environment, depicted in Figure 11.1.

The robot has a simple repertoire of low-level actions, executed on-board, which includes wall-following, turning into doorways, and turning around corners. Using these, the high-level, off-board controller maneuvers the robot around its environment.

1.1 A MOTIVATING EXAMPLE

To get a feel for its capabilities, let's take a look at an example of the way in which the high-level controller functions. Suppose the robot is initially between corners C18 and C19, as shown, and suppose it has the goal of retrieving package P1 from room R6. Informally, this is how the robot's high-level controller achieves the goal.

First, the robot plans a route. As soon as it finds a complete (though perhaps not fully decomposed) plan with an executable first action, the robot starts carrying that plan out. In this case, the first action is to go through door

D4, so the robot sets out along the wall until it reaches corner C19. It then turns the corner, and heads off towards door D4.

Suppose someone now closes door D4. Unfortunately, because of its poor sensors, the robot cannot detect closed doors, which are indistinguishable from walls. So the robot continues wall-following until it reaches corner C22.

Up to this point, the assimilation of the robot's sensor data has been a trivial matter. The sensor events it receives are exactly what it would expect given what it has done and what it believes about its environment. So the explanations of those sensor events are trivial. But the encounter with a corner at this time requires a non-empty explanation.

Using abduction, the robot constructs an explanation of its encounter with the corner door D4 must have been closed, and it must now be at corner C22. But this new piece of information conflicts with the assumptions underlying the plan it is executing. So the robot is forced to replan. It now finds a new route to room R6, via doors D2, D3, and D6, which it successfully executes and retrieves the package.

In the implemented system, the robot is controlled by a sense-plan-act cycle. Sensing and planning are both resolution-based abductive theorem proving processes, and in each iteration of the cycle, the sense phase and plan phase carry out a single step of resolution each. The chief aim of the rest of this paper is to set out the logical foundations of these planning and perception processes.

The paper presents,

- a short introduction to the event calculus, a formalism for reasoning about action,
- a logical account of abductive event calculus planning,
- a complementary logical account of abductive sensor data assimilation, also based on the event calculus, and,
- a brief outline of event calculus hierarchical planning, through which plans can be generated in progression order.

2 REPRESENTING ACTION

To supply the required logical accounts of perception and planning, a formalism for reasoning about action is needed. The formalism presented here is based on the circumscriptive event calculus (Shanahan, 1997c), (Shanahan, 1999). The event calculus is a well-established logical formalism for reasoning about actions. It is capable of representing a wide variety of phenomena, including actions with indirect effects, non-deterministic actions, concurrent actions, and continuous change (Shanahan, 1999). A robust solution to the frame problem exists for the event calculus, that works in the presence of all of these phenomena (Shanahan, 1997c).

Because this material is presented in considerable detail elsewhere, the description here will be kept fairly brief. A many sorted language is assumed, with variables for fluents, actions (or events), and time points. We have the following axioms, whose conjunction will be denoted *EC*. Their main purpose

is to constrain the predicate *HoldsAt*. $HoldsAt(f, t)$ represents that fluent f holds at time t . Throughout the paper, all variables are universally quantified with maximum scope, unless otherwise indicated.

$$HoldsAt(f, t) \leftarrow Initially_P(f) \wedge \neg Clipped(0, f, t) \quad (11.1)$$

$$\begin{aligned} HoldsAt(f, t3) \leftrightarrow \\ Happens(a, t1, t2) \wedge Initiates(a, f, t1) \wedge \\ t2 < t3 \wedge \neg Clipped(t1, f, t3) \end{aligned} \quad (11.2)$$

$$\begin{aligned} Clipped(t1, f, t4) \leftarrow \\ \exists a, t2, t3 [Happens(a, t2, t3) \wedge t1 < t3 \wedge t2 < t4 \wedge \\ [Terminates(a, f, t2) \vee Releases(a, f, t2)]]] \end{aligned} \quad (11.3)$$

$$\neg HoldsAt(f, t) \leftarrow Initially_N(f) \wedge \neg Declipped(0, f, t) \quad (11.4)$$

$$\begin{aligned} \neg HoldsAt(f, t3) \leftarrow \\ Happens(a, t1, t2) \wedge Terminates(a, f, t1) \wedge \\ t2 < t3 \wedge \neg Declipped(t1, f, t3) \end{aligned} \quad (11.5)$$

$$\begin{aligned} Declipped(t1, f, t4) \leftrightarrow \\ \exists a, t2, t3 [Happens(a, t2, t3) \wedge t1 < t3 \wedge t2 < t4 \wedge \\ [Initiates(a, f, t2) \vee Releases(a, f, t2)]]] \end{aligned} \quad (11.6)$$

$$Happens(a, t1, t2) \rightarrow t1t2 \quad (11.7)$$

A particular domain is described in terms of *Initiates*, *Terminates*, and *Releases* formulae. $Initiates(a, f, t)$ represents that fluent f starts to hold after action a at time t . Conversely, $Terminates(a, f, t)$ represents that f ceases to hold after action a at t . $Releases(a, f, t)$ represents that fluent f is no longer subject to the common sense law of inertia after action a at t .

A particular narrative of events is described in terms of *Happens* and *Initially* formulae. The formulae $Initially_P(f)$ and $Initially_N(f)$ respectively represent that fluent f holds at time 0 and does not hold at time 0. $Happens(a, t1, t2)$ represents that action or event a occurs, starting at time $t1$ and ending at time $t2$. Table 1 summarizes the predicates of the calculus.

A two-argument version of Happens is defined as follows.

$$Happens(a, t) \equiv_{def} Happens(a, t, t)$$

Formula	Meaning
$Initiates(a, b, t)$	Fluent b starts to hold after action a at time t
$Terminates(a, b, t)$	Fluent b ceases to hold after action a at time t
$Releases(a, b, t)$	Fluent b is no longer subject to the common sense law of inertia after action a at time t
$Initially_P(b)$	Fluent b holds from time 0
$Initially_N(b)$	Fluent b does not hold from time 0
$t_1 < t_2$	Time point t_1 is before time point t_2
$Happens(a, t)$	Action a occurs at time t
$Happens(a, t_1, t_2)$	Action a starts at time t_1 and ends at time t_2
$HoldsAt(b, t)$	Fluent b holds at time t
$Clipped(t_1, b, t_2)$	Fluent b is terminated between times t_1 and t_2
$Declipped(t_1, b, t_2)$	Fluent b is initiated between times t_1 and t_2

Table 11.1 Event Calculus Predicates

Formulae describing triggered events are allowed, and will generally have the form,

$$Happens(a, t) \leftarrow \Pi.$$

As we'll see in Section 5, similar formulae can be used to define compound actions, which are used for hierarchical planning.

The frame problem is overcome through circumscription. Given a conjunction Σ of *Initiates*, *Terminates*, and *Releases* formulae describing the effects of actions, a conjunction Δ of *Initially*, *Happens* and temporal ordering formulae describing a narrative of actions and events, and a conjunction Ω of uniqueness-of-names axioms for actions and fluents, we're interested in,

$$\begin{aligned} CIRC[\Sigma; \textit{Initiates}, \textit{Terminates}, \textit{Releases}] \wedge \\ CIRC[\Delta; \textit{Happens}] \wedge EC \wedge \Omega. \end{aligned} \tag{11.8}$$

By minimizing *Initiates*, *Terminates* and *Releases* we assume that actions have no unexpected effects, and by minimizing *Happens* we assume that there are no unexpected event occurrences. In many of the cases we're interested in, Σ and Δ will be conjunctions of Horn clauses, and the circumscriptions will reduce to predicate completions.

Care must be taken when domain constraints and triggered events are included. Domain constraints (or "state constraints") are formulae that constrain the combination of fluents that can hold simultaneously, and are used to define actions with indirect effects. Triggered events are events that

occur when some specified combination of fluents holds. The former must be conjoined to EC , while the latter are conjoined to Δ .

3 A LOGICAL ACCOUNT OF PLANNING

Planning can be thought of as the inverse operation to temporal projection, that is to say reasoning forwards in time from causes to effects. Temporal projection in the event calculus is naturally cast as a deductive task. Given Σ , Ω and Δ as above, we're interested in *HoldsAt* formulae Γ such that,

$$\begin{aligned} CIRC[\Sigma; \text{Initiates}, \text{Terminates}, \text{Releases}] \wedge \\ CIRC[\Delta; \text{Happens}] \wedge EC \wedge \Omega \models \Gamma. \end{aligned} \quad (11.9)$$

Conversely, planning in the event calculus can be considered as an abductive task, since it is a form of reasoning from effects to causes. Given a domain description Σ , a conjunction Γ of goals (*HoldsAt* formulae), and a conjunction Δ_N of *Initially_P* and *Initially_N* formulae describing the initial situation, a plan is a consistent conjunction Δ_P of *Happens* and temporal ordering formulae such that,

$$\begin{aligned} CIRC[\Sigma; \text{Initiates}, \text{Terminates}, \text{Releases}] \wedge \\ CIRC[\Delta_N \wedge \Delta_P; \text{Happens}] \wedge EC \wedge \Omega \models \Gamma. \end{aligned} \quad (11.10)$$

The following formulae capture the connectivity of the rooms, as shown in Figure 11.1.

$$Connects(D1, R1, R2) \quad (11.11)$$

$$Connects(D2, R2, R3) \quad (11.12)$$

$$Connects(D3, R2, R4) \quad (11.13)$$

$$Connects(D4, R3, R4) \quad (11.14)$$

$$Connects(D5, R4, R5) \quad (11.15)$$

$$Connects(D6, R4, R6) \quad (11.16)$$

$$Connects(d, r1, r2) \leftarrow Connects(d, r2, r1) \quad (11.17)$$

Let's suppose the robot can perform only one action, which is to go through a specified door d , denoted by the term *GoThrough*(d). (In the implemented system, the *GoThrough* action is broken down by hierarchical planning into a sequence of *FollowWall* and *Turn* actions.) We'll assume, for now, that doors are always open. The only fluent in the domain is *InRoom*(r) representing that the robot is in room r . We have the following *Initiates* and *Terminates* formulae.

$$\begin{aligned} \text{Initiates}(\text{GoThrough}(d), \text{InRoom}(r1), t) \leftarrow \\ Connects(d, r2, r1) \wedge \text{HoldsAt}(\text{InRoom}(r2), t) \end{aligned} \quad (11.18)$$

$$\begin{aligned} \text{Terminates}(GoThrough(d), InRoom(r), t) \leftarrow \\ \quad \text{HoldsAt}(InRoom(r), t) \end{aligned} \quad (11.19)$$

Since there is only one action and only one fluent, this example doesn't require any uniqueness-of-names axioms.

Suppose the robot is initially in room $R3$.

$$Initially_P(InRoom(R3)) \quad (11.20)$$

The goal is to get the robot to room $R6$.

$$HoldsAt(InRoom(R6), T) \quad (11.21)$$

Clearly one plan for achieving the goal is to go through $D4$ then go through $D6$.

$$Happens(GoThrough(D4), T1) \quad (11.22)$$

$$Happens(GoThrough(D6), T2) \quad (11.23)$$

$$T1 < T2 \quad (11.24)$$

$$T2 < T \quad (11.25)$$

The fact that this is a plan according to the abductive definition is expressed in the following proposition.

Proposition 1 Let,

- Σ be the conjunction of (11.18) and (11.19),
- Δ_N be formula (11.20),
- Δ_P be the conjunction of (11.22) to (11.25),
- Ψ be the conjunction of (11.11) to (11.17), and
- Γ be formula (11.21).

We have,

$$\begin{aligned} CIRC[\Sigma; Initiates, Terminates, Releases] \wedge \\ CIRC[\Delta_N \wedge \Delta_P; Happens] \wedge EC \wedge \Omega \models \Gamma. \end{aligned}$$

3.1 IMPLEMENTATION

This account of planning can be implemented through abductive logic programming, as described in (Shanahan, 1997a) and (Shanahan, 2000). The implementation is an abductive meta-interpreter, which has been tailored for the event calculus. *HoldsAt* goals are treated in a special way, and the sub-goals they generate are processed in a particular order to prevent looping. In addition, a dedicated constraint solver is applied to temporal ordering constraints, whose efficiency is improved by the maintenance of a cache of temporal ordering lemmas.

The computation carried out by the resulting system strongly resembles that of a hand-coded partial-order planning algorithm. (The present example of route planning can be reduced to graph search, but for general purpose planning, we need a more generic technique.) In particular, the implementation has to record negated *Clipped* formulae that it has proved, and these correspond to protected links in partial-order planning terminology. A protected link records the fact that the value of a fluent initiated or terminated by a given action must be preserved until the occurrence of another subsequent action whose preconditions depend on that fluent.

The efficiency of this planner is in line with that of a partial order planner. A more recent event calculus planner, based on the planning as satisfiability work of (Kautz and Selman, 1996), is more efficient, but has not yet been applied to robotics (Shanahan and Witkowski, 2000). However, as we'll see later, this is only part of the story for planning. In order to be able to interleave planning, sensing and acting in a respectable way, we need to carry out hierarchical planning. Moreover, by using hierarchical planning wherever possible, we minimize the use of search-heavy planning from first principles.

Now, what exactly is the relationship between the logical specification of planning and its implementation by means of abductive logic programming? This question is addressed more fully in (Shanahan, 2000). But, in outline, here is the answer.

In (Shanahan, 2000), a class of event calculus domain descriptions is defined which, among other restrictions, confines *Initiates* and *Terminates* formulae to the Horn clause subset. This means that the theorems of (Lifschitz, 1994) can be applied to reduce the circumscriptions of these formulae to predicate completion. Moreover, since these restrictions rule out recursion, predicate completion and SLDNF coincide for the class of theories in question. Thus the prospect of a straightforward logic programming implementation is brought closer.

However, the use of temporal ordering constraints whose completions we can't assume complicates the issue. In effect, the abductive meta-interpreter treats temporal constraints in a special way, using a dedicated constraint solver.

For the restricted class of event calculus theories defined, the abductive meta-interpreter of (Shanahan, 2000) is both sound and complete with respect to the abductive characterization of planning. This meta-interpreter forms the basis of both the planning and sensor data assimilation components in the present work.

However, many useful domain descriptions fall outside the scope of these theorems. Domain descriptions including formulae describing compound actions are an example of particular relevance to the present paper, since they form the basis of hierarchical planning. In the presence of such formulae, the soundness and completeness of the meta-interpreter are conjectural. So there is a theoretical gap that needs to be filled here, and this is the subject of ongoing work.

Next, we'll look into the topic of perception.

4 A LOGICAL ACCOUNT OF PERCEPTION

This section offers a logical account of sensor data assimilation (perception) which mirrors the logical account of planning in Section 3. The need for such an account arises from the fact that sensors do not deliver facts directly into the robot's model of the world. Rather they provide raw data from which facts can be inferred.

The methodology for supplying the required logical account is as follows (Shanahan, 1997b). First, using a suitable formalism for reasoning about actions, construct a theory Σ of the effects of the robot's actions on the world and the impact of the world on the robot's sensors. Second, consider sensor data assimilation as abduction with this theory. Roughly speaking, given a narrative Δ of the robot's actions, and a description Γ of the robot's sensor data, the robot needs to find some Ψ such that,

$$\Sigma \wedge \Delta \wedge \Psi \models \Gamma.$$

In event calculus terms, Γ might comprise *Happens* and/or *HoldsAt* formulae describing sensor events or values, and Ψ might comprise *Initially_N* and *Initially_P* formulae describing the environment's initial configuration and/or *Happens* formulae describing the intervening actions of other agents which have modified that configuration.

To illustrate this, we'll stay with the office delivery domain. But we must begin with a look at the sensory capabilities of the Khepera robots which are being used to test the ideas presented in this paper.

The Khepera can be straightforwardly programmed to navigate around the environment of Figure 11.1. Using its proximity sensors, it can follow walls and detect inner and outer corners. If all the doors are open, the *GoThrough* action of Section 3 can be executed, assuming the robot's initial location is known, by counting inner and outer corners until the robot reaches the required door, then passing through it.

If any of the doors is closed, however, this approach to executing the *GoThrough* action will fail, because the infra-red proximity sensors cannot detect a closed door, which looks to them just like a continuation of the wall.

This, in an extreme form, is the predicament facing any perceptual system. Inference must be carried out on raw sensor data in order to produce knowledge. In this case, the robot can abduce the fact that a door is closed as the only possible explanation of its unexpected arrival at an inner corner instead of the outer corner of the doorway.

Our aim here is to give a formal account of this sort of inference that gels with the formal account of planning already supplied. Indeed, in the implemented system, the same knowledge, expressed using the same formalism, is used for both planning and sensor data assimilation. Furthermore, as already emphasised, both planning and sensor data assimilation are viewed as abductive tasks with a very similar character. This means that the same abductive logic programming technology, indeed the very same code, can be used to implement both processes.

4.1 THE ROBOT'S ENVIRONMENT

Returning to the example at hand, the representation of the robot's environment, as depicted in Figure 11.1, now needs to include corners, which were neglected in the planning example. The formula $\text{NextCorner}(r, c1, c2)$ represents that corner $c2$ is the next inner or outer corner in room r after corner $c1$, in a clockwise direction. For room $R1$ alone, we have the following formulae.

$$\text{NextCorner}(R1, C1, C2) \quad (11.26)$$

$$\text{NextCorner}(R1, C2, C3) \quad (11.27)$$

$$\text{NextCorner}(R1, C3, C4) \quad (11.28)$$

$$\text{NextCorner}(R1, C4, C5) \quad (11.29)$$

$$\text{NextCorner}(R1, C5, C6) \quad (11.30)$$

$$\text{NextCorner}(R1, C6, C1) \quad (11.31)$$

In addition, the formula $\text{Door}(d, c1, c2)$ represents that there is a doorway between the two corners $c1$ and $c2$. For each door, there will be a pair of such formulae. Here they are for door $D1$.

$$\text{Door}(D1, C3, C4) \quad (11.32)$$

$$\text{Door}(D1, C15, C16) \quad (11.33)$$

Finally, the formulae $\text{Inner}(c)$ and $\text{Outer}(c)$ represent respectively that c is an inner corner and c is an outer corner. Again confining our attention to room $R1$, we have the following.

$$\text{Inner}(C1) \quad (11.34)$$

$$\text{Inner}(C2) \quad (11.35)$$

$$\text{Outer}(C3) \quad (11.36)$$

$$\text{Outer}(C4) \quad (11.37)$$

$$\text{Inner}(C5) \quad (11.38)$$

$$\text{Inner}(C6) \quad (11.39)$$

Each of these predicates will need to be minimized using circumscription, so that their completions are formed.

4.2 THE ROBOT'S EFFECT ON THE WORLD

Now we can formalize the effects of the robot's actions on the world. To simplify the example, the following formulae assume the robot always hugs the left wall, although parameters are provided which allow for it to hug the right wall as well.

Again, a finer grain of detail is required than for the planning example. Instead of a single *GoThrough* action, the robot's repertoire now comprises three actions: *FollowWall*, *Turn(s)*, and *GoStraight*, where *s* is either *Left* or *Right*. These actions affect three fluents. The fluent *AtCorner(c, s)* holds if the robot is at (inner or outer) corner *c*, with *c* in direction *s*, where *s* is *Left* or *Right*. The fluent *BesideWall(w, s)* holds if the robot is adjacent to wall *w* in direction *s*, where *s* is *Left* or *Right*. The fluent *InDoorway(d, r)* holds if the robot is in doorway *d*, with its back to room *r*. (By convention, the three fluents are mutually exclusive.)

Let's formalize the effects of the three actions in turn. Each action is assumed to be instantaneous, an assumption which has no practical implications in the present example. The term *Wall(c1, c2)* denotes the wall between corners *c1* and *c2*. First, if the robot follows a wall, it ends up at the next visible corner.

$$\begin{aligned} \text{Initiates}(FollowWall, AtCorner(c3, Left), t) \leftarrow \\ \text{HoldsAt}(BesideWall(Wall(c1, c2), Left), t) \wedge \\ \text{NextVisibleCorner}(c1, c3, Left, t) \end{aligned} \quad (11.40)$$

$$\text{Terminates}(FollowWall, BesideWall(w, s), t) \quad (11.41)$$

The formulae *NextVisibleCorner(c1, c2, s, t)* means that, at time *t*, *c2* is the next visible corner after *c1*, where the wall in question is in direction *s*. The corner of a doorway whose door is closed is invisible.

$$\begin{aligned} \text{NextVisibleCorner}(c1, c2, Left, t) \leftarrow \\ \text{NextCorner}(r, c1, c2) \wedge \neg \text{InvisibleCorner}(c2, t) \end{aligned} \quad (11.42)$$

$$\begin{aligned} \text{NextVisibleCorner}(c1, c3, Left, t) \leftarrow \\ \text{NextCorner}(r, c1, c2) \wedge \text{InvisibleCorner}(c2, t) \wedge \\ \text{NextVisibleCorner}(c2, c3, Left, t) \end{aligned} \quad (11.43)$$

$$\begin{aligned} [\text{NextVisibleCorner}(c1, c2, s, t) \wedge \\ \text{NextVisibleCorner}(c1, c3, s, t)] \rightarrow c2 = c3 \end{aligned} \quad (11.44)$$

$$\begin{aligned}
 & \text{InvisibleCorner}(c1, t) \leftarrow \\
 & \exists d, c2 [[\text{Door}(d, c1, c2) \vee \text{Door}(d, c2, c1)] \wedge \\
 & \neg \text{HoldsAt}(\text{DoorOpen}(d), t)]
 \end{aligned} \tag{11.45}$$

Next we have the *GoStraight* action, which the robot executes to bypass a doorway, traveling in a straight line from the near corner of the doorway and coming to rest when it detects the far corner.

$$\begin{aligned}
 & \text{Initiates}(\text{GoStraight}, \\
 & \text{BesideWall}(\text{Wall}(c2, c3), \text{Left}), t) \leftarrow \\
 & \text{HoldsAt}(\text{AtCorner}(c1, \text{Left}), t) \wedge \\
 & \text{Door}(d, c1, c2) \wedge \text{NextCorner}(r, c2, c3)
 \end{aligned} \tag{11.46}$$

$$\text{Terminates}(\text{GoStraight}, \text{AtCorner}(c, s), t) \tag{11.47}$$

Finally we have the *Turn* action. Since the robot has to hug the left wall, it always turns left (or goes straight) at outer corners, and always turns right at inner corners. If it turns left at the corner of a doorway, it ends up in the doorway.

$$\begin{aligned}
 & \text{Initiates}(\text{Turn}(\text{Left}), \text{InDoorway}(d, r), t) \leftarrow \\
 & \text{HoldsAt}(\text{AtCorner}(c1, \text{Left}), t) \wedge \text{Door}(d, c1, c2) \wedge \\
 & \text{HoldsAt}(\text{DoorOpen}(d), t) \wedge \text{NextCorner}(r, c1, c2)
 \end{aligned} \tag{11.48}$$

If the robot turns left when in a doorway, it ends up alongside a wall in the next room.

$$\begin{aligned}
 & \text{Initiates}(\text{Turn}(\text{Left}), \\
 & \text{BesideWall}(\text{Wall}(c2, c3), \text{Left}), t) \leftarrow \\
 & \text{HoldsAt}(\text{InDoorway}(d, r1), t) \wedge \text{Connects}(d, r1, r2) \wedge \\
 & \text{Door}(d, c1, c2) \wedge \text{NextCorner}(r2, c2, c3)
 \end{aligned} \tag{11.49}$$

If the robot turns right at an inner corner, it ends up next to a new wall.

$$\begin{aligned}
 & \text{Initiates}(\text{Turn}(\text{Right}), \\
 & \text{BesideWall}(\text{Wall}(c1, c2), \text{Left}), t) \leftarrow \\
 & \text{HoldsAt}(\text{AtCorner}(c1, \text{Left}), t) \wedge \\
 & \text{Inner}(c1) \wedge \text{NextCorner}(r, c1, c2)
 \end{aligned} \tag{11.50}$$

The mutual exclusivity of the *AtCorner*, *InDoorway* and *BesideWall* fluents is preserved by the following formulae.

$$\text{Terminates}(\text{Turn}(s1), \text{AtCorner}(c, s2), t) \quad (11.51)$$

$$\begin{aligned} \text{Terminates}(\text{Turn}(s), \text{InDoorway}(d, r), t) \leftarrow \\ \text{HoldsAt}(\text{InDoorway}(d, r), t) \end{aligned} \quad (11.52)$$

4.3 THE EFFECT OF THE WORLD ON THE ROBOT

Having axiomatized the effects of the robot's actions on the world, now we need to formalize the impact the world has on the robot's sensors. For this purpose, we introduce two new types of event. The event *GoesHigh*(*s*) occurs if the average value of the two sensors in direction *s* exceeds a threshold δ_1 , where *s* is *Left*, *Right* or *Front*. Similarly the event *GoesLow*(*s*) occurs if the average value of the two sensors in direction *s* goes below a threshold δ_2 . (By making $\delta_1 > \delta_2$, we avoid a chatter of *GoesHigh* and *GoesLow* events when the robot approaches an obstacle.)

$$\begin{aligned} \text{Happens}(\text{GoesHigh}(\text{Front}), t) \leftarrow \\ \text{Happens}(\text{FollowWall}, t) \wedge \\ \text{Initiates}(\text{FollowWall}, \text{AtCorner}(c, s), t) \wedge \text{Inner}(c) \end{aligned} \quad (11.53)$$

$$\begin{aligned} \text{Happens}(\text{GoesLow}(\text{Front}), t) \leftarrow \\ \text{HoldsAt}(\text{AtCorner}(c, \text{Left}), t) \wedge \\ \text{Inner}(c) \wedge \text{Happens}(\text{Turn}(\text{Right}), t) \end{aligned} \quad (11.54)$$

$$\begin{aligned} \text{Happens}(\text{GoesHigh}(s), t) \leftarrow \\ \text{HoldsAt}(\text{AtCorner}(c, s), t) \wedge \text{Outer}(c) \wedge \\ [\text{Happens}(\text{GoStraight}, t) \vee \text{Happens}(\text{Turn}(s), t)] \end{aligned} \quad (11.55)$$

$$\begin{aligned} \text{Happens}(\text{GoesLow}(s), t) \leftarrow \\ \text{Happens}(\text{FollowWall}, t) \wedge \\ \text{Initiates}(\text{FollowWall}, \text{AtCorner}(c, s), t) \wedge \text{Outer}(c) \end{aligned} \quad (11.56)$$

Our overall aim, of course, is to use abduction to explain the occurrence of *GoesHigh* and *GoesLow* events. In the present example, if the doors are all initially open and never subsequently closed, every sensor event is predicted by the theory as it stands, so no explanation is required. The interesting case is where there are sensor events which can only be explained by a closed door.

Accordingly, we need to introduce the events $OpenDoor(d)$ and $CloseDoor(d)$, with the obvious meanings.

$$Initiates(OpenDoor(d), DoorOpen(d), t) \quad (11.57)$$

$$Terminates(CloseDoor(d), DoorOpen(d), t) \quad (11.58)$$

Finally, we need some uniqueness-of-names axioms.

$$UNA[FollowWall, GoStraight, Turn, \\ GoesHigh, GoesLow, OpenDoor, CloseDoor] \quad (11.59)$$

$$UNA[BesideWall, AtCorner, InDoorway, DoorOpen] \quad (11.60)$$

The UNA notation is defined as follows. Let f_1 to f_k be function symbols. Then $UNA[f_1, f_2, \dots, f_k]$ abbreviates the conjunction of the formulae,

$$f_i(x_1, x_2, \dots, x_m) \neq f_j(y_1, y_2, \dots, y_n)$$

for all $i < j < k$, and,

$$f_i(x_1, x_2, \dots, x_n) = f_i(y_1, y_2, \dots, y_n) \rightarrow \\ [x_1 = y_1 \wedge x_2 = y_2 \wedge \dots \wedge x_n = y_n]$$

for all $i < k$.

4.4 AN EXAMPLE NARRATIVE

Now let's examine the narrative of robot actions and sensor events for the example of Section 1.1. The following formulae describe the initial situation.

$$Initially_P(DoorOpen(d)) \quad (11.61)$$

$$Initially_P(BesideWall(w, s)) \leftarrow \\ w = Wall(C18, C19) \wedge s = Left \quad (11.62)$$

$$Initially_N(AtCorner(c, s)) \quad (11.63)$$

$$Initially_N(InDoorway(d, r)) \quad (11.64)$$

The robot follows the wall to its left until it arrives at corner $C'19$, where it turns right and follows the wall to its left again.

$$\text{Happens}(FollowWall, T1) \quad (11.65)$$

$$\text{Happens}(Turn(Right), T2) \quad (11.66)$$

$$\text{Happens}(FollowWall, T3) \quad (11.67)$$

$$T1 < T2 \quad (11.68)$$

$$T2 < T3 \quad (11.69)$$

Now let's suppose someone closes door $D4$ shortly after the robot sets out, and consider the incoming sensor events. The robot's front sensors go high at time $T1$, when it arrives at corner $C19$. (Recall that the *FollowWall* action is considered instantaneous.) They go low when it turns, then (unexpectedly) go high again, when it arrives at corner $C22$, having bypassed door $D4$.

$$\text{Happens}(GoesHigh(Front), T1) \quad (11.70)$$

$$\text{Happens}(GoesLow(Front), T2) \quad (11.71)$$

$$\text{Happens}(GoesHigh(Front), T3) \quad (11.72)$$

The above formulae only describe the sensor events that do occur. But in general, we want explanations of sensor data to exclude those sensor events that have not occurred. Hence we have the following definition, which captures the completion of the Happens predicate for sensor events.

Definition 1

$$\begin{aligned} COMP[\Psi] \equiv_{def} & [Happens(a, t) \wedge \\ & [a = \text{GoesHigh}(s) \vee a = \text{GoesLow}(s)] \rightarrow \\ & \forall_{(\alpha, \tau) \in \Pi} [a = \alpha \wedge t = \tau]] \end{aligned}$$

where $\Pi = \{(\alpha, \tau) | Happens(\alpha, \tau) \in \Psi\}$.

The following formula is one possible explanation of the above sensor events.

$$\text{Happens}(CloseDoor(D4), t) \wedge 0 \leq t < T3 \quad (11.73)$$

This is expressed by the following proposition.

Proposition 2

Let,

- Σ be the conjunction of (11.40) to (11.52) with (11.57) and (11.58),
- Δ_N be the conjunction of (11.61) to (11.69),
- Δ_T be the conjunction of (11.53) to (11.56),
- Δ_E be the formula (11.73),

- Φ be the conjunction of the formulae representing the robot's environment, as described in Section 4.1,
- Ω be the conjunction of (11.59) and (11.60), and
- Γ be the conjunction of (11.70) to (11.72).

We have,

$$\begin{aligned} CIRC[\Sigma; Initiates, Terminates, Releases] \wedge \\ CIRC[\Delta_N \wedge \Delta_T \wedge \Delta_E; Happens] \wedge \\ EC \wedge \Omega \wedge \Phi \models COMP[\Gamma]. \end{aligned}$$

In general, a collection of sensor data can have many explanations. Explanations can be ordered using a preference criterion, such as one which favors explanations with few events. But there can still be many minimal explanations. In these circumstances, the robot can simply proceed on the assumption that the first explanation it finds is the true explanation. It's reasonable to expect that, if the explanation is indeed false, the processing of subsequent sensor data will reveal this. But obviously this topic merits further investigation.

In the present experiment, the policy of adopting the first explanation can lead to faulty explanations if there is more than one door on the same wall. This is because there would then be no way to distinguish between two competing explanations – one involving the first door being closed, and one involving the second door being closed. This problem can be remedied by the use of distance information to select between the competing explanations.

The next section spells out how the planning and perception processes are embedded in the overall sense-plan-act cycle that controls the robot.

5 INTERLEAVING SENSING, PLANNING AND ACTING

In the implemented system, the planning and perception processes each carry out a single resolution step before suspending and going around the cycle again. The perception task can be thought of as a producer of explanations in the form of Happens formulae, which are consumed by the planning process. The planner treats them in exactly the same way as it treats new steps in the plan it's generating, which are also Happens formulae. Therefore, these incoming explanations can violate the plan's "protected links" (previously proved $\neg Clipped$ formulae). When this occurs, the system replans from scratch.

Consider the example of Section 1.1. In response to the *GoesHigh(Front)* sensor event the robot receives when it encounters corner *C22*, the perception process generates a formula of the form,

$$Happens(CloseDoor(D4), \tau)$$

as described in the previous section. When this is assimilated by the planning process, it violates a protected link of the form,

$$\neg Clipped(t1, DoorOpen(D4), t2)$$

where $t1 < t < t2$. This precipitates replanning, whereupon the planner finds the alternative route via room $R2$.

As it stands, the planner of Section 3 produces actions in regression order. That is to say, the last action to be carried out is generated first. This means that, if interrupted, the planner's partial results are useless. What we require instead is a progression planner – one that generates the earliest action of a plan first. If a progression planner is interrupted, its partially constructed plan will contain actions that can be executed immediately.

One way to generate plans in progression order is via *hierarchical planning*. This is the approach adopted here. The foregoing logical treatment of partial order planning can be straightforwardly extended to planning via hierarchical decomposition. Compound action definitions are introduced, and the abductive definition of planning can be retained as is.

As an example, here's the definition of a *GoToRoom* action in terms of the *GoThrough* action from Section 3. The term $GoToRoom(r1, r2)$ denotes the action of going from room $r1$ to room $r2$. (In the implemented system, the *GoThrough* action itself is broken down into *FollowWall* and *Turn* actions in a similar way.)

$$Happens(GoToRoom(r, r), t, t) \quad (11.74)$$

$$\begin{aligned} & Happens(GoToRoom(r1, r3), t1, t3) \leftarrow \\ & Connects(d, r1, r2) \wedge Happens(GoThrough(d), t1) \wedge \quad (11.75) \\ & \quad Happens(GoToRoom(r2, r3), t2, t3) \wedge \\ & \quad t1 < t2 \wedge \neg Clipped(t1, InRoom(r2), t2) \end{aligned}$$

$$\begin{aligned} & Initiates(GoToRoom(r1, r2), InRoom(r2), t) \leftarrow \quad (11.76) \\ & HoldsAt(InRoom(r1), t) \end{aligned}$$

In effect, when implemented via abductive logic programming, these clauses carry out a forward search, in contrast to the backward search effected by the clauses in Section ???. The clauses used in the implemented system incorporate a heuristic to give more direction to the search.

In general, if the effects of a compound action follow from the effects of its sub-actions, it adds little to the formalization, logically. But, if they are defined in the right way, the presence of compound actions will adjust the computation so that it generates actions in progression order. Specifically, the earliest component action in a compound action definition must be the closer to a low level action than its successors. In (11.75), for example, the *GoThrough* action is at a lower level than the *GoToRoom* action.

Moreover, the ability of hierarchical decomposition to quickly generate a first action in response to a situation justifies the use of a replan-from-scratch strategy rather than a more sophisticated replanning technique.

Formulae (11.74) and (11.75) illustrate both conditional decomposition and recursive decomposition: a compound action can decompose into different sequences of sub-actions depending on what conditions hold, and a compound action can be decomposed into a sequence of sub-actions that includes a compound action of the same type as itself. A consequence of this is that the event calculus with compound actions could be used to implement a universal Turing machine, and is therefore formally as powerful as any programming language. In this respect, it can be used in the same way as GOLOG (Levesque et al., 1997). Note, however, that we can freely mix direct programming with planning from first principles.

6 RELATED WORK

The title of this paper, "Reinventing Shakey", alludes to the fact that the logic-based approach to robotics was first seriously attempted in the Shakey project in the late Sixties (Nilsson, 1984). One the successes for which the Shakey project is well-known was the STRIPS approach to planning (Fikes and Nilsson, 1971). The STRIPS planner arose out of dissatisfaction with Green's earlier attempts to use resolution-based theorem proving for planning (Green, 1969). In many ways, Green's work is more akin to modern cognitive robotics than STRIPS. We'll return to STRIPS shortly. But first, let's consider how contemporary work in cognitive robotics is an advance on Green's efforts. Why is it not subject to the same pitfalls?

There are two main differences between contemporary cognitive robotics and Green's work. First, Green's approach to planning was beset by the frame problem. But we now have a number of satisfactory solutions to the frame problem that can be deployed in cognitive robotics (Shanahan, 1997c). Second, the Shakey project relied on full search-based planning from first principles, which is computationally very expensive. Like other recent work in cognitive robotics (Levesque et al., 1997), the approach presented here uses chiefly pre-compiled plans – programs, effectively. In the present approach, this also facilitates reactivity, another feature notably lacking in Shakey.

Let's return briefly to STRIPS. The STRIPS planning algorithm is now of purely historical interest, as it has long been superseded by more efficient techniques. But the STRIPS language – the language in which planning problems are described – has had lasting influence on the planning community. One aim of contemporary cognitive robotics is to return to logic as a planning language. This move can be justified in many ways: logic is a more expressive language, logic is a *lingua franca* used in other areas of AI, logic has a clear semantics with well-understood mathematical properties, and so on.

However, the logic-based approach to robotics faces a number of challenges. First, with respect to the present work, a major question is how to scale up. The Khepera robots have very poor sensors, and can only carry out wall-following, or similar basic navigational operations. When we move up to richer sensors, such as sonar or vision, how will the techniques of this paper be adapted? Likewise, the Kheperas in the experiments reported here inhabit a simple, static, uniform

environment, of just the sort that Brooks criticizes (Brooks, 1991). How will logic fare when confronted with a complex, dynamic, messy environment?

One approach is to employ an architecture that cleanly separates low- and high-level issues, both on the control front and on the sensing front. This is the approach taken by the Toronto group (Lespérance et al., 1994). Then, the question of how to deal with complex sensor data and motor control can be pushed into the lower level, where off-the-shelf techniques can be employed. But ideally, the line between low-level and high-level, between the level of perception and control and the logical level, should be drawn as low as possible, to allow logical reasoning as much influence as possible. So more work is required on topics such as how to move directly from raw sensor data to logical representations of low-level detail of the environment, such as the shapes of obstacles.

From a methodological point of view, the present work falls somewhere between the robotics work carried out at the University of Texas (Baral and Tran, 1998) and the early work carried out at the University of Toronto (Lespérance et al., 1994). In the Toronto work, there is no planning, except as a last resort. Instead, the robot directly executes a program written in GOLOG (Levesque et al., 1997), a language based on the situation calculus. A planner is invoked only if the plan monitor detects a failure, in which case planning is used to get the world into a state from which execution of the program can resume (DeGiacomo et al., 1998). Therefore, the course of actions the robot is to take is worked out almost entirely in advance, and is fixed in the program the robot executes.

More recently, the Toronto group have developed a variant of Golog called ConGolog (DeGiacomo et al., 1997). ConGolog incorporates facilities for concurrent execution, interrupt handling, and dealing with exogenous actions. Using ConGolog, the precise course of actions taken by the robot depends on conditions and events at run-time, resulting in more reactivity. However, in neither Golog nor ConGolog is there any notion of a goal with respect to which a program can be proved correct. In the Texas work (Baral and Tran, 1998), by contrast, the effects of actions are specified using logic, and the resulting theory is used to generate, off-line, a set of provably correct condition-action control rules which are then used to direct the robot. The course of actions carried out by the robot is, therefore, largely determined at run-time, and the resulting behavior is highly reactive.

The present work differs from each of these approaches. In contrast to both the Toronto and Texas approaches, the work reported here uses on-line planning, and there is a clear notion of a goal, which must be entailed by the plan. However, the heavy use of hierarchical planning means that most of the time the robot is, in effect, executing a program. On the other hand, the constant checking of protected links (negated *Clipped* literals) during execution, which can precipitate rapid replanning, means that the robot is reactive to unexpected changes, like a ConGolog program or the Texas robot. The aim is to combine the advantages of planning, direct programming, and reactive control rules in a uniform, logic-based architecture.

7 CONCLUDING REMARKS

To summarize, the aim of the ongoing work reported here is to design and build theoretically well-founded, general purpose systems for high-level robot control, in which each computational step is also a step of logical inference, and each computational state has declarative meaning. Needless to say, the ideas presented merit a good deal of further study, and although preliminary results are promising, it remains to be seen whether they will scale up to robots with richer sensors in more realistic environments.

Acknowledgements

Thanks to Jack Minker, Mark Witkowski, and two anonymous referees. This work was carried out as part of EPSRC project GR/L20023 "Cognitive Robotics".

References

- Baral, C. and Tran, S. C. (1998). Relating theories of actions and reactive control. *Linköping Electronic Articles in Computer and Information Science*, 3(9).
- Brooks, R. A. (1991). Intelligence without reason. In *Proceedings 1991 International Joint Conference on Artificial Intelligence (IJCAI 91)*, pages 569–595. Morgan Kaufmann.
- DeGiacomo, G., Lespérance, Y., and Levesque, H. (1997). Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *Proceedings 1997 International Joint Conference on Artificial Intelligence (IJCAI 97)*, pages 1221–1226. Morgan Kaufmann.
- DeGiacomo, G., Reiter, R., and Soutchanski, M. (1998). Execution monitoring of high-level robot programs. In *Proceedings 1998 Knowledge Representation Conference (KR 98)*, pages 453–464. Morgan Kaufmann.
- Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.
- Green, C. (1969). Applications of theorem proving to problem solving. In *Proceedings 1969 International Joint Conference on Artificial Intelligence (IJCAI 69)*, pages 219–240.
- Kautz, H. and Selman, B. (1996). Pushing the envelope: Propositional logic and stochastic search. In *Proceedings 1996 American Association for Artificial Intelligence Conference (AAAI 96)*, pages 1194–1201. MIT Press.
- Lespérance, Y., Levesque, H. J., Lin, F., Marcu, D., Reiter, R., and Scherl, R. B. (1994). A logical approach to high-level robot programming: A progress report. In B. Kuipers, editor, *Control of the Physical World by Intelligent Systems: Papers from the 1994 American AAAI Fall Symposium*, page 7985.
- Levesque, H., Reiter, R., Lespérance, Y., Lin, F., and Scherl, R. B. (1997). Golog: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 31:59–83.
- Lifschitz, V. (1994). Circumscription. In Gabbay, D. M., Hogger, C. J., and Robinson, J. A., editors, *The Handbook of Logic in Artificial Intelligence*

- and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*, pages 297–352. Oxford University Press.
- Nilsson, N. (1984). Shakey the robot. Technical Report 323, SRI International, Menlo Park, California.
- Shanahan, M. P. (1997a). Event calculus planning revisited. In *Proceedings 4th European Conference on Planning (ECP 97)*, number 1348 in Springer-Verlag Lecture Notes in Artificial Intelligence, pages 390–402. Springer-Verlag.
- Shanahan, M. P. (1997b). Noise, non-determinism and spatial uncertainty. In *Proceedings 1996 American Association for Artificial Intelligence Conference (AAAI 97)*, pages 153–158. MIT Press.
- Shanahan, M. P. (1997c). *Solving the Frame Problem: A Mathematical Investigation of the Common Sense Law of Inertia*. MIT Press.
- Shanahan, M. P. (1999). The event calculus explained. In Wooldridge, M. J. and Veloso, M., editors, *Artificial Intelligence Today*, number 1600 in Springer-Verlag Lecture Notes in Artificial Intelligence, pages 409–430. Springer-Verlag.
- Shanahan, M. P. (2000). An abductive event calculus planner. *The Journal of Logic Programming*. To appear.
- Shanahan, M. P. and Witkowski, M. (2000). Event calculus planning through satisfiability. Unpublished.

VIII

LOGIC FOR AGENTS AND ACTIONS

Chapter 12

REASONING AGENTS IN DYNAMIC DOMAINS

Chitta Baral,

Department of Computer Sc. and Engg.

Arizona State University, Tempe, AZ 85287

chitta@asu.edu

Michael Gelfond

Department of Computer Sc.

Texas Tech University, Lubbock, TX 79409

michael.gelfond@coe.ttu.edu

Abstract The paper discusses an architecture for intelligent agents based on the use of *A-Prolog* - a language of logic programs under the answer set semantics. *A-Prolog* is used to represent the agent's knowledge about the domain and to formulate the agent's reasoning tasks. We outline how these tasks can be reduced to answering questions about properties of simple logic programs and demonstrate the methodology of constructing these programs.

Keywords: Intelligent agents, logic programming and nonmonotonic reasoning.

1 INTRODUCTION

This paper is a report on the attempt by the authors to better understand the design of software components of intelligent agents capable of reasoning, planning and acting in a changing environment. The class of such agents includes, but is not limited to, intelligent mobile robots, softbots, immobots, intelligent information systems, expert systems, and decision-making systems. The ability to design intelligent agents (IA) is crucial for such diverse tasks as space exploration, intelligent communication with the Internet, and development of various types of control systems. Despite the substantial progress in the work on IA achieved in the last decade we are still far from a clear understanding of the basic principles and techniques needed for their design.

The problem is complicated by the fact that IA differ from more traditional software systems in several important aspects:

- An agent could have a large amount of knowledge about the domain in which it is intended to act, and about its own capabilities and goals.
- It should be able to frequently expand this knowledge by new information coming from observations, communication with other agents, and awareness of its own actions.
- All this knowledge cannot be explicitly represented in the agent's memory. This implies that the agent should be able to reason, i.e. to extract knowledge stored there implicitly.
- Finally, the agent should be able to use its knowledge and its ability to reason to rationally plan and execute its actions.

These observations imply that solid theoretical foundations of agent design should be based on theories of knowledge representation and reasoning. Work reported in this paper is based on two such theories: theory of logic programming and nonmonotonic reasoning (Baral and Gelfond, 1994; Lifschitz, 1996; Marek and Truszczyński, 1993) and theory of actions and change (Sandewall, 1998).

The latter develops the ontology and basic relations needed for modeling the agent's domain. The former provides a logic for representing and reasoning with the domain knowledge. This logic is more expressive than classical first-order predicate calculus (Dantsin et al., 1997). This additional expressibility is needed to represent defaults, causal relations, various forms of transitive closures, etc. The entailment relation of the logic is nonmonotonic, i.e. it allows the reasoner to withdraw previously made conclusions when new information becomes available. This property substantially simplifies the process of assimilating new information. There are efficient systems implementing rather general reasoning algorithms developed in the logic programming community. In this paper we show how these systems can be used to implement specific planning, explanation finding, and plan checking algorithms in a simple observe-think-act style agent architecture.

There are several other approaches which use logic as a basis for agent design. They differ primarily by the languages and the logics in which the agent's knowledge is specified, by the algorithms used by the agent to perform its reasoning tasks, and by the agent's architecture. For instance, the Toronto School of Cognitive Robotics (Levesque et al., 1997) uses modified and substantially expanded versions of the situation calculus of (McCarthy and Hayes, 1969) to specify the agent knowledge. Despite occasional use of nonmonotonic entailment (primarily in the form of circumscription (McCarthy, 1980)) the Toronto School seems to prefer to stay as close as possible to the entailment relation and reasoning algorithms of classical logic. In their approach the agent's behavior is determined by a program written in a programming language Golog (Levesque et al., 1997) or one of its variants. Such a program allows procedural constructs such as sequences,

loops, conditionals, and has non-deterministic operators and test conditions where the non-determinism is resolved by the interpreter so that the overall plan is executable. To interpret and execute these constructs the agent uses the situation calculus based entailment relation described above. The main program can be complemented by an execution monitor capable of modifying plans invalidated by exogenous actions (DeGiacomo et al., 1998), as well as by other special purpose reasoning modules. In (Kowalski, 1995; Kowalski and Sadri, 1999), the authors investigate an architecture based on a variant of an observe-think-act cycle. The internal state of the agent is determined by a collection of standard logic programming rules, integrity constraints, condition-actions rules, etc. The approach is somewhat independent of a particular entailment relation but the authors seem to favor entailments associated with the Clark's completion (Clark, 1978). The thinking part of a cycle is performed by a combination of traditional logic programming algorithms, abduction (Denecker and De Schreye, 1998; Kakas et al., 1998) and forward reasoning by means of integrity constraints.

Even though all these approaches develop in parallel they share many common insights and ideas. Sometimes this is a result of direct influence and sometimes (and probably more often) the similarities are determined by the common subject of study. The technical differences between the approaches are however rather large and we believe that the complete understanding of their pros and cons will require a substantial amount of further work. This paper is aimed at explaining our approach and we do not make any attempt at the comparison.

2 MODELING THE AGENT

In this paper we adopt the following simplifying assumptions about the agents and their environments:

1. The dynamics of the agent's environment is viewed as a (possibly infinite) transition diagram whose states are sets of fluents (i.e., statements whose truth depends on time) and whose arcs are labeled by actions.
2. The agent is capable of making correct observations, performing actions, and remembering the domain history.

These assumptions hold in many realistic domains and are suitable for a broad class of applications. In many domains, however, the effects of actions and the truth of observations can only be known with a substantial degree of uncertainty which cannot be ignored in the modeling process. In this case our basic approach can be expanded by introducing probabilistic information. We prefer to start our investigation with the simplest case and work our way up the ladder of complexity only after this simple case is reasonably well understood. The assumptions above determine the structure of the agent's knowledge base. It consists of two parts. The *first part*, called an *action description*, specifies the transition diagram of the agent. It contains descriptions of actions and fluents relevant to the domain together with the definition of possible successor states to which the system can move after an action a is executed in a state σ .

Due to the size of the diagram the problem of finding its concise specification is far from being trivial. Its solution requires a good understanding of the nature of causal effects of actions in the presence of complex interrelations between fluents. An additional level of complexity is added by the fact that, unlike standard mathematical reasoning, causal reasoning is nonmonotonic, i.e., new information about the domain can cause a reasoning agent to withdraw its previously made conclusions. Nonmonotonicity is partly caused by the need for representing defaults, i.e., statements of the form "Normally, objects of type A have property P ". A default that is often used in causal reasoning is the so called inertia axiom (McCarthy and Hayes, 1969). It says that "Normally, actions do not change the values of fluents". The problem of finding a concise and accurate representation of this default, known as the Frame Problem, substantially influenced AI research during the last twenty years (Shanahan, 1997). The second part of the agent's knowledge contains observations made by the agent together with a record of its own actions. It defines a collection of paths in the diagram which can be interpreted as the domain's possible trajectories traversed so far. If the agent's knowledge is complete (i.e., it has complete information about the initial state and the action occurrences) and its actions are deterministic then there is only one such path. We believe that a good theory of agents should contain a logical language (equipped with a consequence relation) which would be capable of representing both types of the agent's knowledge.

A-Prolog

In this paper the language of choice is *A-Prolog* - a language of logic programs under the answer set semantics (Gelfond and Lifschitz, 1988; Gelfond and Lifschitz, 1991). An *A-Prolog* program consists of a signature Σ and a collection of rules of the form

$$\text{head} \leftarrow \text{body} \quad (12.1)$$

where the *head* is empty or consists of a literal l_0 and *body* is of the form $l_1, \dots, l_m, \text{not } l_{m+1}, \dots, \text{not } l_n$ where l_i 's are literals over Σ . A literal is an atom p or its negation $\neg p$. If the *body* is empty we replace \leftarrow by a period. While $\neg p$ says that p is false, $\text{not } p$ has an epistemic character and can be read as "there is no reason to believe that p is true". The symbol *not* denotes a non-standard logical connective often called *default negation* or *negation as failure*. An *A-Prolog* program Π can be viewed as a specification given to a rational agent for constructing beliefs about possible states of the world. Technically these beliefs are captured by the notion of *answer set* of a program Π . By *ground*(Π) we denote a program obtained from Π by replacing variables by the ground terms of Σ . By answer sets of Π we mean answer sets of *ground*(Π). If Π consists of rules not containing default negation then its answer set S is the smallest set of ground literals of Σ which satisfies two conditions:

1. S is closed under the rules of *ground*(Π), i.e. for every rule (12.1) in Π , either there is a literal l in its body such that $l \notin S$ or its non-empty head $l_0 \in S$.

2. if S contains an atom p and its negation $\neg p$ then S contains all ground literals of the language.

It is not difficult to show that there is at most one set $Cn(\Pi)$ satisfying these conditions.

Now let Π be an arbitrary ground program of *A-Prolog*. For any set S of ground literals of its signature Σ , let Π^S be the program obtained from Π by deleting

- (i) each rule that has an occurrence of *not l* in its body with $l \in S$,
- (ii) all occurrences of *not l* in the bodies of the remaining rules.

Then S is an answer set of Π if

$$S = Cn(\Pi^S). \quad (12.2)$$

In this paper we limit our attention to *consistent* programs, i.e. programs with at least one consistent answer set. Let S be an answer set of Π . A ground literal l is *true* in S if $l \in S$; *false* in S if $\neg l \in S$. This is expanded to conjunctions and disjunctions of literals in a standard way. A query Q is *entailed* by a program Π ($\Pi \models Q$) if Q is true in all answer sets of Π . Queries $l_1 \wedge \dots \wedge l_n$ and $\bar{l}_1 \vee \dots \vee \bar{l}_n$ (where for an atom p , \bar{p} denotes $\neg p$, and $\overline{\neg p}$ denotes p) are called complementary. If Q and \overline{Q} are complementary queries then Π 's answer to Q is *yes* if $\Pi \models Q$; *no* if $\Pi \models \overline{Q}$, and *unknown* otherwise.

Here are some examples. Assume that signature Σ contains two object constants a and b . The program

$$\Pi_1 = \{\neg p(X) \leftarrow \text{not } q(X). \quad q(a).\}$$

has the unique answer set $S = \{q(a), \neg p(b)\}$. The program

$$\Pi_2 = \{p(a) \leftarrow \text{not } p(b). \quad p(b) \leftarrow \text{not } p(a).\}$$

has two answer sets, $\{p(a)\}$ and $\{p(b)\}$. The programs

$$\Pi_3 = \{p(a) \leftarrow \text{not } p(a).\} \text{ and } \Pi_4 = \{p(a). \quad \leftarrow p(a).\}$$

have no answer sets.

It is easy to see that programs of *A-Prolog* are *nonmonotonic*. ($\Pi_1 \models \neg p(b)$ but $\Pi_1 \cup \{q(b).\} \not\models \neg p(b).$) *A-Prolog* is closely connected with more general nonmonotonic theories. In particular, as was shown in (Marek and Truszczyński, 1989; Gelfond and Lifschitz, 1991), there is a simple and natural mapping of programs of *A-Prolog* into a subclass of Reiter's default theories (Reiter, 1980). (Similar results are also available for Autoepistemic Logic of Moore (Moore, 1985).)

Specifying transition diagrams

To describe the agent's transition diagram, the domain's history and the type of queries available to the agent we use action languages (Gelfond and Lifschitz, 1992) which can be viewed as formal models of parts of natural language that are used for talking about the effects of actions. A particular action language reflects properties of a domain and the abilities of an agent. In this paper we define a class of action languages \mathcal{AL} which combine ideas from (Baral, 1995; McCain and Turner, 95; Baral et al., 1995; Baral et al., 1997; Guinchiglia and Lifschitz, 1998). Languages $\mathcal{AL}(\Sigma)$ from this class will be parameterized with respect to a signature Σ which we normally assume fixed and omit from our notation. The simplicity of \mathcal{AL} as a language makes it suitable for illustrating our methodology, which is a primary goal of this paper. The methodology however is applicable to languages with much richer ontology.

We follow a slightly modified view of (Lifschitz, 1997) and divide an action language into three parts: *action description language*, *history description language*, and *query language*. We start with defining the action description part of \mathcal{AL} , denoted by \mathcal{AL}_d . Its signature Σ will consist of two disjoint, non-empty sets of symbols: the set \mathbf{F} of fluents and the set \mathbf{A} of *elementary actions*. A set $\{a_1, \dots, a_n\}$ of elementary actions is called a *compound action*. It is interpreted as a collection of elementary actions performed simultaneously. By *actions* we mean both, elementary and compound actions. By *fluent literals* we mean fluents and their negations. By \bar{l} we denote the fluent literal complementary to l . A set S of fluent literals is called *complete* if for any $f \in \mathbf{F}$, $f \in S$ or $\neg f \in S$. An action description of $\mathcal{AL}_d(\Sigma)$ is a collection of propositions of the form

1. $\text{causes}(a_e, l_0, [l_1, \dots, l_n]),$
2. $\text{caused}(l_0, [l_1, \dots, l_n]),$ and
3. $\text{impossible_if}(a, [l_1, \dots, l_n])$

where a_e and a are elementary and arbitrary actions respectively and l_0, \dots, l_n are fluent literals from Σ . The first proposition says that, if the action a were to be executed in a situation in which l_1, \dots, l_n hold, the fluent literal l_0 will be caused to hold in the resulting situation. Such propositions are called *dynamic causal laws*. (The restriction on a_e being elementary is not essential and can be lifted. We require it to simplify the presentation). The second proposition, called a *static causal law*, says that, in an arbitrary situation, the truth of fluent literals, l_1, \dots, l_n , (often called the body of (2)) is sufficient to cause the truth of l_0 . The last proposition says that action a cannot be performed in any situation in which l_1, \dots, l_n hold. Notice that here a can be compound, e.g. $\text{impossible}(\{a_1, a_2\}, []))$ means that elementary actions a_1 and a_2 cannot be performed concurrently. An action description \mathcal{A} of \mathcal{AL}_d defines a transition diagram describing effects of actions on the possible states of the domain. A *state* is a complete and consistent set σ of fluent literals such that σ is closed under the static causal laws of \mathcal{A} , i.e. for any static causal law (2) of \mathcal{A} , if $\{l_1, \dots, l_n\} \subseteq \sigma$ then $l_0 \in \sigma$. States serve as the nodes of the transition

1.	$holds_at(L, T')$	$\leftarrow next(T, T'),$ $causes(A_e, L, P),$ $occurs_at(A_e, T),$ $hold_at(P, T).$	} $\Pi(N)$
2.	$holds_at(L, T)$	$\leftarrow caused(L, P),$ $hold_at(P, T).$	
3.	$holds_at(L, T')$	$\leftarrow next(T, T'),$ $holds_at(L, T),$ $not holds_at(\overline{L}, T').$	
4.	$hold_at([], T).$		
5.	$hold_at([L Rest], T)$	$\leftarrow holds_at(L, T),$ $hold_at(Rest, T).$	
6.	$\neg occurs_at(A_e, T)$	$\leftarrow not occurs_at(A_e, T).$	
7.	$\neg occurs_at(A_c, T)$	$\leftarrow member(A_e, A_c),$ $\neg occurs_at(A_e, T).$	
8.	$\neg occurs_at(A_c, T)$	$\leftarrow member(A_e, \overline{A_c}),$ $occurs_at(A_e, T).$	
9.	$occurs_at(A_c, T)$	$\leftarrow not \neg occurs_at(A_c, T).$	
10.		$\leftarrow impossible_if(A, P),$ $hold_at(P, T),$ $occurs_at(A', T),$ $subset(A, A').$	

Figure 12.1

diagram. Nodes σ_1 and σ_2 are connected by a directed arc labeled by an action a if σ_2 may result from executing a in σ_1 . (To simplify the diagram we do not distinguish between links labeled by an elementary action a_e and ones labeled by $\{a_e\}$.) The set of all states that may result from executing a in a state σ will be denoted by $res(a, \sigma)$. Defining this set for action descriptions of increasingly complex action languages seems to be one of the main issues in the development of action theories. Several sixpoint definitions of this set were recently given by different authors; see for instance (McCain and Turner, 95; Lin, 95; Guinchiglia and Lifschitz, 1998). The definition we give in this paper is based on the semantics of *A-Prolog* and is similar in spirit to the work in (Baral, 1995). First let us consider the program in Figure 12.1. We use A_e , A_c , and A as variables for elementary, compound, and arbitrary actions respectively; L and P are variables for fluent literals and (finite) sets of fluent literals, and T and T' are variables for integers from some interval $[0, N]$. Integers from this interval

will be later interpreted as time points. The program uses the list notation [] and predicate symbols *member* and *subset* denoting the standard membership and proper subset relations; $\text{member}(A_e, \overline{A_c})$ holds when A_e is an elementary action not belonging to A_c . Rules one and two of $\Pi(N)$ describe the effects of causal laws. The predicate symbol *holds_at*(L, T) denotes the relation "A fluent literal L is true at moment T "; *hold_at*(P, T) denotes the expansion of this relation to sets of fluents; *occurs_at*(A, T) indicates that an action A occurs at moment T . The relation *next*(T, T') is satisfied by two consecutive moments of time from the interval $[0, N]$. Rule three encodes the law of inertia. The default nature of this law is nicely captured by the use of default negation of *A-Prolog*. Rules four and five define *hold_at*(P, T). Normally, the program $\Pi(N)$ will be used in conjunction with a complete list of elementary actions which occurred in the domain. The completeness is expressed by the closed world assumption for elementary actions encoded by rule six. Rules seven and eight define the complete list of compound actions which do not occur at moment T . Rule nine uses the closed world assumption to define a compound action which does occur at that moment. (It is easy to see that if a_1, \dots, a_k is the complete list of elementary actions which occur at moment T then the only compound action which occurs at this moment is $\{a_1, \dots, a_k\}$.) The last rule with the empty head is used to insure that impossible actions are indeed impossible.

By $\Pi(1)$ we denote the program obtained from $\Pi(N)$ by replacing the time variables by 0 and 1. Now we define the transition relation determined by an action description \mathcal{A} .

Definition 1 For any action a and state σ_1 , a state σ_2 is a *successor state* of a on σ_1 if there is an answer set S of

$$\Pi(1) \cup \mathcal{A} \cup \{\text{holds_at}(l, 0) : l \in \sigma_1\} \cup \{\text{occurs_at}(a_i, 0) : a_i \in a\}$$

such that $\sigma_2 = \{l : \text{holds_at}(l, 1) \in S\}$.

It is essential to notice that Definition 1 allows us to *reduce computing successor states of the transition diagram representing our dynamic domain to computing answer sets of comparatively simple logic programs*. For domains without concurrent actions this definition is equivalent to one from (McCain and Turner, 95). The idea however is not limited to \mathcal{AL}_d and can be used to define a larger range of transition functions.

We say that an action description \mathcal{A} of \mathcal{AL}_d is *deterministic* if for any state σ and any action a from Σ there is at most one successor state, i.e., the cardinality of the set $\text{res}(a, \sigma)$ is at most one. The following example shows that action descriptions of \mathcal{AL}_d can be nondeterministic.

Example 1 Let \mathcal{A}_1 be an action description

$\text{causes}(a, f, [\]) . \quad \text{caused}(\neg g_1, [f, g_2]) . \quad \text{caused}(\neg g_2, [f, g_1]) .$

Using the Definition 1 it is easy to show that the transition diagram of \mathcal{A}_1 can be given as follows:

$$\begin{aligned}
 res(a, \{f, g_1, \neg g_2\}) &= \{\{f, g_1, \neg g_2\}\} \\
 res(a, \{f, \neg g_1, g_2\}) &= \{\{f, \neg g_1, g_2\}\} \\
 res(a, \{f, \neg g_1, \neg g_2\}) &= \{\{f, \neg g_1, \neg g_2\}\} \\
 res(a, \{\neg f, g_1, g_2\}) &= \{\{f, g_1, \neg g_2\}, \{f, \neg g_1, g_2\}\} \\
 res(a, \{\neg f, g_1, \neg g_2\}) &= \{\{f, g_1, \neg g_2\}\} \\
 res(a, \{\neg f, \neg g_1, g_2\}) &= \{\{f, \neg g_1, g_2\}\} \\
 res(a, \{\neg f, \neg g_1, \neg g_2\}) &= \{\{f, \neg g_1, \neg g_2\}\}
 \end{aligned}$$

Action descriptions not containing static causal laws are deterministic and hence the addition of such laws adds to expressive power of the language. The following proposition gives a sufficient condition guaranteeing that this property holds.

Proposition 1 Any action description \mathcal{A} of \mathcal{AL}_d with static causal laws containing at most one literal in the body is deterministic.

The proposition is meant to illustrate how the syntactic form of action description can be used to learn important properties of the corresponding transition relations. Similar results in somewhat different context can be found in (Pinto, 1999; Lin, 2000).

Specifying the history

We now describe a language \mathcal{AL}_h for specifying the history of the agent's domain. To do that we expand action signature Σ by integers $0, 1, 2, \dots$, which are used to denote time points in the actual evolution of the system. If such evolution is caused by a sequence of consecutive actions a_1, \dots, a_n then 0 corresponds to the initial situation and k ($0 < k \leq n$) corresponds to the end of the execution of a_1, \dots, a_k .

The domain's past is described by a set, Γ , of axioms of the form

1. $happened(a, k)$.
2. $observed(l, k)$.

where a 's are elementary actions. The first axiom says that (elementary) action a has been executed at time k ; the second axiom indicates that fluent literal l was observed to be true at time k . The axioms of Γ will be often referred to as *observations*. Every set Γ of observations uniquely defines the *current moment of time*, $t_c(\Gamma)$. If $\Gamma = \emptyset$ then $t_c(\Gamma) = 0$; If every occurrence, t' , of time in Γ is less than or equal to some t such that $happened(a, t) \in \Gamma$ then $t_c(\Gamma) = t + 1$; otherwise, $t_c(\Gamma) = \max\{t : observed(l, t) \in \Gamma\}$. (Γ will be omitted whenever possible.)

A set of axioms defines the collection of paths in a transition diagram T which can be interpreted as possible histories of the domain represented by T . (As usual, by a path we mean a sequence $(\sigma_0, a_1, \sigma_1, \dots, a_n, \sigma_n)$, where σ_0 is a state and for $1 \leq i \leq n$, $\sigma_i \in res(a_i, \sigma_{i-1})$.) If our knowledge about the initial situation is complete and actions are deterministic then there is only one such path. A pair $\langle \mathcal{A}, \Gamma \rangle$, where \mathcal{A} is an action description and Γ is a set of

observations, is called a *domain description*. The following definition refines the intuition behind the meaning of observations.

Definition 2 Let $\mathcal{D} = \langle \mathcal{A}, \Gamma \rangle$ be a domain description with the current time n , T be the transition diagram defined by \mathcal{A} , and $H = \langle \sigma_0, a_1, \sigma_1, \dots, a_n, \sigma_n \rangle$ be a path in T . We say that H is a *possible history* of \mathcal{D} if

1. action a_i is the i -th action in H iff $a_i = \{a : \text{happened}(a, i - 1) \in \Gamma\}$.
2. If $\text{observed}(l, k) \in \Gamma$ then $l \in \sigma_k$.

A domain description \mathcal{D} is called *consistent* if it has a non-empty set of possible histories.

Queries and the consequence relation

Let us now assume that at each moment of time an agent maintains a knowledge base in the form of a domain description \mathcal{D} . Various reasoning tasks of an agent can be reduced to answering queries about properties of the agent's domain. The corresponding query language, \mathcal{L}_q , includes the following queries:

1. $\text{holds_at}(l, t)$.
2. $\text{currently}(l)$.
3. $\text{holds_after}(l, [a_n, \dots, a_1], t)$.

Query (1) asks if fluent literal l holds at time t . Query (2) asks if l holds at the current moment of time. The last query is hypothetical and is read as: "Is it true that a sequence a_1, \dots, a_n of actions could have been executed at the time $0 \leq t \leq t_c$, and if it were, then fluent literal l would be true immediately afterwards". If $t < t_c$ and the sequence a_1, \dots, a_n is different from the one that actually occurs at t then the corresponding query expresses a counterfactual. If $t = t_c$ then the query expresses a hypothesis about the system's future behavior. (In this case we often omit t from the query and simply write $\text{holds_after}(l, [a_n, \dots, a_1])$.) The definition below formalizes this intuition. Let $\mathcal{D} = \langle \mathcal{A}, \Gamma \rangle$ be a domain description with the current moment n , and T be the transition diagram defined by \mathcal{A} .

1. a query $\text{holds_at}(l, t)$ is a *consequence* of \mathcal{D} if for every possible history $\sigma_0, a_1, \sigma_1, \dots, a_n, \sigma_n$ of \mathcal{D} , $0 \leq t \leq n$, and we have $l \in \sigma_t$;
2. a query $\text{currently}(l)$ is a *consequence* of \mathcal{D} if for every possible history $\sigma_0, a_1, \sigma_1, \dots, a_n, \sigma_n$ of \mathcal{D} , we have $l \in \sigma_n$;
3. a query $\text{holds_after}(l, [a'_m, \dots, a'_1], t)$ is a *consequence* of \mathcal{D} if for every possible history $\sigma_0, a_1, \sigma_1, \dots, a_n, \sigma_n$ of \mathcal{D} , $0 \leq t \leq n$ and a'_1, \dots, a'_m is executable in σ_t and for any path $\sigma'_0, a'_1, \sigma'_1, \dots, a'_m, \sigma'_m$ of T such that $\sigma'_0 = \sigma_t$, $l \in \sigma'_m$.

The consequence relation between query Q and domain description $\mathcal{D} = \langle \mathcal{A}, \Gamma \rangle$ will be denoted by $\Gamma \models_{\mathcal{A}} Q$.

Architecture for intelligent agents

We now demonstrate how the notion of consistency of a domain description and the consequence relation $\Gamma \models_A Q$ can be used to describe an architecture of an intelligent agent. In this architecture the set of elementary actions is divided into two parts: the actions which can be performed by an agent and exogenous actions performed by nature or other agents in the domain. We assume that at each moment t of time the agent's memory contains domain description $\mathcal{D} = \langle \mathcal{A}, \Gamma \rangle$ and a partially ordered set \mathcal{G} of agent's *goals*. By a goal we mean a finite set of fluent literals the agent wants to make true. Partial ordering corresponds to comparative importance of goals. The agent operates under the assumption that it was able to observe all the exogenous actions. This assumption can however be contradicted by observations which may force the agent to conclude that some exogenous actions in the past remained unobserved. An agent will repeatedly execute the following steps

1. observe the world and incorporate the observations in Γ ;
2. select one of the most important goal $g \in \mathcal{G}$ to be achieved;
3. find plan a_1, \dots, a_n to achieve g ;
4. execute a_1 ;

During the first step the agent does two things. First it observes exogenous actions $\{b_1, \dots, b_k\}$ which happen at the current moment of time, t_0 . These observations are recorded by simply adding the statements $happened(b_i, t_0)$ ($1 \leq i \leq k$) to the current set of observations, Γ_0 . Now the agent's history is encoded in the new set of axioms, Γ' with the current moment of time, t' . Second, the agent observes the truth of some fluent literals, O . If the agent observed all the exogenous actions which happen at time t_0 then the domain description $\langle \mathcal{A}, \Gamma' \cup O \rangle$ where $O = \{observed(l_i, t') : l_i \in O \text{ and } l_i \text{ is observed to be true at } t'\}$ will be consistent. Otherwise it may be inconsistent. In the latter case the new observations should be explained by assuming some occurrences of unobserved exogenous actions. This suggests that instead of adding observations to the domain description¹ the agent should first check their consistency and, if necessary, provide a suitable explanation. In precise terms, by *explanation* of unexplained observations O we mean a collection of statements

$H_0 = \{happened(a_i, t_k) : t_k < t_c, a_i \text{ is an elementary exogenous action}\}$ such that

$$\Gamma \cup H_0 \cup O \text{ is consistent} \quad (12.3)$$

If O can be explained in several possible ways the agent should be supplied with some mechanism of selecting a most plausible one. (To simplify the

¹To the best of our knowledge the idea that "observations should be added together with possible explanations" was first published in (Rciter, 1995).

discussion we assume that this is possible). Let us denote the set of axioms of the agent after the first step of the loop by Γ_1 and its current time by t_1 . During the second stage the agent selects a goal $g \in G$ which will, at least for a while, determine its future actions. The goals in G may have priorities which depend on the current state of the domain and hence the agent may change its immediate goal during the next execution of the loop. Meanwhile however it goes to step three and looks for a plan to achieve g . This planning problem can be reduced to finding a sequence α of actions such that

$$\Gamma \models_{\mathcal{A}} \text{holds_after}(g, \alpha) \quad (12.4)$$

After (12.4) is solved and a plan $\alpha = a_1, \dots, a_n$ is found the agent proceeds to execute the first action, a_1 , records $\text{happened}(a_1, t_1)$ in the domain history, and goes back to step one of the loop. Of course this architecture is only valid if computation performed during the execution of the body of the loop is fast enough to not allow the possibility of occurrence of exogenous actions, that may change the relevant characteristics of the domain, before a_1 is performed. In many situations the process can be made considerably faster by pre-computing solutions of (12.3) and (12.4) for certain goals. A solution for (12.4) for instance can be stored in the agent's memory as rules of the form, say,

```
if  $\Gamma \models_{\mathcal{A}} \text{currently}(l)$  then execute( $a_1$ )
else execute( $a_2$ )
```

Agents whose ability to plan is limited to the use of such rules are called *reactive*. Otherwise, we characterize them as *deliberative*. Normally, agents should have both² deliberative and reactive components. Notice, however, that even reactive actions depend on the current knowledge of an agent and the choice of such actions may require a certain amount of reasoning. (Of course, checking if l holds in the current situation requires substantially less effort than planning or explaining observations.)

Example 2 Consider a domain with two locations, *home* and *airport*, and two objects, *money* and a *ticket*. An agent, Jack, is capable of driving from his current location to the other one and of getting money and a ticket. The last action however is possible only if Jack has money and is located at the airport. Jack views the world in terms of two fluents, *jack_at(L)* and *has_jack(O)*. The only exogenous action relevant to Jack is that of losing an object. Let us construct an action description \mathcal{A} of Jack's world and use it to model Jack's behavior. Even though all the steps in the example can be easily proven the discussion will be informal. Later we will comment on the ways to automate all the reasoning steps.

²More details on the combination of reactive and deliberative reasoning in the context of action based languages can be found in (Baral and Son, 1998). It also contains more detailed comparison to the reactive aspects of ConGolog (De Giacomo et al., 1997).

Types :

<i>location(home).</i>	<i>object(money).</i>
<i>location(airport).</i>	<i>object(ticket).</i>

Fluents :

<i>fluent(jack_at(L))</i>	\leftarrow	<i>location(L).</i>
<i>fluent(has_jack(O))</i>	\leftarrow	<i>object(O).</i>

Actions :

<i>agent_action(drive_to(L))</i>	\leftarrow	<i>location(L).</i>
<i>agent_action(get(O))</i>	\leftarrow	<i>object(O).</i>
<i>exogenous_action(lose(O))</i>	\leftarrow	<i>object(O).</i>

Causal Laws :

<i>impossible_if(drive_to(L), [jack_at(L)]).</i>	}
<i>causes(drive_to(L), jack_at(L), []).</i>	
<i>impossible_if(get(ticket), [¬has_jack(money)]).</i>	
<i>impossible_if(get(ticket), [¬jack_at(airport)]).</i>	
<i>causes(get(O), has_jack(O), []).</i>	
<i>causes(lose(O), ¬has_jack(O), []).</i>	
<i>caused(¬jack_at(L1), [jack_at(L2), L1 ≠ L2]).</i>	
<i>impossible_if([get(O), drive_to(L)], []).</i>	
<i>impossible_if([get(O1), lose(O2)], []).</i>	

A

Let us now assume that Jack's goal is to have a ticket. He starts with step one of the algorithm and records the following observations:

Axioms :

<i>observed(has_jack(money), 0)</i>	}
<i>observed(¬has_jack(ticket), 0)</i>	
<i>observed(jack_at(home), 0).</i>	

Γ₀

Since Jack has only one goal the selection is trivial. He goes to the planning stage, solves equation

$$\Gamma_0 \models_A \text{holds_after}(\text{has_jack(ticket)}, X)$$

and finds a plan, $\alpha_1 = [\text{drive_to(airport)}, \text{get(ticket)}]$. (We assume that he has enough time to find the shortest plan). Jack proceeds by executing the first action of this plan and recording this execution in the history of domain. The new collection of axioms is as follows:

$$\Gamma_1 = \Gamma_0 \cup \{\text{happened}(\text{drive_to(airport)}, 0)\}.$$

Now the current time is 1 and Jack continues his observations. Suppose he observes that his money is missing. This observation contradicts (\mathcal{A}, Γ_1) and hence needs an explanation. The only explanation, obtained by solving equation

$\Gamma_1 \cup Y \cup \{\text{observed}(\neg \text{has_jack}(\text{money}), 1)\}$ is consistent,

is $Y = \{\text{happened}(\text{lose}(\text{money}), 0)\}$, and hence

$$\Gamma_2 = \Gamma_1 \cup \{\text{happened}(\text{lose}(\text{money}), 0), \text{observed}(\neg \text{has_jack}(\text{money}), 1)\}.$$

Assuming that Jack's goal is not changed, he proceeds to solve the equation $\Gamma_2 \models_{\mathcal{A}} \text{holds_after}(\text{has_jack}(\text{ticket}), X)$, finds a new plan, $\alpha_2 = [\text{get}(\text{money}), \text{get}(\text{ticket})]$, gets money, goes back to step one and hopefully proceeds toward his goal without further interruptions.

3 REASONING ALGORITHMS

The logic programming community has developed a large number of reasoning algorithms which range from the SLDNF resolution implemented in traditional Prolog systems to the XSB resolution (Chen et al., 1995) implementing the well-founded semantics (Van Gelder et al., 1991), and comparatively recent techniques which can be used for computing answer sets of *A-Prolog* (Cholewiński et al., 1996; Niemela and Simons, 1997; Faber et al., 1999; Wang and Zaniolo, 2000). The latter form the basis for answer set programming advocated in (Niemela, 1999; Marek and Truszczyński, 1999). In this section we illustrate how these algorithms can be used to implement the above architecture and to perform the agent's reasoning tasks.

Planning

In this subsection we discuss model-theoretic planning using *A-Prolog*. In this approach, the answer sets of the program encode possible plans. We slightly differ from the previous work on answer set planning by emphasizing the use of "planning modules" to restrict the kind of plans that we are looking for. These modules can allow concurrency or force sequentiality of plans, specify preference between actions, incorporate reactive components into deliberative planning, etc. They can also contain a domain dependent control information.

We start with assuming that the corresponding domain description $\mathcal{D} = \langle \mathcal{A}, \Gamma \rangle$ is *consistent and deterministic*, the set Γ of axioms is *complete*, i.e. uniquely determines the initial situation, and the goal g is a *set of fluent literals*. We also assume that time is represented by integers from $[0, N]$, that m is the maximum length of a plan the agent is willing to search for, and that $t_c + m \leq N$.

To find a solution of equation (12.4) we consider a program

$$\Pi_d = \Pi(N) \cup \mathcal{A} \cup \Gamma \cup R$$

where R consists of rules

$$\text{occurs_at}(A, T) \leftarrow \text{happened}(A, T).$$

holds_at(L, T) ← observed(L, T).

agent_action(a). (for any agent action a.)

Next we discuss how to construct *planning modules* of agents with different degrees of sophistication. We start with a simple planning module, $PM_0(m)$:

$$\left. \begin{array}{ll} (p1) & found \quad \leftarrow t_c \leq T < t_c + m, \\ & \quad hold_at(g, T). \\ (p2) & \quad \leftarrow not\ found. \\ (p3) & occurs_at(A, T) \leftarrow t_c \leq T < t_c + m, \\ & \quad not\ hold_at(g, T), \\ & \quad agent_action(A), \\ & \quad not\ \neg occurs_at(A, T). \end{array} \right\} PM_0(m)$$

Let

$$P_0(m) = \Pi_d \cup PM_0(m)$$

The use of $P_0(m)$ for planning is based on the following observation:

Let $0 \leq k < m$. A sequence $\alpha = a_0, \dots, a_k$ of actions is a plan for g iff there is an answer set S of $P_0(m)$ such that for any a_i from α , $a_i = \{a : a \text{ is elementary and } occurs_at(a, t_c + i) \in S\}$. (Due to the space limitations we cannot give a proof of this statement. The idea is however rather simple. One can notice that at any future moment t such that the goal is not yet satisfied the rule (p3) above and the rule (6) from program $\Pi(N)$ generates all possible sets of occurrences of the agent's actions at t . Other rules of Π_d reject those which do not lead to the plan.)

A plan of minimal length can be found by checking the existence of answer sets of $P_0(1), P_0(2), \dots, P_0(m)$ or by varying the depth of a plan in some other way.

Similar techniques can be used to look for plans in *incomplete domain descriptions*. If, for instance, the values of fluents f_1, \dots, f_k in the initial situation are unknown the planning can be done as follows:

1. Expand $P_0(m)$ by the rules:

$$\begin{aligned} observed(f_i, 0) &\leftarrow not\ observed(\neg f_i, 0). \\ observed(\neg f_i, 0) &\leftarrow not\ observed(f_i, 0). \end{aligned}$$

2. Find an answer set S of the resulting program P'_0 :

3. Let X be the set of propositions of the form $occurs_at(a, t)$ from S and check if $P'_0 \cup X \models_A g$. If the test succeeds then X defines a plan. Otherwise the process can be repeated, i.e. we can look for another answer set. If all answer sets are analyzed and a plan is still not found it does not exist. Alternatively, we can look for conditional plans, incorporate testing of values of fluents into planning or use a variety of other techniques.

A-Prolog can be used to implement planning modules which are different from $PM_0(m)$. For instance, the module $PM_1(m)$ consisting of the first two rules of $PM_0(m)$ and the rules

$$\left. \begin{array}{l} \text{occurs_at}(A, T) \leftarrow t_c \leq T < t_c + m, \\ \quad \text{agent_action}(A), \\ \quad \text{not other_occurs}(A, T), \\ \quad \text{not hold_at}(g, T). \\ \\ \text{other_occurs}(A, T) \leftarrow t_c \leq T < t_c + m, \\ \quad \text{agent_action}(A_1), A \neq A_1, \\ \quad \text{occurs_at}(A_1, T). \end{array} \right\} PM_1$$

restricts the agent's attention to sequential plans. (These rules can be viewed as *A-Prolog* implementation of the choice operator from (Saccà and Zaniolo, 1997).) As before, finding a sequential plan for g of the length bounded by m can be reduced to computing answer sets of

$$P_1(m) = \Pi_d \cup PM_1(m).$$

To illustrate a slightly more sophisticated planning module let us go back to Example 2 and assume that Jack always follows a prudent policy of reporting the loss of his money to the police. This knowledge should be incorporated in Jack's planning module which can be done by adding to it the following rules:

$$\begin{aligned} \text{occurs_at}(\text{report}, t_c) &\leftarrow T_0 < t_c, \\ &\quad \text{happened}(\text{lose}(\text{money}), T_0), \\ &\quad \text{not reported_after}(T_0). \\ \text{reported_after}(T_0) &\leftarrow T_0 < T < t_c, \\ &\quad \text{happened}(\text{report}, T). \end{aligned}$$

The resulting rules allow the agent to create plans which include a simple reactive component.

Now suppose that Jack can buy his ticket either using a credit card or paying cash, and that he prefers to pay cash if he has enough. To represent this information we introduce two new objects, *cash* and *card* and two static causal laws $\text{caused}(\text{has_jack}(\text{money}), [\text{has_jack}(\text{card})])$ and $\text{caused}(\text{has_jack}(\text{money}), [\text{has_jack}(\text{cash})])$. Jack's preference will be represented by adding the following rules to his planning module:

$$\begin{aligned} \text{occurs_at}(\text{pay}(\text{cash}), T) &\leftarrow \text{occurs_at}(\text{get}(O), T), \\ &\quad \text{holds_at}(\text{has_jack}(\text{cash}), T). \\ \text{occurs_at}(\text{pay}(\text{card}), T) &\leftarrow \text{occurs_at}(\text{get}(O), T), \\ &\quad \text{holds_at}(\neg \text{has_jack}(\text{cash}), T). \end{aligned}$$

From now on Jack will plan to pay with cash if possible and use the credit card only as the last resort. Planning modules can also be used to incorporate

heuristic information needed to speed up the planning process. Preliminary experiments show that this allows a very substantial increase in the efficiency of the planning process but a more systematic investigation is needed to make really precise and general claims. It also will be very interesting to do a serious study of the relationship between the planning methods described in this section and satisfiability planning (Kautz and Selman, 1992).

Explaining Observations

Now we demonstrate how answer set programming can be used for finding explanations, i.e. for solving equation 12.3. This can be achieved by finding answer sets of a program

$$E_1(m) = \Pi_d \cup EM_0(m)$$

where m determines the time interval in the past the agent is willing to consider in its search for explanations and $EM_0(m)$ is an explanation module consisting of rules

$$\begin{array}{lcl} \text{unexplained} & \leftarrow & \text{member}(l, O), \\ & & \text{not holds_at}(l, t_c). \\ & \leftarrow & \text{unexplained}. \\ \text{occurs_at}(A, T) & \leftarrow & t_c - m \leq T < t_c, \\ & & \text{exogenous_action}(A), \\ & & \text{not } \neg\text{occurs_at}(A, T). \\ \neg\text{occurs_at}(A, T) & \leftarrow & t_c - m < T < t_c, \\ & & \text{exogenous_action}(A), \\ & & \text{not occurs_at}(A, T). \end{array} \quad \boxed{EM_0(m)}$$

Let S be an answer set of $E_1(m)$ and let $H_0(m)$ be the set of statements of the form $\text{happened}(a_i, t)$ such that $t_c - m \leq t < t_c$, a_i is an elementary exogenous action, $\text{occurs_at}(a_i, t) \in S$ and $\text{happened}(a_i, t) \notin S$ (i.e. this occurrence of a_i has not been explicitly observed). Then $H_0(m)$ is an explanation of O . Moreover, any explanation of O of the depth m can be obtained in this way.

As in the case of the planning modules, the explanation module $EM_0(m)$ can be elaborated and tailored to a particular domain. For instance possible explanations for certain observations may be represented in the agent's knowledge base by a list of statements of the form

$$\text{poss_exp}(l, a)$$

where l is a fluent literal and a is an exogenous action. (Often such a list can be extracted automatically from the corresponding action description). The following explanation module, $EM_1(m)$ uses this information to explain a single unexplained observation l by an occurrence of a single exogenous action

in the interval $[t_0, t_c)$ where $t_0 = t_c - m$:

$\leftarrow \neg \text{holds_at}(l, t_c).$	$\left. \begin{array}{l} \leftarrow \text{impossible_if}(A, P), \\ \text{hold_at}(P, T). \end{array} \right\} EM_1(m)$
$\text{impossible}(A, T) \leftarrow \text{impossible_if}(A, P),$	
$\text{hold_at}(P, T).$	
$\text{occurs_at}(A, T) \leftarrow t_0 \leq T < t_c,$	
$\text{poss_exp}(l, A),$	
$\neg \text{impossible}(A, T),$	
$\neg \text{other_occurs}(A, T).$	
$\text{other_occurs}(A, T) \leftarrow t_0 \leq T, T' < t_c,$	
$T' \neq T,$	
$\text{occurs_at}(A, T'),$	
$\neg \text{happened}(A, T').$	
$\text{other_occurs}(A, T) \leftarrow t_0 \leq T, T' < t_c,$	
$A \neq A',$	
$\text{occurs_at}(A', T'),$	
$\neg \text{happened}(A', T').$	

Let S be an answer set of a program

$$E_2(m) = \Pi_d \cup EM_1(m)$$

and let $H_0(m)$ be the set of statements of the form $\text{happened}(a_i, t)$ such that $t_0 \leq t < t_c$, a_i is an elementary exogenous action, $\text{occurs_at}(a_i, t) \in S$ and $\text{happened}(a_i, t) \notin S$. Then $H_0(m)$ is an explanation of O . Moreover, any explanation of O consisting of an occurrence of a single elementary exogenous action in the interval $[t_0, t_c]$ can be obtained in this way.

Checking the entailment

Now let us consider an agent's reactive component. Implementation of this component requires checking the condition

$$\Gamma \models_A \text{currently}(l) \tag{12.5}$$

It can be shown that this can be reduced to checking the condition

$$\Pi_d \models \text{currently}(l) \tag{12.6}$$

As before, this task can be accomplished by the constraint satisfaction approach in answer set programming. For a very broad class of deterministic domain descriptions this condition can be also checked by using more traditional Prolog or XSB systems with a query $\text{currently}(l)$ and a simple modification of program Π_d as an input. The modification requires addition of information about types of variables in the bodies of some rules. It is needed to make sure that reasoning done by Prolog (XSB) on Π_d is sound and complete with respect to

queries of the type *currently(l)*. (The corresponding proof follows the lines of the similar proof in (Baral et al., 1997) and demonstrates that for this input the Prolog interpreter always terminates, does not flounder, and does not require the occur check. This allows to reduce the proof of soundness and completeness of the interpreter to soundness and completeness of SLDNF resolution.) Ramifications of the choice of inference engine used for solving this and other problems are not entirely clear and require further investigation. We believe however that in this particular case diversity can be beneficial.

Finally, let us look at the situation in which the agent found a plan a_1, \dots, a_n , executed action a_1 , modified Γ to record new observations and discovered that its goal, g , is not changed. In this case it seems natural to start the planning with simply checking if

$$\Gamma \models_A \text{holds_after}(g, [a_n, \dots, a_2]) \quad (12.7)$$

As before for complete, deterministic, and acyclic domain descriptions (Watson, 1999a) this can be done by running a Prolog interpreter on the query

$$\text{holds_after}(g, [a_n, \dots, a_2], T) \quad (12.8)$$

and program Π_c obtained by expanding Π_d by rules

$\text{impossible}(A, S, T)$	$\leftarrow \text{impossible_if}(A, P)$	}
	$\text{hold_after}(P, S, T).$	
$\text{possible}(A, S, T)$	$\leftarrow \text{not impossible}(A, S, T).$	
$\text{holds_after}(L, [], T)$	$\leftarrow \text{holds_at}(L, T).$	
$\text{holds_after}(L, [A S], T)$	$\leftarrow \text{possible}(A, S, T),$	
	$\text{causes}(A, L, P),$	
	$\text{hold_after}(P, S, T).$	
$\text{holds_after}(L, S, T)$	$\leftarrow \text{caused}(L, P),$	
	$\text{hold_after}(P, S, T).$	
$\text{holds_after}(L, [A S], T)$	$\leftarrow \text{possible}(A, S, T),$	
	$\text{holds_after}(L, S, T),$	
	$\text{not holds_after}(\overline{L}, [A S], T).$	

As before typing information should be added to the program to guarantee soundness and completeness of this method.

4 CONCLUSION

We have proposed a model of a simple intelligent agent acting in a changing environment. A characteristic feature of this model is its extensive use of a declarative language *A-Prolog*. The language is used for describing the agent's knowledge about the world and its own abilities as well as for precise mathematical characterization of the agent's tasks. We demonstrated how the agent's reasoning tasks can be formulated as questions about programs

of *A-Prolog* and how these questions can be answered using various logic programming algorithms. A high expressiveness of the language allows one to represent rather complex domains not easily expressible in more traditional languages. The corresponding knowledge bases have a reasonably high degree of elaboration tolerance. Recent advances in logic programming theory allow one to prove correctness of these algorithms for a broad range of domains. New logic programming systems allow their implementation. Four such systems, CCALC, DeRes, DLV, and Smodels, were demonstrated at the LBAI workshop. These systems were used to find plans from Example 2 as well as to solve much more complex tasks. Some experimental results aimed at testing the efficiency of the system are rather encouraging. The current rate of improvement of the systems performance and rapid advances in our understanding of methodology of programming in *A-Prolog* allow us to believe in the practicality of this approach. Some applications using XSB, DLV, Smodels and CCALC can be found in (Watson, 1999b; Soininen and Niemela, 1999; Erdem et al., 2000; McCain and Turner, 98; Cui et al., 1999) and the systems can be reached from <http://www.cs.utexas.edu/users/tag/>.

Acknowledgments

We would like to thank V. Lifschitz, E. Erdem, M. Nogueira, J. Minker and the referees for their useful comments.

References

- Baral, C. (1995). Reasoning about Actions: Non-deterministic effects, Constraints and Qualification. In Mellish, C., editor, *Proc. of IJCAI 95*, pages 2017–2023. Morgan Kaufmann.
- Baral, C. and Gelfond, M. (1994). Logic programming and knowledge representation. *Journal of Logic Programming*, 19,20:73–148.
- Baral, C., Gelfond, M., and Provetti, A. (1995). Representing Actions I: Laws, Observations and Hypothesis. In *Proc. of AAAI 95 Spring Symposium on Extending Theories of Action: Formal theory and practical applications*.
- Baral, C., Gelfond, M., and Provetti, A. (1997). Representing Actions: Laws, Observations and Hypothesis. *Journal of Logic Programming*, 31(1-3):201–243.
- Baral, C. and Son, T. (1998). Relating theories of actions and reactive control. *Electronic transactions on Artificial Intelligence*, 2(3-4).
- Chen, W., Swift, T., and Warren, D. (1995). Efficient top-down computation of queries under the well-founded semantics. *Journal of Logic Programming*, 24(3):161–201.
- Cholewiński, P., Marek, W., and Truszczyński, M. (1996). Default reasoning system DeRes. In Aiello, L., Doyle, J., and Shapiro, S., editors, *Proc. of KR 96*, pages 518–528. Morgan Kaufmann.
- Clark, K. (1978). Negation as failure. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York.

- Cui, B., Swift, T., and Warren, D. (1999). A case study in using preference logic grammars for knowledge representation. In Gelfond, M., Leone, N., and Pfeifer, G., editors, *Proc. of LPNMR 99*, pages 206–220. Springer.
- Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (1997). Complexity and expressive power of logic programming. In *Proc. of 12th annual IEEE conference on Computational Complexity*, pages 82–101.
- De Giacomo, G., Lesperance, Y., and Levesque, H. (1997). Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *IJCAI 97*, pages 1221–1226. Morgan Kaufmann.
- DeGiacomo, G., Reiter, R., and Soutchanski, M. (1998). Execution monitoring of high-level robot programs. In Cohn, A., Schubert, L., and Shapiro, S., editors, *Proc. of KR 98*, pages 453–464. Morgan Kaufmann.
- Denecker, M. and De Schreye, D. (1998). SLDNFA: an abductive procedure for normal abductive logic programs. *Journal of Logic Programming*, 34(2):111–167.
- Erdem, E., Lifschitz, V., and Wong, M. (2000). Wire routing and satisfiability planning. In *Proc. CL-2000 (to appear)*.
- Faber, W., Leone, N., and Pfeifer, G. (1999). Pushing goal derivation in DLP computations. In Gelfond, M., Leone, N., and Pfeifer, G., editors, *Proc. of LPNMR 99*, pages 177–191. Springer.
- Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski, R. and Bowen, K., editors, *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, pages 1070–1080. MIT Press.
- Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–387.
- Gelfond, M. and Lifschitz, V. (1992). Representing actions in extended logic programs. In Apt, K., editor, *Joint International Conference and Symposium on Logic Programming*, pages 559–573. MIT Press.
- Guinchiglia, E. and Lifschitz, V. (1998). An action language based on causal explanation: Preliminary report. In *Proc. AAAI-98*, pages 623–630. MIT Press.
- Kakas, A., Kowalski, R., and Toni, F. (1998). The role of abduction in logic programming. In Gabbay, D., Hogger, C., and Robinson, J., editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. Oxford University Press.
- Kautz, H. and Selman, B. (1992). Planning as satisfiability. In *Proc. of ECAI-92*, pages 359–363.
- Kowalski, R. (1995). Using metalogic to reconcile reactive with rational agents. In Apt, K. and Turini, F., editors, *Meta-logics and logic programming*, pages 227–242. MIT Press.
- Kowalski, R. and Sadri, F. (1999). From logic programming towards multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, 25:391–419.
- Levesque, H., Reiter, R., Lesperance, Y., Lin, F., and Scherl, R. (1997). GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–84.

- Lifschitz, V. (1996). Foundations of declarative logic programming. In Brewka, G., editor, *Principles of Knowledge Representation*, pages 69–128. CSLI Publications.
- Lifschitz, V. (1997). Two components of an action language. *Annals of Math and AI*, 21(2-4):305–320.
- Lin, F. (2000). From causal theories to successor state axioms and STRIPS like systems. In *Proc. of AAAI 2000 (to appear)*.
- Lin, F. (95). Embracing causality in specifying the indirect effects of actions. In Mellish, C., editor, *Proc. of IJCAI 95*, pages 1985–1993. Morgan Kaufmann.
- Marek, W. and Truszczyński, M. (1989). Stable semantics for logic programs and default reasoning. In Lusk, E. and Overbeek, R., editors, *Proc. of the North American Conf. on Logic Programming*, pages 243–257. MIT Press.
- Marek, W. and Truszczyński, M. (1993). *Nonmonotonic Logic: Context dependent reasoning*. Springer.
- Marek, W. and Truszczyński, M. (1999). Stable models and an alternative logic programming paradigm. In Apt, K., Marek, V., Truszczynski, M., and Warren, D., editors, *The Logic Programming Paradigm: a 25-Year perspective*, pages 375–398. Springer.
- McCain, N. and Turner, H. (95). A causal theory of ramifications and qualifications. In Mellish, C., editor, *Proc. of IJCAI 95*, pages 1978–1984. Morgan Kaufmann.
- McCain, N. and Turner, H. (98). Satisfiability planning with causal theories. In Cohn, A., Schubert, L., and Shapiro, S., editors, *Proc. of KR 98*, pages 212–223. Morgan Kaufmann.
- McCarthy, J. (1980). Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1, 2):27–39,171–172.
- McCarthy, J. and Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B. and Michie, D., editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh.
- Moore, R. (1985). Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94.
- Niemela, I. (1999). Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–271.
- Niemela, I. and Simons, P. (1997). Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In Dix, J., Furbach, U., and Nerode, A., editors, *Proc. 4th international conference on Logic programming and non-monotonic reasoning*, pages 420–429. Springer.
- Pinto, J. (1999). Compiling Ramification Constraints into Effect Axioms. *Computational Intelligence*, 15(3):280–307.
- Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13(1,2):81–132.
- Reiter, R. (1995). On specifying database updates. *Journal of Logic Programming*, 25:25–91.
- Saccà, D. and Zaniolo, C. (1997). Deterministic and non-deterministic stable models. *Journal of Logic and Computation*, 7(5):555–579.

- Sandewall, E. (1998). Special issue. *Electronic Transactions on Artificial Intelligence*, 2(3-4):159–330. <http://www.ep.liu.se/ej/etai/>.
- Shanahan, M. (1997). *Solving the frame problem: A mathematical investigation of the commonsense law of inertia*. MIT press.
- Soininen, T. and Niemela, I. (1999). Developing a declarative rule language for applications in product configuration. In Gupta, G., editor, *Proc. of Practical Aspects of Declarative Languages '99*, volume 1551, pages 305–319. Springer.
- Van Gelder, A., Ross, K., and Schlipf, J. (1991). The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650.
- Wang, H. and Zaniolo, C. (2000). Nonmonotonic Reasoning in LDL++. In Minker, J., editor, *This volume*. Kluwer.
- Watson, R. (1999a). *Action languages and domain modeling*. PhD thesis, University of Texas at El Paso.
- Watson, R. (1999b). An application of action theory to the space shuttle. In Gupta, G., editor, *Proc. of Practical Aspects of Declarative Languages '99*, volume 1551, pages 290–304. Springer.

Chapter 13

DYNAMIC LOGIC FOR REASONING ABOUT ACTIONS AND AGENTS

J.-J.Ch. Meyer

Utrecht University

Department of Computing Science

Intelligent Systems Group

P.O. Box 80.089

3508 TB Utrecht, The Netherlands

Abstract Dynamic logic is a logic to reason about the dynamics of (natural or artificial) systems in general, ranging from the effects of actions of human agents to the behavior of artificial agents and software systems. Therefore it is to be expected that in AI it can be fruitfully employed both to represent knowledge about the dynamics of the domain at hand as well as to describe/specify (the dynamic behavior of) AI systems themselves. A typical example of the former is the description of the effects of actions (of humans, for example) in the commonsense world, while the specification of a particular reasoning system would be of the latter type. In this paper a number of examples are given to illustrate the usefulness (and wide scope!) of dynamic logic for AI.

Keywords: Dynamic logic, dynamic deontic logic, dynamic update logic, logic of actions, single agents, multi-agent systems

1 INTRODUCTION

Originally, dynamic logic was proposed in computer science as a logic for reasoning about programs in order to verify their correctness. Later it was realised that dynamic logic could also be fruitfully employed as a logic for reasoning about actions in general. Recently in AI and computer science the concept of an intelligent agent has become popular, with both intelligent robots and software agents (softbots) as intended applications. Since the various attitudes an intelligent agent is supposed to possess can be captured as actions operating on a complex mental state, for the logical description of (the behavior of) agents one might also resort to dynamic logic (which should

then be combined with other modal logics to provide a description of an agent's mental state).

In this paper some examples are given illustrating how dynamic logic might constitute a basis of a logic for reasoning about actions and agents. These include a range of applications, from database updates to the description of agent attitudes (belief revision, commitments, obligations) and reasoning about actions in a commonsense environment, where we touch upon the 'infamous' frame problem. The paper contains a personal choice of topics and the paper is not meant as a complete overview of dynamic logic and its uses. Furthermore, the topics that are treated are only sketched, without providing all the sordid details, to get a flavor of them, since a full treatment is beyond the scope of the present paper. (For this the reader is referred to other papers.)

2 DYNAMIC LOGIC AS A BASIC LOGIC OF ACTIONS

Dynamic logic is a modal logic especially designed to reason about actions. Historically it dates back to work by Vaughan Pratt (Pratt, 1976), Bob Moore (Moore, 1985), and David Harel (Harel, 1979; Harel, 1984), and it has been used for reasoning about programs, thus providing a formalism for program verification and specification (Kozen and Tiuryn, 1990; Cousot, 1990).¹ It is very much akin to Hoare's logic (Hoare, 1969) for program correctness, and can in fact be considered as a generalization of this logic. In this section we will describe the basic idea behind an elementary form of (propositional) dynamic logic², which will serve as a basis for the later logics in this paper.

2.1 LANGUAGE

For the purpose of this basic treatment we introduce the following logical language \mathcal{L}_{DL} . Assume a set \mathcal{P} of propositional atoms, and a set \mathcal{A} of atomic actions. We will use p, q (with possible marks and indices) to denote elements from \mathcal{P} , and a, b (with possible marks and indexes) to denote elements from \mathcal{A} .

Definition 1 The logical language \mathcal{L}_{DL} and action language \mathcal{L}_{ACT} are given as the least sets closed under the following constraints:

1. $\mathcal{P} \subseteq \mathcal{L}_{DL}$
2. $\mathcal{A} \subseteq \mathcal{L}_{ACT}$
3. $\varphi, \psi \in \mathcal{L}_{DL}$ implies $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \varphi \rightarrow \psi, \varphi \leftrightarrow \psi \in \mathcal{L}_{DL}$

¹We note that the term "dynamic logic" is somewhat overloaded, since in particular the Amsterdam School in logic employs the term for a dynamic interpretation of classical (non-modal) logic, mostly used in linguistic applications, such as for a formal treatment of discourses (Groenendijk and Stokhof, 1991). This branch of "dynamic logic" will not be treated in this paper, although we'll touch on it when we'll consider multiple agent updates in Section 7.2.1.

²Propositional Dynamic Logic or PDL is due to Fischer & Ladner (Fischer and Ladner, 1979)

4. $\varphi \in \mathcal{L}_{DL}$, $\alpha \in \mathcal{L}_{ACT}$ implies $[\alpha]\varphi, (\alpha)\varphi \in \mathcal{L}_{DL}$
5. $\varphi \in \mathcal{L}_{DL}$ implies $\varphi? \in \mathcal{L}_{ACT}$
6. $\alpha, \beta \in \mathcal{L}_{ACT}$ implies $\alpha; \beta, \alpha + \beta, \alpha^* \in \mathcal{L}_{ACT}$

Here the logical language \mathcal{L}_{DL} is an extension of propositional logic with modal operators of the form $[\alpha]$ (and duals (α)) where α is an element from the action language \mathcal{L}_{ACT} . The intended reading of these modal operators are: ' $[\alpha]\varphi$ ' is read as "after performance (execution) of the action α it *necessarily* holds that φ ", or "*all* ways of executing α lead to a situation (state) in which φ is true", while ' $(\alpha)\varphi$ ' is read as "after performance (execution) of action α it *possibly* holds that φ ", or in other words, "*there exists* a way that execution of α will end up in a situation (state) where φ holds. The action language \mathcal{L}_{ACT} is here taken to be a very basic programming language, viz. that of the regular expressions over 'action alphabet' A . The ';' operator denotes sequential composition ('followed by'), '+' means (nondeterministic) choice, and '*' denotes arbitrary finite repetition. (This choice of operators is 'classic' for dynamic logic (cf. (Harel, 1984; Kozen and Tiuryn, 1990; Stirling, 1992)). In this paper different operators such as '&' (parallel composition) and '-' (action negation) are also employed; in this section, however, we will restrict ourselves to the familiar ones.) Finally the *action* expression $\varphi?$ stands for a test whether the *logical* expression φ holds in the current state. This is typically used in an expression involving a choice operator to guide control of this choice, as in e.g. the expression $p?; a + \neg p?; b$, where $p \in \mathcal{P}$ and $a, b \in A$. (This expression is the representation in our action language of the familiar '*if p then a else b fi*' construct in imperative programming languages.)

2.2 MODELS

The language \mathcal{L}_{DL} is given an interpretation on the basis of Kripke-models, as is usual for a modal language. We will use *tt* and *ff* for the classical truth values. Formally a Kripke model for language \mathcal{L}_{DL} is a structure of the following form:

Definition 2 A Kripke model for \mathcal{L}_{DL} is a structure \mathcal{M} of the form $\langle S, \pi, r \rangle$, where

- S is a non-empty set (the set of *states*);
- $\pi : S \rightarrow (\mathcal{P} \rightarrow \{\text{tt}, \text{ff}\})$ is a truth assignment function to the atoms per state;
- $r : \mathcal{L}_{ACT} \rightarrow 2^{S \times S}$ are state transition relations per action, satisfying the following properties:
 1. $r(\varphi?) = \{<s, s> \mid \mathcal{M}, s \models \varphi\}$, where $\mathcal{M}, s \models \varphi$ is defined below;
 2. $r(\alpha; \beta) = r(\alpha) \circ r(\beta)$, where \circ stands for the relational composition;
 3. $r(\alpha + \beta) = r(\alpha) \cup r(\beta)$;

4. $r(\alpha^*) = r(\alpha)^*$, where $*$ stands for the reflexive, transitive closure operator on relations

Truth of a formula $\varphi \in \mathcal{L}_{DL}$ in a state $s \in S$ in a model $\mathcal{M} = \langle S, \pi, r \rangle$ (written $(\mathcal{M}, s) \models \varphi$), is defined by the following.

Definition 3 Let $\mathcal{M} = \langle S, \pi, r \rangle$ be a given model and $s \in S$. Then:

- $\mathcal{M}, s \models p$ if, and only if, $\pi(s)(p) = tt$, for $p \in \mathcal{P}$;
- $\mathcal{M}, s \models \neg\varphi$ if, and only if, not $\mathcal{M}, s \models \varphi$;
- $\mathcal{M}, s \models \varphi \wedge \psi$ if, and only if, $\mathcal{M}, s \models \varphi$ and $\mathcal{M}, s \models \psi$;
- $\mathcal{M}, s \models \varphi \vee \psi$ if, and only if, $\mathcal{M}, s \models \varphi$ or $\mathcal{M}, s \models \psi$;
- $\mathcal{M}, s \models \varphi \rightarrow \psi$ if, and only if, $\mathcal{M}, s \models \varphi$ implies $\mathcal{M}, s \models \psi$;
- $\mathcal{M}, s \models \varphi \leftrightarrow \psi$ if, and only if, $\mathcal{M}, s \models \varphi$ bi-implies $\mathcal{M}, s \models \psi$;
- $\mathcal{M}, s \models [\alpha]\varphi$ if, and only if, $\mathcal{M}, s' \models \varphi$ for all s' with $r(\alpha)(s, s')$;
- $\mathcal{M}, s \models \langle \alpha \rangle \varphi$ if, and only if, $\mathcal{M}, s' \models \varphi$ for some s' with $r(\alpha)(s, s')$;

A formula φ is *valid in a model* $\mathcal{M} = \langle S, \pi, r \rangle$, denoted $\mathcal{M} \models \varphi$, if $\mathcal{M}, s \models \varphi$ for every $s \in S$. A formula φ is *valid* with respect to a set MOD of models, denoted $MOD \models \varphi$, if $\mathcal{M} \models \varphi$ for every model $\mathcal{M} \in MOD$. If MOD is the set of *all* Kripke models of the above form, we generally write $\models \varphi$ instead of $MOD \models \varphi$.

2.3 THE LOGIC PDL

The simple propositional dynamic logic introduced above can be finitely axiomatized by the following system:

- any axiomatization of propositional logic
- $[\alpha](\varphi \rightarrow \psi) \rightarrow ([\alpha]\varphi \rightarrow [\alpha]\psi)$
- $[\varphi?] \psi \leftrightarrow (\varphi \rightarrow \psi)$
- $[\alpha; \beta]\varphi \leftrightarrow [\alpha][\beta]\varphi$
- $[\alpha + \beta]\varphi \leftrightarrow [\alpha]\varphi \wedge [\beta]\varphi$
- $[\alpha^*]\varphi \rightarrow \varphi$
- $[\alpha^*]\varphi \rightarrow [\alpha][\alpha^*]\varphi$
- $[\alpha^*](\varphi \rightarrow [\alpha]\varphi) \rightarrow (\varphi \rightarrow [\alpha^*]\varphi)$
- $[\alpha]\varphi \leftrightarrow \neg\langle\alpha\rangle\neg\varphi$

and rules modus ponens (MP) and necessitation:

$$\frac{\varphi}{[\alpha]\varphi}$$

Theorem 1 The above axiomatic system is sound and complete with respect to the class of Kripke models of Definition 2.

We note that taking different choices for the basic operators in the action language, and especially further extensions to these, has a large influence on the axiomatization. This is the reason that sometimes a quite different axiomatic system is obtained. However, as the above system is standard in the literature (e.g. (Stirling, 1992)), we use it here as a basis.

3 DYNAMIC UPDATE LOGIC

Since dynamic logic is suited *par excellence* for reasoning about the dynamics of systems, it may also be used for the description of performing updates on a database (or information system more in general). In (Spruit et al., 1995) this idea is elaborated. Here the logic Propositional Dynamic Database Logic (PDDL) is proposed for rather simple update programs.³ These programs are in fact of the form of regular programs as in general propositional dynamic logic, where the atomic actions are now instantiated to updates of the form Ip and Dp for atoms p , denoting insertion and deletion of this atomic fact, respectively. Actually, there are two versions of these updates, *passive* ones and *active* ones, where the latter are parametrized by a logic program H , serving as a background theory, and which are denoted as $T^H p$ and $D^H p$. Here a logic program is a finite set of program clauses of the form $p \leftarrow p_1, \dots, p_n$, where all p_i and p are atoms. Given a logic program H , we define the set $Der(H)$ of derived atoms in H as the set of all atoms occurring as a head of some clause in H , and the set $Base(H)$ of base atoms in H as $\mathcal{P} \setminus Der(H)$. Thus, base atoms do not appear as heads of clauses of the logic program, and are therefore not defined (in the sense of derived) by means of the program.

The semantics of this logic is as in the general case, only now the specific atomic actions should be catered for. To this end we need some additional definitions.

Definition 4 ■ Let $\mathcal{M} = \langle S, \pi, r \rangle$ be a Kripke model. The function $prop : S \rightarrow 2^{\mathcal{P}}$ is given by $prop(s) = \{p \in \mathcal{P} \mid \pi(s)(p) = tt\}$.

- A Kripke model $\mathcal{M} = \langle S, \pi, r \rangle$ is *full* if, and only if, for all $P \subseteq \mathcal{P}$ there exists a state $s \in S$ such that $prop(s) = P$.
- Let H be a logic program. The function $min^H : 2^{\mathcal{P}} \rightarrow 2^{\mathcal{P}}$ is given by: $min^H(P)$ is the minimal set P' (w.r.t. set inclusion) such that

³We must mention here that work in a similar vein appears in (Manchanda and Warren, 1988). In this paper elementary updates inserting and deleting base atoms are used like the ones employed in this section, together with update rules of the form $(E) \leftarrow C_1 \wedge (E_1)(C_2 \wedge (E_2)(\dots))$, expressing that the update E is defined in terms of updates E_i , where every C_i is a conjunction of atoms.

$(P \cap \text{Base}(H)) \subseteq P'$ and such that if $p \leftarrow p_1, \dots, p_n \in H$ and $p_1, \dots, p_n \in P'$ then also $p \in P'$.

Now the semantics of (passive and active) insertion and deletion reads:

Definition 5 ■ $r(Ip) = \{<s, s'> \mid \text{prop}(s') = \text{prop}(s) \cup \{p\}\}$

■ $r(Dp) = \{<s, s'> \mid \text{prop}(s') = \text{prop}(s) \setminus \{p\}\}$

■ $r(I^H p) = \{<s, s'> \mid \text{prop}(s') = \min^H(\text{prop}(s) \cup \{p\})\}$

■ $r(D^H p) = \{<s, s'> \mid \text{prop}(s') = \min^H(\text{prop}(s) \setminus \{p\})\}$

The axiomatization of this logic includes the following axioms (besides the familiar ones from PDL):

- $[Ip]p$
- $[Dp]\neg p$
- $\langle Ip \rangle \text{tt}$
- $\langle Dp \rangle \text{tt}$

One also has to deal with non-changes using so-called *frame axioms*:

- $q \rightarrow [Ip]q$, for $q \neq p$
- $\neg q \rightarrow [Ip]\neg q$, for $q \neq p$
- $q \rightarrow [Dp]q$, for $q \neq p$
- $\neg q \rightarrow [Dp]\neg q$, for $q \neq p$

As to active updates we have analogous axioms:

- $[I^H p]p$
- $[D^H p]\neg p$
- $\langle I^H p \rangle \text{tt}$
- $\langle D^H p \rangle \text{tt}$
- $q \rightarrow [Ip]q$, for $q \in \text{Base}(H) \setminus \{p\}$
- $\neg q \rightarrow [Ip]\neg q$, for $q \in \text{Base}(H) \setminus \{p\}$
- $q \rightarrow [Dp]q$, for $q \in \text{Base}(H) \setminus \{p\}$
- $\neg q \rightarrow [Dp]\neg q$, for $q \in \text{Base}(H) \setminus \{p\}$

However, to deal with derived atoms we need further definitions:

Definition 6 Let H be a logic program.

- Let $q \in \text{Der}(H)$ such that

$$q \leftarrow p_{11}, \dots, p_{1n_1}$$

...

$$q \leftarrow p_{m1}, \dots, p_{mn_m}$$

are all program clauses in H with q as head. Then the *completion formula* of q in H , denoted H_q , is defined as:

$$H_q = (p_{11} \wedge \dots \wedge p_{1n_1}) \vee \dots \vee (p_{m1} \wedge \dots \wedge p_{mn_m})$$

- The binary ('direct dependency') relation $\text{dep}_H \subseteq \mathcal{P} \times \mathcal{P}$ is defined as: $(p, q) \in \text{dep}_H$ if, and only if, there is a program clause $p \leftarrow q_1, \dots, q_n \in H$ such that $q = q_i$ for some i . (Informally, $(p, q) \in \text{dep}_H$ is read as "p is directly dependent on q within program H".) The relation dep_H^+ is the transitive closure of dep_H .⁴
- An atom p is said to be *recursive* in H if there is an atom q such that $(p, q) \in \text{dep}_H^+$ and $(q, q) \in \text{dep}_H^+$. The set of recursive atoms in H is denoted $\text{Rec}(H)$. H is called recursive if $\text{Rec}(H) \neq \emptyset$.
- A *mutual dependency group* of H is non-empty subset P of $\text{rec}(H)$ such that $\forall p, q \in P : (p, q) \in \text{dep}_H^+$.

Now we have the following axioms pertaining to derived atoms:

- $[I^H p](q \leftarrow H_q)$, for $q \in \text{Der}(H)$
- $[D^H p](q \leftarrow H_q)$, for $q \in \text{Der}(H)$
- $[I^H p](\vee_{q \in P} q \rightarrow \vee_{q \in P} H_q[\text{ff}/P])$
for P a mutual dependency group of H
- $[D^H p](\vee_{q \in P} q \rightarrow \vee_{q \in P} H_q[\text{ff}/P])$
for P a mutual dependency group of H

(Here $\varphi[\text{ff}/P]$ stands for the formula φ in which all occurrences of atoms $p \in P$ are replaced by ff .)

The last two axioms are minimization axioms: they state that if recursive atoms are true, then there must be an "external reason" for this; the atoms in a mutual dependency group cannot be true only because other atoms in this group are true. If a recursive atom in such a group is true, then the completion formula for some atom in the group must be true, even if the recursive atoms in the completion are replaced by ff .

⁴This notion of dependency was introduced in (Apt et al., 1988) to deal with stratified logic programs with negation as failure.

In (Spruit et al., 1995) it is shown that this axiomatization of PDDL is complete with respect to so-called *full* structures. (In that paper also a complete axiomatization is given for *all* structures, but we do not elaborate on this here.) Next in (Spruit et al., 1999) this idea is generalized to the first-order case, which in the context of databases is almost a ‘must’. In fact in this paper two update logics are proposed, both based on (first-order) dynamic logic.

The first update logic is Relational Algebra Update Logic (RAUL), which can be viewed as an extension of relational algebra with assignment. The second logic is Dynamic Database Logic (DDL) in which the atomic updates are updates of the extension of predicates and assignments, which are updates to the values of attributes construed as updatable functions. DDL can be viewed as a true extension of PDDL to the first-order case. We will not go into this here further, but refer to (Spruit et al., 1999) for this, and restrict ourselves here to a brief sketch of RAUL.

In RAUL we consider as actions typically assignments of the form ‘ $p := e$ ’, where p is a predicate symbol, and e is a relational algebra expression, which has a relation (i.e., a set of tuples) as its meaning. Relational algebra expressions are of the form ‘ $e_1 \cup e_2$ ’, ‘ $e_1 \setminus e_2$ ’, ‘ $e_1 \times e_2$ ’, ‘ $e[k_1, \dots, k_n]$ ’, ‘ e where φ ’, denoting union, difference, product, projection and selection (here φ is a test with which one can test whether attributes are equal to a term or to each other), respectively. We omit here the formal semantics of these expressions, and go straight to the logic. Typical axioms in RAUL include: (Here we use pT, qT to denote formulas in which p, q are predicate symbols and T is a tuple of terms matching the arity of p, q ; likewise fT denotes a term in which f is a function symbol and T is a tuple of terms matching the arity of f .)

- $qT \rightarrow [p := q]pT$
- $\neg qT \rightarrow [p := q]\neg pT$
- $[p := e_1 \cup e_2]pT \leftrightarrow ([p := e_1]pT \vee [p := e_2]pT)$
- $[p := e_1 \setminus e_2]pT \leftrightarrow ([p := e_1]pT \wedge \neg[p := e_2]pT)$
- $[p := e_1 \times e_2]pT_1 T_2 \leftrightarrow ([p_1 := e_1]p_1 T_1 \wedge [p_2 := e_2]p_2 T_2)$
- $\langle p := e \rangle tt$

Again, one has to deal with non-changes by use of frame axioms:

- $qT \rightarrow [p := e]qT$, for $q \neq p$
- $fT = t \rightarrow [p := e]fT = t$

expressing that, by assigning to an updatable predicate, other updatable predicates and updatable functions do not change.

4 DYNAMIC SEMANTICS OF REASONING SYSTEMS

In the previous section we have seen how dynamic logic can be applied to describe the dynamics of updating a database or an information system.

In the case of active updates this also involved reasoning already due to the logic program stored in the information system that served as a background theory, in particular pertaining to derived atoms. Of course one may turn to a more advanced system of reasoning. Also such more advanced reasoning systems themselves can be analyzed by means of dynamic logic. The basic idea behind this is that reasoning systems like any other dynamic system display a certain behavior that can also be thought of being brought about by reasoning steps. These may be ‘classical steps’ like applying deduction, but—and in AI this is even more important—also reasoning steps in a ‘non-standard’ logic or reasoning system. One may, for instance, think of default reasoning, but there are many other examples as well.

4.1 DESCRIPTIVE DYNAMIC LOGIC

For example, Sierra *et al.* (Sierra et al., 1996) have applied dynamic logic for analyzing/describing reasoning by means of a reflective (multi-language) architecture, where typical reasoning steps include ‘reflection’ steps, such as moving from an object level to a meta-level where, for example, one may reason about one’s knowledge and ignorance on object level assertions. Generally in multi-language architectures one employs so-called *bridge rules* to infer, transfer or translate information between two different *units* (that is to say, languages or, if one thinks more in terms of an implementation of such an architecture, modules). Of course inferences within a unit may also take place.

In their set-up called Descriptive Dynamic Logic they tailor (propositional) dynamic logic to suit their purpose by taking as atomic formulas so-called “quoted” formulas of the form $k : [\varphi]$, where k is a unit identifier denoting a unit u_k of the system, and φ is a formula in the language of that unit. ‘Subatomic’ programs are inferences (deduction steps) by means of (intra-unit or inter-unit) inference rules, written as $[\Gamma \vdash_{kl} \varphi]$, where k and l refer to the units u_k and u_l in the system, respectively (if $k = l$, we have an intra-unit inference), and Γ is a set of formulas in the language of u_k and φ is a formula in the language of u_l .

Atomic programs in the sense of propositional dynamic logic are now taken to be non-deterministic choices of subatomic actions.⁵ For instance, we have atomic actions such as $\vdash_{kl} = \bigcup\{\alpha \mid \alpha \text{ is of the form } [\Gamma \vdash_{kl} \varphi]\}$, where \bigcup stands for the nondeterministic choice of the subatomic actions that are given to it as arguments.

In this language one may express the case where a unit u_k is endowed with the modus ponens inference rule by the (valid) formula:

$$MP : (\beta)(k : [\varphi] \wedge k : [\varphi \rightarrow \psi]) \rightarrow (\beta; \vdash_{kk}^{mp})k : [\psi],$$

where the modus ponens program $\vdash_{kk}^{mp} = \bigcup_{\gamma, \delta} [\{\gamma, \gamma \rightarrow \delta\} \vdash_{kk} \delta]$.

Other typical axioms in this logic include:

⁵We employ here a somewhat different terminology than in (Sierra et al., 1996) to stay in line with the rest of our paper.

- $\langle [\Gamma \vdash_{ij} \psi] \rangle tt \leftrightarrow \bigwedge_{\varphi \in \Gamma} i : [\varphi]$
- $[[\Gamma \vdash_{ij} \psi]]j : [\psi]$
- $\langle [\Gamma \vdash_{ij} \psi] \rangle p \rightarrow p$, for atomic proposition $p \neq j : [\psi]$

The first one expresses that an application of a deduction step (viewed as a program) $[\Gamma \vdash_{ij} \psi]$ succeeds if and only if the formulas in the antecedent Γ are all established within unit u_i . The second one says that an application of the deduction step $[\Gamma \vdash_{ij} \psi]$ (when it succeeds) results in a state where the conclusion ψ is established in unit u_j . The third is again a kind of frame axiom: after application of a step $[\Gamma \vdash_{ij} \psi]$ every atom p different from $j : [\psi]$ that is true was already true before application (informally, of course, since the step has no bearing on such an atom).

In (Sierra et al., 1996) a complete axiomatization of the logic is given. Furthermore it is also indicated how this logic might be used to describe a reflective architecture for non-standard reasoning methods such as default reasoning.

5 DYNAMIC DEONTIC LOGIC

In this section we consider another application of dynamic logic, viz. reasoning about the deontics of actions. Deontic logic itself is an old branch of philosophical (modal) logic dating back to Mally (Mally, 1926), but which has become a serious subject of study since the seminal work of Von Wright (von Wright, 1951; von Wright, 1964). Deontic logic is the logic of obligation, prohibition, and permission and can, in principle, be applied both to obligated, forbidden or permitted states of affairs and obligated, forbidden or permitted actions. In the literature these two forms of deontic logic have been referred to as *ought-to-be* and *ought-to-do* logic, respectively, and it is mostly not made explicit which form is intended. In (Meyer, 1988) an explicit ought-to-do logic has been proposed, based on dynamic logic. The main idea here was inspired by work of Anderson (Anderson, 1958), who tried to reduce deontic logic to alethic modal logic. The crux of his idea was that something (φ) is forbidden if, and only if, violation of the constraint φ has some undesirable effect. In (Meyer, 1988) this idea is applied explicitly to actions: an action is forbidden if, and only if, performing it leads to an undesirable effect. This is easily formalized in dynamic logic.

5.1 THE LOGIC *PDEL*

In order to reason about the deontics of actions we take the following language, where we assume a special propositional constant $\mathbf{V} \in \mathcal{P}$, denoting a state of ‘violation’ of a deontic constraint.

Definition 7 The logical language \mathcal{L}_{DeL} and action language \mathcal{L}_{ACT_0} are given as the least sets closed under the clauses:

1. $\mathcal{P} \subseteq \mathcal{L}_{DeL}$

2. $\mathcal{A} \subseteq \mathcal{L}_{ACT0}$
3. $\varphi, \psi \in \mathcal{L}_{DeL}$ implies $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \varphi \rightarrow \psi, \varphi \leftrightarrow \psi \in \mathcal{L}_{DeL}$
4. $\varphi \in \mathcal{L}_{DeL}, \alpha \in \mathcal{L}_{ACT0}$ implies $[\alpha]\varphi, \langle\alpha\rangle\varphi \in \mathcal{L}_{DeL}$
5. $\alpha, \beta \in \mathcal{L}_{ACT0}$ implies $\alpha; \beta, \alpha + \beta, \alpha \& \beta, \bar{\alpha} \in \mathcal{L}_{ACT0}$

Thus in the language we use for deontic purposes we employ a slightly different set of action operators, viz. sequential composition, alternative composition (choice), parallel composition and action negation(!). The latter is a little non-standard in dynamic logic. It expresses that the action is *not* performed. This construct will enable us to express the obligation of an action below. Note that the non-performance of an action should not be regarded in direct association with an *attempt* of performing it and failing. It just says that the action was not executed. This is understood most easily when one views actions to be executed in 'steps'.⁶ A step contains a number of actions that are executed concurrently. If a particular action is not a member of this 'step', it is not performed in this (execution) step. In a computational context this would mean that the computation step at hand does not involve the execution of this particular action.

Models for the language \mathcal{L}_{DeL} are like those for \mathcal{L}_{DL} , but of course, one has now to cater for the operators & and -. Since we would like the interpretations of the operators to satisfy the rules of a Boolean algebra, this is (surprisingly) rather involved (due to the presence of the sequential composition ','), so that we do not give its full definition here. (It involves the notion of a 'step' as described informally above.) Here it suffices that & and - resemble the intersection and complement operator, respectively, on the accessibility relations (just as + resembles the union operator) (cf. (Meyer, 1988; Dignum et al., 1996a)).⁷

We can now define the deontic operators as abbreviations as follows:

Definition 8 ■ $F\alpha \equiv [\alpha]V$, i.e., the action α is forbidden if, and only if, performing α leads to a state of violation;

- $P\alpha \equiv \neg F\alpha$, i.e., the action α is permitted if, and only if, it is not forbidden;
- $Obl\alpha \equiv [\bar{\alpha}]V$, i.e., an action α is obligated if, and only if, not-doing it leads to a violation.

Although the basic ideas behind these definitions are very simple indeed, it leads already to some interesting deontic properties, the proofs of which can be found in (Meyer, 1988):

⁶Of course, for this to make sense, actions should be 'named' entities, which in our setting is the case since actions are elements of an action language.

⁷With respect to the negation of sequential composition we have that in the semantics it holds that $\alpha; \beta = \bar{\alpha} + (\alpha; \beta)$.

Proposition 1 1. $(\mathbf{Obl}\alpha \vee \mathbf{Obl}\beta) \rightarrow \mathbf{Obl}(\alpha + \beta)$

2. $\mathbf{P}(\alpha + \beta) \leftrightarrow (\mathbf{P}\alpha \vee \mathbf{P}\beta)$
3. $\mathbf{F}(\alpha + \beta) \leftrightarrow (\mathbf{F}\alpha \wedge \mathbf{F}\beta)$
4. $\mathbf{Obl}(\alpha \& \beta) \leftrightarrow (\mathbf{Obl}\alpha \wedge \mathbf{Obl}\beta)$
5. $\mathbf{P}(\alpha \& \beta) \rightarrow (\mathbf{P}\alpha \wedge \mathbf{P}\beta)$
6. $(\mathbf{F}\alpha \vee \mathbf{F}\beta) \rightarrow \mathbf{F}(\alpha \& \beta)$
7. $\mathbf{Obl}(\alpha; \beta) \leftrightarrow (\mathbf{Obl}\alpha \wedge [\alpha]\mathbf{Obl}\beta)$
8. $\mathbf{P}(\alpha; \beta) \leftrightarrow \langle \alpha \rangle \mathbf{P}\beta$
9. $\mathbf{F}(\alpha; \beta) \leftrightarrow [\alpha]\mathbf{F}\beta$

Although these properties are rather intuitive, one might consider alternative approaches with (slightly) different ones. For example, in legal reasoning it is not always the case that everything that is not forbidden is automatically permitted, so that one might also consider a class of actions that is neither (explicitly) forbidden nor (explicitly) permitted (cf. e.g. (Soeteman, 1989)). Sometimes also a property like $(\mathbf{Obl}\alpha \vee \mathbf{Obl}\beta) \rightarrow \mathbf{Obl}(\alpha + \beta)$ and, hence, $\mathbf{Obl}\alpha \rightarrow \mathbf{Obl}(\alpha + \beta)$ is considered undesirable. This property is called Ross's paradox, and it is counterintuitive if one reads the choice operator in such a way that the agent is free to choose its alternatives itself: "if one ought to mail the letter, then one ought to mail the letter or burn it" sounds counterintuitive in this reading. Deontic logic traditionally has a history of such paradoxes.⁸ Another paradox was discovered by Van der Meyden (van der Meyden 1990) concerning permission in our approach, where only the 'end' situation is decisive as to whether the action is permitted: $\mathbf{P}(\alpha; \beta) \leftrightarrow \langle \alpha \rangle \mathbf{P}\beta$ is equivalent with $\mathbf{P}(\alpha; \beta) \leftrightarrow \langle \alpha \rangle (\langle \beta \rangle \neg \mathbf{V})$, which has as a consequence that doing α resulting in a violation state while then doing β resolving the violation is permitted according to our definitions. So this is a kind of "the end justifies the means" approach, and while this might be acceptable for some applications, it is obviously not the case for all uses or readings of permission, so that one should then use a stronger notion. In fact several of these alternatives have been proposed. We ourselves have also considered alternatives to deal with these so-called paradoxes (e.g. (Dignum et al., 1996a)), but most of them can be viewed harmless once one realizes what exactly the meanings of these are, and one is then able to 'fine-tune' these notions to one's needs.

Deontic logic can be employed for the representation of knowledge in domains where obligations, permissions and prohibitions occur directly, such as the legal reasoning domain (Royakkers, 1998), but its application goes beyond that. It has also proved very useful in the description of systems where it is important to

⁸To be fair, other modal logics have some of this trouble as well; for instance epistemic logic suffers from the so-called logical omniscience problem(s), which is very much akin conceptually and technically to the deontic paradoxes (cf. (Meyer and van der Hoek, 1995)).

distinguish ideal from actual behavior. Ideal behavior is then specified by using deontic operators which can then be well distinguished from actual behavior. So here we see that there are more or less implicit (ideal) constraints that can be (actually) violated by the system. Examples of such systems include fault-tolerant software systems (Coenen, 1993), but also information systems such as databases where there may be so-called soft constraints that every system state ought to/should satisfy, but again might be violated (Meyer et al., 1998). It has recently been recognized that being able to represent such a violation (by e.g. the use of deontic operators) is very helpful to specify for instance a way to recover from such a violation. Soft constraints typically occur in systems where not all actors (components) in the system are under complete control and have a certain *autonomy* so that they can choose themselves to violate these constraints. A commonsense example would be an automated library system where nevertheless the people borrowing books may choose to return their books too late according to the library's regulations. Examples of applications of the use of deontic logic for system specification can be found in (Wieringa and Meyer, 1993). A more elaborate example of the use of dynamic deontic logic is given in (Meyer et al., 1998), where we discuss the specification of procedures for overseas trade.

6 TYPICAL EFFECTS OF COMMONSENSE ACTIONS AND THE FRAME PROBLEM

In AI there has been considerable effort to give an adequate treatment of reasoning about actions in a commonsense context. It has appeared to be a very hard problem to describe the effects and particularly the *non-effects* of actions. This problem has become known as the *frame problem*. Related problems include the specification of the necessary preconditions for an action to be performed successfully (or intendedly), also known as the *qualification problem*, and the precise determination of the derived effects of an action, the *ramification problem* (cf. (Sandewall and Shoham, 1994)). Although there has been a considerable amount of research on this problem and definitely some progress have been made, I believe that in general the frame problem and its related problems remain highly problematic. The problem is the space of possible (non-) effects and preconditions/qualifications and ramifications is potentially infinite, or at least astronomical in practical cases, so that it is virtually impossible to give such a precise specification in those cases, and neither do we really want to. What is searched for is a 'convenient' method to deal with this problem, which triggered a lot of research in nonmonotonic reasoning (such as default reasoning) with the idea in mind that 'by default', unless explicitly specified or deduced otherwise, aspects/features of the world remain the same. (This is sometimes called '*inertia*', but one has to bear in mind that this has nothing to do with physics, but rather with the reasoner's mental laziness to reckon with all conceivable changes/preconditions/ramifications of the world due to the performance of some action! But this laziness on the part of the reasoner is not to be blamed on him/her, since s/he has to act in the world in a real-time fashion.)

6.1 PREFERENTIAL ACTION SEMANTICS

In (Meyer and Doherty, 1999) an approach to the specification of actions dealing with some of the issues mentioned above is proposed which is based on the use of dynamic logic (inspired by work reported in (Łukaszewicz and Madalińska-Bugaj, E., 1995) using the different but related framework of Dijkstra's weakest precondition calculus as well as the Features and Fluents approach as taken in (Sandewall, 1994)). Essentially this approach consists of specifying, per action, which aspects of the world, in this area often called *features*, are definitely *set* to some value, which are *framed*, that is subject to *inertia*, and which fluents are considered to be truly *variable*, expressing that their values may be nondeterministically change during the performance of the action at hand. In the literature on action changes and the frame problem it has proved to be also useful to sometimes '*release*' features temporarily from being framed (i.e. being subject to inertia) so that they become temporarily variable (cf., for example, (Sandewall, 1994; Kartha and Lifschitz, 1994)).

The action language that we consider here is the following variant of the basic one. Besides a set \mathcal{A} of atomic actions we assume a set \mathcal{F} of *feature* variables.

Definition 9 The action language \mathcal{L}_{PA} is given as the least set closed under the clauses:

1. $\mathcal{A} \subseteq \mathcal{L}_{PA}$
2. $\alpha, \beta \in \mathcal{L}_{PA}$ implies $\alpha \oplus \beta, \alpha + \beta, \alpha \parallel \beta, \text{ if } b \text{ then } \alpha \text{ else } \beta \in \mathcal{L}_{PA}$, where b is a boolean test on feature variables
3. $\alpha \in \mathcal{L}_{PA}$ implies $\alpha \# \in \mathcal{L}_{PA}$

Thus here we have atomic actions, and compositions of these by means of the operators \oplus (restricted choice), $+$ (liberal choice), \parallel (parallel or simultaneous composition) and a conditional composition operator. The difference between the restricted choice and liberal choice operator is that in the latter a release mechanism is applied to the operands in a joint fashion (to be explained later on), resulting in a greater number of outcomes, also with respect to the preferred behavior, whereas this is not the case for the restricted choice operator. Note the lack of a sequential composition in this language.⁹

To render semantics to these action expressions, we need to fix some things. We endow an atomic action $a \in \mathcal{A}$ with a 'signature' $(\text{set}_a, \text{frame}_a, \text{release}_a)$, where $\text{set}_a, \text{frame}_a, \text{release}_a \subseteq \mathcal{F}$, and $\text{release}_a \subseteq \text{frame}_a$, describing the status of the feature variables w.r.t. this action: set_a is the set of feature variables that are *set* by a , frame_a is the set of variables that are *framed*

⁹This operator may be added, but as it is not clear what it means to have the preferred behavior of a sequentially composed action, one has to take care that the $\#$ operator cannot be applied to such an action. Thus a layered language is needed which we will not define here. Here we only note that in the logical language a property of a sequentially composed scenario $\alpha; \beta$, like $[\alpha; \beta]\varphi$, can be expressed by $[\alpha][\beta]\varphi$.

while performing a , of which the variables in $release_a$ are *released* in the sense we mentioned above. For convenience we also use the abbreviations $inert_a = frame_a \setminus release_a$ for the feature variables that are subject to inertia while performing action a and $var_a = \mathcal{F} \setminus (set_a \cup frame_a)$ for the feature variables that are ‘truly variable’, i.e. are known to be subject to change of their values in a nondeterministic way.

In this context we define a state s as a function: $\mathcal{F} \rightarrow \mathcal{V}$, where \mathcal{V} is a set of feature values (in many examples this set may be taken to be the booleans). The set of states is denoted Σ . The state $s\{d/x\}$, with $s \in \Sigma, d \in \mathcal{V}, x \in \mathcal{F}$, is defined as the state s' satisfying $s'(x) = d$, and $s'(y) = s(y)$ for $y \neq x$. For $X \subseteq \mathcal{F}$ and $s \in \Sigma$, we define $vary_X(s) = \{s' \mid s' = s\{v_1/x_1, v_2/x_2, \dots, v_n/x_n\}$ for any subsets $\{v_1, v_2, \dots, v_n\} \subseteq \mathcal{V}, \{x_1, x_2, \dots, x_n\} \subseteq X\}$. Furthermore, for $S \subseteq \Sigma$, we put $vary_X(S) = \bigcup_{s \in S} vary_X(s)$. Note that *vary* has the property that $vary_X(vary_Y(S)) = vary_{X \cup Y}(S)$.

Now we are ready for the formal semantics of our actions expressions:

For atomic actions $a \in \mathcal{A}$ we stipulate an accessibility relation $r(a) \subseteq \Sigma \times \Sigma$ that yields the behavior of a w.r.t. the variables in set_a . Thus, $r(a)(s, s')$ captures the setting of the variables in set_a by means of a .

To cater for the other variables, we include in the semantics of an action α (given an input state) a number of items, viz.

1. the set of all states that *possibly* may result as a consequence of performing the action α , when allowing all the variables apart from the *set* ones in set_α to vary arbitrarily,
2. the set of all states that are preferred/expected as a consequence of performing the action α , taking inertia of the *inert* variables in $inert_\alpha$ into account, and
3. a triple $(set_\alpha, frame_\alpha, release_\alpha)$ that records the exact resulting status of the variables during performance of α (for future reference, see below).

For atomic actions this is given by means of the semantic function []:

$$[a](s) = \{(\bigcup_{s' : r(a)(s, s')} vary_{var_a \cup frame_a}(s'),$$

$$\bigcup_{s' : r(a)(s, s')} vary_{var_a \cup release_a}(s'), (set_a, frame_a, release_a)\}\}$$

For compound actions α we also need to establish the status of the variables. We do this again by sets set_α , $frame_\alpha$ and $release_\alpha$, and use abbreviations $inert_\alpha = frame_\alpha \setminus release_\alpha$ and $var_\alpha = \mathcal{F} \setminus (set_\alpha \cup frame_\alpha)$. The semantics of the conditional composition operator is as usual. A restricted choice of two actions results in either doing the first action or doing the second one (with the respective associated outcomes), where—with respect to preferred behavior—*inertia* is applied to the chosen action separately. A liberal choice also results in performing the one or the other action, but now with respect to preferred behavior the variables set by the actions in isolation (besides the released ones

in isolation) are (restrictly) released for the liberal choice as a whole, so that in absence of inertia for these variables an outcome comprising a joint effect resulting from both actions at the same time may occur as well. In fact, it is this very joint effect that is rendered by the parallel composition of the actions. The reader is referred to (Meyer and Doherty, 1999) for a commonsense example.

To shape the above ideas in a formal way we define:

$$[\alpha \Delta \beta](s) = [\alpha](s) \Delta [\beta](s)$$

for $\Delta = \oplus, +, ||$ and

$$[\alpha \#](s) = \#([\alpha](s))$$

where $Z_1 \Delta Z_2 = \bigcup_{z_1 \in Z_1, z_2 \in Z_2} z_1 \Delta z_2$ for $\Delta = \oplus, +, ||$ and $\#(Z) = \bigcup_{z \in Z} \#(z)$, and

- regarding the operator \oplus we define:

$$(S_1, S'_1, (set_1, frame_1, release_1)) \oplus (S_2, S'_2, (set_2, frame_2, release_2)) = \{(S_1, S'_1, (set_1, frame_1, release_1)), (S_2, S'_2, (set_2, frame_2, release_2))\}$$

- regarding the operator $+$ we have:

$$(S_1, S'_1, (set_1, frame_1, release_1)) + (S_2, S'_2, (set_2, frame_2, release_2)) = \{(S_3, S'_3, (set_1 \cup set_2, frame_1 \cup frame_2, release_1 \cup release_2)), (S_4, S'_4, (set_1 \cup set_2, frame_3, release_3))\}$$

where $S_3 = vary_{(set_2 \cup frame_2) \setminus set_1}(S_1)$,

$S'_3 = vary_{(set_2 \cup release_2) \setminus set_1}(S'_1)$,

$S_4 = vary_{(set_1 \cup frame_1) \setminus set_2}(S_2)$,

$S'_4 = vary_{(set_1 \cup release_1) \setminus set_2}(S'_2)$,

$frame_3 = (frame_1 \cup frame_2 \cup set_2) \setminus set_1$,

$release_3 = (release_1 \cup release_2 \cup set_2) \setminus set_1$,

$frame_4 = (frame_1 \cup frame_2 \cup set_1) \setminus set_2$,

$release_4 = (release_1 \cup release_2 \cup set_1) \setminus set_2$

- regarding the operator $||$ we have:

$$(S_1, S'_1, (set_1, frame_1, release_1)) || (S_2, S'_2, (set_2, frame_2, release_2)) = \{(S_3 \cap S_4, S'_3 \cap S'_4, (set_3, frame_3, release_3))\}$$

where S_3, S'_3, S_4, S'_4 are as in the previous case, and $set_3 = set_1 \cup set_2$, $frame_3 = (frame_1 \cup frame_2) \setminus (set_1 \cup set_2)$, $release_3 = (release_1 \cup release_2) \setminus (set_1 \cup set_2)$

- and finally regarding the preference operator $\#$ it holds that:

$$\#(S, S', (set, frame, release)) = (S', S', (set \cup inert, frame \setminus release, release))$$

where $inert = frame \setminus release$.

Note that $[(\alpha_\#)_\#] = [\alpha_\#]$. If, for $[\alpha](s) = \{(S_1, S'_1, \dots), (S_2, S'_2, \dots), \dots\}$, we use the notation $[\alpha]_1(s) = \bigcup_i S_i$ and $[\alpha]_2(s) = \bigcup_i S'_i$, we may define the interpretation of formulas $[\alpha]\varphi$ and $[\alpha_\#]\varphi$ as follows:

- $s \models [\alpha]\varphi$ if, and only if, $t \models \varphi$ for all $t \in [\alpha]_1(s)$ ly if, $t \models \varphi$ for all $t \in [\alpha]_2(s)$

Typical validities in this framework are:

- $\models [\alpha]\varphi \rightarrow [\alpha_\#]\varphi$
- $\models [\alpha]\varphi \rightarrow [\alpha \parallel \beta]\varphi$
- $\models [(\alpha + \beta)_\#]\varphi \rightarrow [(\alpha \oplus \beta)_\#]\varphi$

the first expressing that the expected states after execution of an action constitute a subset of the possible states after executing that action; the second that concerning all possible successor states those of α are a superset of those of the parallel execution $\alpha \parallel \beta$; and the third expressing that the expected states after the execution of the restricted choice between two actions is a subset of those resulting after a liberal choice.

Typical *non*-validities are:

- $\not\models [\alpha_\#]\varphi \rightarrow [(\alpha \parallel \beta)_\#]\varphi$
- $\not\models [(\alpha \oplus \beta)_\#]\varphi \rightarrow [(\alpha + \beta)_\#]\varphi$

the first stating that in general the set of *expected* states when executing an action is *not* a superset of the set of *expected* states when executing that action in parallel with some other action; the second that the expected states after the execution of the restricted choice between two actions is generally a *proper* subset of those resulting after a liberal choice.

Interestingly, in this approach preferred behavior in terms of applying inertia when possible is explained in terms of ‘concurrency’: if no concurrent events (or actions as performed by other agents) are known to interfere, the behavior of the action at hand in which inertia is applied to the *inert* variables is preferred. Thus, for example, the preferred behavior of a *wait* action, viewed in isolation is the same as that of the *skip* action, the interpretation of which is the identity function, while viewed in a concurrent context we may have to take into account the interference of other events. (This is different from the approach proposed in (Łukaszewicz and Madalińska-Bugaj, E., 1995) where the *wait* action is simply identified with the *skip* action.) In (Meyer and Doherty, 1999) we show how typical AI scenarios involving inertia like the infamous Yale Shooting Scenario can be treated adequately in this set-up.

7 SPECIFYING INTELLIGENT AGENTS

In the philosophical literature the term ‘agent’ is often used to refer to the (human) subject who knows, reasons or performs acts. Recently in computer science the term ‘(intelligent) agent’ has become popular to refer to intelligent pieces of software (or hardware) that perform ‘intelligent’ tasks autonomously,

that is, in a way that is not directly dependent on the user's actions or requests (Wooldridge and Jennings, 1995; Jennings et al., 1998). Typical examples of software agents (or softbots) are personal assistants to help and guide the Internet user through the ever increasing and perhaps already almost impenetrable jungle of the World Wide Web. Of course, another example of these artificial agents are robots that should be able to reason and perform intelligent tasks. In this section we will use a modal logic based on dynamic logic to describe the behavior of such intelligent agents, thus providing a theoretic framework on which the realization of these artifacts may be founded.

7.1 SINGLE AGENTS

First we will discuss the behavior and attitudes of a single agent in isolation. In the next subsection we will see what issues have to be tackled when 'societies of agents' are studied. This provides a simplified treatment of our work reported in (van Linder et al., 1995; van Linder et al., 1996; van Linder et al., 1997; Meyer and van der Hoek, 1999).

7.1.1 Agent Attitudes. Typical agent attitudes include, besides the ability of acting(!), possession and maintenance of knowledge/belief, and intentional or motivational ones, such as the possession and maintenance of desires, goals and commitments. These attitudes may be described in a logic that consists of dynamic logic, enriched with (modal) operators dealing with such notions as ability, opportunity, knowledge, belief, desires and goals. So we enrich the language with operators **A**, **O**, **K**, **B**, **D**, **G**.¹⁰ Moreover, we introduce special actions **revise** and **commit** in the language to deal with revision of knowledge/belief and the performance of commitments (which can be seen as an operator that revises the agent's internal agenda).

Definition 10 The logical language \mathcal{L}_{SA} and action language $\mathcal{L}_{ACT,SA}$ are the least extensions of the languages \mathcal{L}_{DL} and \mathcal{L}_{ACT} , respectively, that are closed under the clauses:

1. $\varphi \in \mathcal{L}_{SA}$ implies **K** φ , **B** φ , **D** φ , **G** $\varphi \in \mathcal{L}_{SA}$
2. $\varphi \in \mathcal{L}_{SA}$, $\alpha \in \mathcal{L}_{ACT,SA}$ implies **A** α , **O** $\alpha \in \mathcal{L}_{SA}$
3. $\varphi \in \mathcal{L}_{SA}$, $\alpha \in \mathcal{L}_{ACT,SA}$ implies $[\alpha]\varphi, \ll\alpha\gg\varphi \in \mathcal{L}_{SA}$
4. $\varphi \in \mathcal{L}_{SA}$ implies **revise** $\varphi \in \mathcal{L}_{ACT,SA}$
5. $\alpha \in \mathcal{L}_{ACT,SA}$ implies **commit** $\alpha \in \mathcal{L}_{ACT,SA}$

¹⁰We mention here also related work (Huang et al., 1996) in which a so-called preference-based action logic is proposed based on dynamic logic in which goals are expressed as well. The main difference between this logic and ours is that Huang et al. use a blend of dynamic logic, preference logic (where preferences are taken in the sense of Herbert Simon), (a kind of) conditional logic and Veltman's update semantics, and define goals in these terms and preferences in particular. Their work is more motivated from the perspective of social science and particularly organization theory than from our standpoint of specifying and realising an intelligent system.

To interpret this extended language we need also to extend our models:

Definition 11 A Kripke model for \mathcal{L}_{SA} is a structure \mathcal{M} of the form $(S, \pi, r, t, R_K, R_B, R_D, Agenda)$, where

- S is a non-empty set (the set of *states*);
- $\pi : S \rightarrow (\mathcal{P} \rightarrow \{\text{tt}, \text{ff}\})$ is a truth assignment function to the atoms per state;
- $r, t : \mathcal{L}_{ACT, SA} \rightarrow 2^{S \times S}$ are state transition relations per action, satisfying the properties of Definition 2, as well as some constraints regarding the *revise* and *commit* actions, to which we will return in the sequel. The r transition relation models the transition resulting from performing an action, with respect to the aspect of the agent's opportunity (provided by the environment) to perform the action, regardless of its ability to perform this action, whereas the t transition relation models the transition resulting from doing the action, with respect to the aspect of the agent's (internal) capability to perform the action, regardless whether it has the actual opportunity to do so (cf. (van der Hoek et al., 1999));
- $R_K, R_B, R_D \subseteq S \times S$, such that R_K is an equivalence relation, R_B is serial¹¹, transitive and euclidean¹², and $R_B \subseteq R_K$;
- $Agenda : S \rightarrow 2^{\mathcal{L}_{ACT}}$ is a function that yields the agent's agenda of commitments, consisting of the set of actions it is committed to, at any state.

The interpretation of the (new) formulas in \mathcal{L}_{SA} now reads:

Definition 12 Let $\mathcal{M} = (S, \pi, r, t, R_K, R_B, R_D, Agenda)$ and $s \in S$. Then:

- $\mathcal{M}, s \models K\varphi$ if, and only if, $\mathcal{M}, s' \models \varphi$ for all s' with $R_K(s, s')$;
- $\mathcal{M}, s \models B\varphi$ if, and only if, $\mathcal{M}, s' \models \varphi$ for all s' with $R_B(s, s')$;
- $\mathcal{M}, s \models D\varphi$ if, and only if, $\mathcal{M}, s' \models \varphi$ for all s' with $R_D(s, s')$;
- $\mathcal{M}, s \models \text{Com}\alpha$ if, and only if, $\alpha \in Agenda(s)$,¹³
- $\mathcal{M}, s \models [\alpha]\varphi$ if, and only if, $\mathcal{M}, s' \models \varphi$ for all s' with $t(\alpha)(s, s')$;
- $\mathcal{M}, s \models \ll \alpha \gg \varphi$ if, and only if, $\mathcal{M}, s' \models \varphi$ for some s' with $t(\alpha)(s, s')$;

These modalities serve as a basis to define further operators, such as ability, opportunity, practical possibility, can, goal, (possible) intend, as follows:

¹¹ A binary relation R is serial if for all s there exists t such that $R(s, t)$.

¹² A binary relation R is euclidean if for all s, t, u : $R(s, t)$ and $R(s, u)$ implies $R(t, u)$.

¹³ This is a simplification of our treatment in (Meyer and van der Hoek, 1999), where we also incorporated semantical implications of such an action into the notion of commitment, dealing e.g. with initial computations of the action.

- Definition 13**
- (ability) $\mathbf{A}\alpha \equiv \ll\alpha \gg tt$, i.e., an agent is able to do an action if, and only if, there is a successor state w.r.t. the t -relation;
 - (opportunity) $\mathbf{O}\alpha \equiv (\alpha)tt$, i.e., an agent has the opportunity to do an action if, and only if, there is a successor state w.r.t. the r -relation;
 - (practical possibility) $\mathbf{P}(\alpha, \varphi) \equiv \mathbf{A}\alpha \wedge \mathbf{O}\alpha \wedge (\alpha)\varphi$, i.e., an agent has the practical possibility to do an action with result φ if, and only if, it is both able and has the opportunity to do that action and the result of actually doing that action leads to a state where φ holds;
 - (can) $\mathbf{Can}(\alpha, \varphi) \equiv \mathbf{KP}(\alpha, \varphi)$, i.e., an agent can do an action with a certain result if, and only if, it knows it has the practical possibility to do so;
 - (realisability) $\Diamond\varphi \equiv \exists a_1, \dots, a_n \mathbf{P}(a_1; \dots; a_n, \varphi)$ ¹⁴, i.e., a state property φ is realisable if, and only if, there is a finite sequence of atomic actions of which the agent has the practical possibility to perform it with the result φ ;
 - (goal) $\mathbf{G}\varphi \leftrightarrow \neg\varphi \wedge \mathbf{D}\varphi \wedge \Diamond\varphi$, i.e., a goal is a formula that is not (yet) satisfied, but desired and realisable.¹⁵
 - (possible intend) $\mathbf{I}(\alpha, \varphi) \equiv \mathbf{Can}(\alpha, \varphi) \wedge \mathbf{KG}\varphi$, i.e., an agent (possibly) intends an action with a certain result if, and only if, the agent can do the action with that result and it moreover knows that this result is a goal of his.

We are now in a position to state the constraints for the `revise` and `commit` actions. Given a \mathcal{L}_{SA} -model \mathcal{M} , we define the semantics of these actions as model/state transformers:

1. $r(\text{revise}\varphi)(\mathcal{M}, s) = \text{update_belief}(\varphi, (\mathcal{M}, s))$, and likewise for $t(\text{revise}\varphi)$;
2. $r(\text{commit}\alpha)(\mathcal{M}, s) = \text{update_agenda}(\alpha, (\mathcal{M}, s))$, if $\mathcal{M}, s \models \mathbf{I}(\alpha, \varphi)$ for some φ , otherwise $r(\text{commit}\alpha)(\mathcal{M}, s) = \emptyset$ (indicating failure of the commit action), and likewise for $t(\text{commit}\alpha)$.

Here `update_belief` and `update_agenda` are functions that update the agent's belief and agenda, respectively. Their formal definitions can be found in (van Linder et al., 1995; van Linder et al., 1997) and (Meyer and van der Hoek, 1999). The `revise` operator can be used to cater for revisions due to observations

¹⁴We abuse our language here slightly, since strictly speaking we do not have quantification in our object language. See (Meyer and van der Hoek, 1999) for a proper definition.

¹⁵In fact, here we again simplify matters slightly. In (Meyer and van der Hoek, 1999) we also stipulate that a goal should be explicitly selected somehow from the desires it has, which is modeled in that paper by means of an additional modal operator. Here we leave this out for simplicity's sake.

and communication with other agents, which we will not go into further here (see (van Linder et al., 1997)). Note that the revise and commit action are ‘model-transforming’ actions rather than ‘state-transforming’ ones as is the usual interpretation of actions in dynamic logic. This should not surprise us too much, since these actions change the mental state of the agent, which in our Kripke-style representation involves (part of) the whole model rather than just one (the actual) state.

Besides the familiar properties from epistemic logic (see e.g. (Fagin et al., 1995; Meyer and van der Hoek, 1995)), typical properties of this framework, called the KARO logic, include (cf. (van Linder et al., 1995; Meyer and van der Hoek, 1999)):

1. $\models \mathbf{O}(\alpha; \beta) \leftrightarrow \langle \alpha \rangle \mathbf{O} \beta$
2. $\models \mathbf{A}(\alpha; \beta) \leftrightarrow \ll \alpha \gg \mathbf{A} \beta$
3. $\models \mathbf{Can}(\alpha; \beta, \varphi) \leftrightarrow \mathbf{Can}(\alpha, \mathbf{P}(\beta, \varphi))$
4. $\models [\text{revise} \varphi] \mathbf{B} \varphi$
5. $\models \mathbf{K} \neg \varphi \leftrightarrow [\text{revise} \varphi] \mathbf{B} \neg \varphi$
6. $\models \mathbf{K}(\varphi \leftrightarrow \psi) \rightarrow ([\text{revise} \varphi] \mathbf{B} \chi \leftrightarrow [\text{revise} \psi] \mathbf{B} \chi)$
7. $\models \mathbf{I}(\alpha, \varphi) \rightarrow \langle \text{commit} \alpha \rangle \mathbf{C} \text{om} \alpha$

In (Meyer and van der Hoek, 1999) an elaborate example of the use of the full KARO logic can be found. Here we restrict ourselves to a very simple example to give a flavor of how dynamic logic is employed to specify actions.

Example 1 We imagine a robot, equipped with wheels and a gripper, in a room. The way to the corridor is blocked by an obstacle in the doorway. The robot gets a command to go to the corridor. However, although the robot is able to do this, it lacks the opportunity due to the obstacle. So we can state that $\mathbf{A}_{\text{robot}} \text{go_thru_doorway} \wedge \neg \mathbf{O}_{\text{robot}} \text{go_thru_doorway}$ holds. However, since the robot is equipped with a gripper, it is able to remove the obstacle: $\mathbf{A}_{\text{robot}} \text{remove_obstacle}$ holds. For the sake of simplicity we stipulate that the action `remove_obstacle` is atomic. By the very fact that there is an obstacle the robot also has the opportunity to remove it, i.e. $\mathbf{O}_{\text{robot}} \text{remove_obstacle}$ is true. We assume that removing the obstacle results in the robot having the opportunity to go thru the doorway. Thus (under a further mild frame assumption regarding abilities) we have that:

$$\ll \text{do}_\text{robot}(\text{remove_obstacle}) \gg \mathbf{A}_{\text{robot}}(\text{go_thru_doorway}) \wedge \\ (\text{do}_\text{robot}(\text{remove_obstacle}))((\text{do}_\text{robot} \text{go_thru_doorway}) \text{in_the_corridor}))$$

From this it follows in our logic that:

$$\mathbf{A}_{\text{robot}}(\text{remove_obstacle}; \text{go_thru_doorway}) \wedge \\ (\text{do}_\text{robot}(\text{remove_obstacle}; \text{go_thru_doorway})) \text{in_the_corridor}$$

(The robot is able to remove the obstacle and then go thru the doorway resulting in being in the corridor.) And from this it follows immediately that:

$$\begin{aligned} & \mathbf{A}_{\text{robot}}(\text{remove_obstacle}; \text{go_thru_doorway}) \wedge \\ & \mathbf{O}_{\text{robot}}(\text{remove_obstacle}; \text{go_thru_doorway}) \wedge \\ & (\text{do}_{\text{robot}}(\text{remove_obstacle}; \text{go_thru_doorway}))_{\text{in_the_corridor}} \end{aligned}$$

that is $\mathbf{P}_{\text{robot}}(\text{remove_obstacle}; \text{go_thru_doorway}, \text{in_the_corridor})$, so that the robot has the practical possibility to remove the obstacle and then go thru the doorway in order to get in the corridor.

Under the assumption that the command to the robot yields a desire that the robot gets to the corridor, i.e. $\mathbf{D}_{\text{robot in the corridor}}$, it now also becomes a goal for the robot to get to the corridor ($\mathbf{G}_{\text{robot in the corridor}}$), since the robot is not yet in the corridor ($\neg \text{in_the_corridor}$ holds) and it is readily established that it is realisable for the robot to get to the corridor ($\Diamond_{\text{robot in the corridor}}$): there is a sequence of atomic actions, viz. $\text{remove_obstacle}; \text{go_thru_doorway}$, for which it holds that the robot has the practical possibility to perform it and that lead to the desired situation.

7.1.2 Actions with Typical Effects. As we have seen before, when we reason about actions we have to take care of an economic specification of the effects and, perhaps even more importantly, the non-effects of actions. The same problem, of course, holds for actions as performed or initiated by agents. As we have also seen, a way to do this is to consider *default* or *typical* effects of actions. In (Dunin-Keplicz and Radzikowska, 1995) an attempt has been made to incorporate this in an agent setting, more in particular in a KARO-like logic. Here we discuss the main idea(s) of their approach.

Firstly a new construct $\langle \alpha \rangle \varphi, \psi$ is added to the logical language with an intended reading that the agent has the opportunity to perform the action α and as a result of this event φ holds (always) and ψ *typically* holds.

Semantically, in the Kripke model, besides the usual accessibility relation r , a relation r° is added that deals with the *typical* results of an action; it is stipulated that $r^\circ(\alpha) \subseteq r(\alpha)$. However, contrary to what one might expect, in the approach of (Dunin-Keplicz and Radzikowska, 1995) the meaning of the construct $\langle \alpha \rangle \varphi, \psi$ is not given by means of r° . Instead the meaning of this construct is given only in terms of the relation r : $\mathcal{M}, s \models \langle \alpha \rangle \varphi, \psi$ if, and only if, $\forall s' : r(\alpha)(s, s') \Rightarrow \mathcal{M}, s' \models \varphi$ and $\exists s'' : r(\alpha)(s, s'') \wedge \mathcal{M}, s'' \not\models \neg \psi$.

The typical effects (and the semantical function r°) come into play in a rather sophisticated way. So-called scenarios are introduced, which more or less consist of a sequence of actions $\alpha = \alpha_1; \dots; \alpha_n$ together with a pre- and postcondition, γ_{pre} and γ_{post} , respectively. Next a so-called epistemic path is defined as a sequence of states in the model $\langle s_0, \dots, s_n \rangle$, such that s_0 satisfies γ_{pre} , s_n satisfies γ_{post} , and the s_i are such that $r(\alpha)(s_i, s_{i+1})$ holds.¹⁶ (In terms of dynamic logic it thus holds in s_0 that $\gamma_{\text{pre}} \wedge [\alpha]\gamma_{\text{post}}$.) The state

¹⁶In this simplified account I leave out the requirements regarding capabilities.

s_n is called an end state of the scenario. Now in this path it is counted how many of the s_i are in fact atypical, i.e. such that $\neg r^0(\alpha)(s_i, s_{i+1})$ holds. If a path has less atypical states than another, it is more preferred. This is used to define a preference-based nonmonotonic inference relation between scenarios and formulas as follows: $SC \models \beta$ (where SC is a scenario) if, and only if, for every end state s of the most preferred epistemic paths with respect to the scenario SC it holds that $\mathcal{M}, s \models \beta$. We thus see here a sophisticated mixture of dynamic logic (or rather the KARO logic) and nonmonotonic inference techniques in order to treat typical results / effects of actions.

7.1.3 Real World Agents. When considering agents that operate in the real world, such as robots, significant additional uncertainty has to be dealt with. For example, when a robot gets information from sensor devices this may contain errors, both random and systematic ones. The same holds for the other actions that agent performs in the real world. Executing a command of moving ahead for 2 meters may result in actually moving ahead for 1.9 meters. This depends on the devices (e.g. wheels) within the robot that actually achieve such a command but also on the environment (e.g. being on a slippery floor or on a slope). To treat the behavior of a real world agent properly one has to resort to means of describing such forms of uncertainty in the agent's dynamics. There has been some work on this already (Halpern, 1997; Shanahan, 1996a; Shanahan, 1996b). In (de Weerdt et al., 1999a; de Weerdt et al., 1999b) we are now trying to bring probabilistic reasoning within the framework of dynamic logic. As the work is still in a preliminary stage I restrict myself to an informal description of the main idea. As before we treat actions of the agent as model-transformers. However, these models now contain also probabilistic information. Roughly one can say that the 'state' of a real-world agent is given by a set of possible worlds with a probability distribution, indicating the probability that the agent is in that particular world. Now, for instance an observation action will result in a new 'state' described by a set of possible worlds where possibly some worlds have been eliminated and the probability distribution is adjusted according to Bayesian reasoning. This models the fact that by observation the agent learns something about the state it is in (cf. (de Weerdt et al., 1999a)). Other actions (such as a move) are given by a transition to a model in which every possible world is replaced by a set of worlds with a probability distribution on them reflecting the agent's uncertainty (for example, with respect to its position in the case of a move action) after execution of the action (de Weerdt et al., 1999b).

7.2 MULTI-AGENT SYSTEMS

In the agent systems community it is appreciated that probably the most interesting and useful applications of such systems will involve a number of agents (possibly even very many). These systems have great potential in applications such as electronic commerce. To describe, specify and realize such multi-agent systems (MAS) one needs to go beyond the individual level of agents (such as a description of their mental states/attitudes), but one has

also to pay serious attention to social attitudes/behavior of the agents in such a system. Things that naturally come to mind here are communication, coordination, negotiation, co-operation and the like. Although it is not entirely clear how much of this can be specified by purely logical means (it is, for instance, quite obvious that other theoretical concepts and techniques such as those stemming from decision and game theory are of crucial importance to obtain an adequate description), it is nevertheless interesting to investigate whether logic (and dynamic logic in particular) is also of some help in this context. Some work in this direction has been carried out already.

7.2.1 Epistemic Group Updates. For example, in (Baltag et al., 1998) belief updates within a MAS have been considered. The generalization of belief updates from the single agent case to the multi-agent one is not trivial, since one should take care of an adequate representation of whose belief is updated and whose is not! For instance, if agent i communicates to agent j some information such that another agent k is not aware of this, the knowledge/belief of agent k , and in particular his *beliefs about the beliefs of agents i and j* (!), should remain unaffected. Whereas in the single agent it is sufficient to eliminate ‘arrows’ in the Kripke model (reflecting the elimination of epistemic possibilities) this is not an adequate procedure in the multi-agent case. Baltag et al. solve the problem by *adding* arrows instead. The main idea is that, for instance in the previous example, the epistemic relations of agent j change (by eliminating epistemic possibilities), from the perspective of agent k things should remain the same, so that his accessibility relations should still point to worlds where the agents i and j have still the old epistemic possibilities. Precisely this effect can be obtained by copying the old model (for agent k) as well as adding a submodel dealing with agents i and j in isolation.

In (Gerbrandy and Groeneveld, 1997; Gerbrandy, 1999) related work is reported from a more linguistic perspective, using the framework of Veltman’s update semantics (Veltman, 1996).¹⁷ Here group updates of the form $U_G\alpha$ are considered, denoting the update of group G by means of action or ‘program’ α . These programs are either tests $\varphi?$, group updates again, or sequential or alternative compositions of programs. A sound and complete axiomatization, based on dynamic logic, is given with typical axioms such as:

- $[U_G\alpha]B_i\varphi \leftrightarrow B_i\varphi$ if $i \notin G$
- $[U_G\alpha]B_i\varphi \leftrightarrow B_i[\alpha][U_G\alpha]\varphi$ if $i \in G$

The former is called the *privacy* axiom, since it expresses that the belief of someone who is not involved in the group update will not change its beliefs. The latter is called the *Ramsey* axiom; it expresses that after a group update with action α an agent believes φ just in case s/he already believed that after

¹⁷Here we must also mention (Lomuscio, 1999), where also similar updates on Kripke models are investigated; however, here these updates are not included in the logical language itself but only considered in the meta-language.

executing α an update with $U_G\alpha$ could only result in a world where φ would be true. (This is related to the so-called Ramsey test in philosophical logic.)

7.2.2 Communication and Speech Acts. Another example of the use of (dynamic) logic for describing MAS is that pertaining to communication. Dignum and Van Linder (Dignum and van Linder, 1997) have attempted to deal with agent communication by means of dynamic logic. The idea here is to view communication between agents as *speech acts*, that is to say actions by means of speech, so that it becomes amenable to a logic for reasoning about actions such as dynamic logic. In fact they use a sophisticated mixture of the KARO logic from the previous subsection and (a refinement of) dynamic deontic logic from Section 5, since in their set-up some speech acts may give rise to obligations! (For example, they have an operator \mathbf{Obl}_{ij} such that $\mathbf{Obl}_{ij}\alpha$ expresses that agent i has an obligation to agent j to perform action α .) Furthermore, of course, in a multi-agent setting it must be indicated who is performing the action. We do this by employing the notation $do_i(\alpha)$, denoting that agent i is performing the action α . They consider the following speech acts: commitments, directions, declarations and assertions, formalized by the actions:

1. $COMMIT(i, j, \alpha)$ denoting the act of commitment of agent i to agent j to perform the action α .
2. $DIR(i, j, \alpha)$ denoting the act of agent i directing agent j to perform action α .
3. $DECL(i, \varphi)$ denoting the act of declaration of i that φ holds.
4. $ASSN(i, j, \varphi)$ denoting the act of agent i asserting to agent j that φ holds.

Without giving details of the involved semantic framework, we mention that these actions are given a model-transforming interpretation again (like the revise and commit actions for single agents, above). Suffice here to say that

1. $COMMIT(i, j, \alpha)$ changes the model in such a way that afterwards there exists an obligation of i to j to perform α , i.e. \mathbf{Obl}_{ij} holds.
2. $DIR(i, j, \alpha)$ changes the model such that afterwards j has an obligation to i to perform α . The action only succeeds if the agent i has the authority to give this directive.
3. $DECL(i, \varphi)$ changes the model such that φ holds in all resulting states, provided that φ is consistent, and the agent i has the authority to perform this declarative action.

(The action $ASSN$ is similar to the multi-agent belief update we have seen before, and I leave it out here.)

Typical formulas that are valid in this framework are thus:

- $[do_i(COMMIT(i, j, \alpha))]Obl_{ij}\alpha$
- $auth(i, DIR(i, j, \alpha)) \rightarrow [do_i(DIR(i, j, \alpha))]Obl_{ij}\alpha$
- $auth(i, DECL(i, \varphi)) \rightarrow [do_i(DECL(i, \varphi))]\varphi$

7.2.3 Social Attitudes: a Challenge for the Future. Of course, the behavior of a non-trivial multi-agent system with possibly many heterogeneous agents involves much more than their knowledge and the speech acts between them. Also more complex social attitudes are to be investigated and described. Examples include negotiations, co-ordination of behaviors, co-operation and competition, delegation, trust, coping with joint intentions and goals, but also ‘fault-tolerant’ behavior in the sense that if one agent fails to accomplish a task for some reason, another can take over or can at least take some compensatory action. Although already interesting work has been done on these issues (cf. e.g. (Dignum et al., 1996b; Castelfranchi and Falcone, 1998; Dignum and Conte, 1998)¹⁸), I believe that there is still a lot to be done here. Obviously a comprehensive and adequate treatment of these matters needs elements from game and decision theory (or theory of economics in general) as well as the theory of distributed computing. On the other hand, since it is obvious that here, too, (the description of) dynamics plays an important role, there is no *a priori* reason to doubt the use of dynamic logic based on the idea of social actions viewed as transformers of models capturing these social attitudes in this context as well. I think it is a very interesting issue to study where and how these ‘extra-logical’ elements such as the ones mentioned above may fit in.

8 DISCUSSION AND CONCLUSION

In this paper I have reviewed a number of different instances of knowledge representation involving agents, actions and changes, in which it has been proven (or at least made plausible) to be fruitful to employ dynamic logic as a basic action logic. From the instances shown I believe the wide applicability of dynamic logic is clearly demonstrated. Naturally, in each specific application domain the bare dynamic logic has to be ‘embellished’ to increase expressibility, but dynamic logic remains the ‘core’ of the resulting system.

Of course, to reason about the dynamics of (intelligent) systems one might also consider alternatives. A prominent ‘rival’ of dynamic logic is temporal logic, the logic of time (van Benthem, 1995), which has also been put to use for the purpose of the verification and specification of programs (Emerson, 1990). Although there are some similarities between dynamic and temporal logic (e.g. both are modal logics, and both can be used to describe the behavior of systems), a main difference between the two is that in dynamic logic the actions (and perhaps even the actor) involved are mentioned explicitly in the object language (this is called an *exogenous* logic in (Kozen and Tiuryn, 1990)),

¹⁸In the latter paper a formalization of goal formation/generation within a group of agents is presented in a KARO-like logic containing dynamic logic operators

while this is not the case in temporal logic (called an *endogenous* logic in (Kozen and Tiuryn, 1990)), although, of course, one may add additional predicates to express aspects of actions. As such dynamic logic has a *compositional* semantics in the sense that the meaning of formulas involving complex actions can be broken down into assertions about the atomic actions that appear in it, thus allowing proofs of such formulas by structural induction, while this is typically not the case for temporal logic. In the latter the actions (in a computerized context laid down by a program) are fixed and considered part of the structure over which the logic is interpreted. On the other hand the logic is more general: it contains general constructs such as modalities for e.g. ‘sometime in the future’ and ‘always in the future’ with no reference to actions or programs at this level. As Kozen and Tiuryn (Kozen and Tiuryn, 1990) put it: “*Thus Temporal Logic sacrifices compositionality for a less restricted and hence more generally applicable formalism*”.

In my opinion both temporal and dynamic logics have their merits, and may be convenient tools for different applications. Dynamic logic is a true logic of action with actions appearing in the logical language as first-class citizens. Therefore, if one is interested in properties of complex actions that display a neat inductive structure I believe dynamic logic is to be preferred. Put in other words, one should choose an exogenous logic such as dynamic logic (or its precursor Hoare’s Logic (Hoare, 1969)), if the very structure of actions needs to be examined logically in order to prove certain properties.

On the other hand, since dynamic logic is typically a logic to reason about input/output behavior of actions it is less suited to reason about infinite behavior (which for some systems is not anomalous and even desired, such as, for example, an operating system of a computer). Moreover, if one is primarily interested in temporal properties (properties pertaining to [real-] time), temporal logic seems the obvious choice as well. Another advantage of temporal logic is that nowadays powerful model checking methods are available to verify whether an execution model of a particular program satisfies certain properties (Emerson, 1990), while I’m not aware of similar tools for dynamic logic. Related to this is the issue whether one aims at *executable* specifications. For temporal logic there are methods available to render specifications of systems written in fragments of temporal logic executable (cf. e.g. (Dixon et al., 1998)), while for dynamic logic specifications this is still in its infancy (see e.g. (Hustadt et al., 2000)). However, at the end of the day, choosing between dynamic and temporal logic is most of all a matter of taste, and *de gustibus non est disputandum*, there is no disputing about tastes!

Acknowledgements. The author would like to express his gratitude to the various co-authors of joint papers that are cited in the present paper for their invaluable contribution. Moreover, Michael Fisher, an anonymous referee and the editor are thanked gratefully for their suggestions to improve this material.

References

- Anderson, A.R. (1958). A Reduction of Deontic Logic to Alethic Modal Logic. *Mind* 67, 1958, pp. 100–103.

- Apt K.R., Blair, H.A., and Walker A. (1988) Towards a Theory of Declarative Knowledge. In: *Foundations of Deductive Databases and Logic Programming* (J. Minker, ed.), Morgan Kaufmann, Los Altos, CA, 1988, pp. 89–148.
- Baltag, A., Moss, L.S., and Solecki, S. (1998) The Logic of Public Announcements, Common Knowledge, and Private Suspicions. In: *Theoretical Aspects of Rationality and Knowledge (Proc. TARK 1998)* (I. Gilboa, ed.), Morgan Kaufmann, San Francisco, 1998, pp. 43–56.
- van Benthem, J. (1995) Temporal Logic. In: *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 4: Epistemic and Temporal Reasoning* (D.M. Gabbay, C.J. Hogger & J.A. Robinson, eds.), Clarendon Press, Oxford, 1995, pp. 241–350.
- Castelfranchi, C. and Falcone, R. (1998) Principles of Trust for MAS: Cognitive Anatomy, Social Importance, and Quantification. In: *Proc. 3rd Int. Conf. on Multi-Agent Systems (ICMAS'98)* (Y. Demazeau, ed.), IEEE, Los Alamitos, CA, 1998, pp. 72–79.
- Coenen, J. (1993) Top-Down Development of Layered Fault-Tolerant Systems and Its Problems - a Deontic Perspective. *Annals of Mathematics and Artificial Intelligence* 9(1,2), 1993, pp. 133–150.
- Cousot, P. (1990) Methods and Logics for Proving Programs. In: J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*, Elsevier, Amsterdam, 1990, pp. 841–993.
- Dignum, F. and Conte, R., (1998) Intentional Agents and Goal Formation. In: Intelligent Agents IV (M.P. Singh, A. Rao & M.J. Wooldridge, eds.), Springer, Berlin, 1998, pp. 231–243.
- Dignum, F. and van Linder, B. (1997) Modeling Social Agents: Communication as Action. In: *Intelligent Agents III* (J.P. Müller, M.J. Wooldridge & N.R. Jennings, eds.), Springer, Berlin, 1997, pp. 205–218.
- Dignum, F.P.M., Meyer, J.-J. Ch., and Wieringa, R.J., (1996a) Free Choice and Contextually Permitted Actions. *Studia Logica* 57(1), 1996, pp. 193–220.
- Dignum, F., Meyer, J.-J. Ch., Wieringa, R.J., and Kuiper, R. (1996b) A Modal Approach to Intentions, Commitments and Obligations: Intention plus Commitment Yields Obligation. In: *Deontic Logic, Agency and Normative Systems (Proc. DEON'96)* (M.A. Brown & J. Carmo, eds.), Workshops in Computing, Springer, Berlin, 1996, pp. 80–97.
- Dixon, C., Fisher, M. and Wooldridge, M. (1998) Resolution for Temporal Logics of Knowledge, *J. of Logic and Computation* 8(3), 1998, pp. 345–372.
- Dunin-Keplicz, B. and Radzikowska, A. (1995) Epistemic Approach to Actions with Typical Effects. In: *Symbolic and Quantitative Approaches to Reasoning and Uncertainty (Proc. ECSQARU'95)* (Chr. Froideveaux & J. Kohlas, eds.), Lecture Notes in Artificial Intelligence 946, Springer, Berlin, 1995, pp. 180–188.
- Emerson, E.A. (1990) Temporal and modal logic. In: J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*, Elsevier, Amsterdam, 1990, pp. 995–1072.
- Fagin, R., Halpern, J.Y., Moses, Y., and Vardi, M. (1995) *Reasoning about Knowledge*. The MIT Press, Cambridge, MA, 1995.

- Fischer, M. and Ladner, R. (1979) Propositional Dynamic Logic of Regular Programs. *J. Comput. System Sci.* 18, 1979, pp. 194–211.
- Gerbrandy, J. (1999) Bisimulations on Planet Kripke. Ph.D. Thesis, University of Amsterdam, 1999.
- Gerbrandy, J. and Groeneveld, W. (1997) Reasoning about Information Change. *Journal of Logic, Language, and Information* 6, 1997, pp. 147–169.
- Groenendijk, J. and Stokhof, M. (1991) Dynamic Predicate Logic. *Linguistics and Philosophy* 14(1), 1991, pp. 31–100.
- Halpern, J.Y. (1997) A Logical Approach to Reasoning about Uncertainty: a Tutorial. In: *Discourse, Interaction and Communication* (X. Arrazola, K. Korta & F.J. Pelletier, eds.), Kluwer, 1998, pp. 141–155.
- Harel, D. (1984) Dynamic Logic, in: D. Gabbay & F. Guenther (eds.). *Handbook of Philosophical Logic, Vol. II*, Reidel, Dordrecht/Boston, 1984, pp. 497–604.
- Harel, D. (1979) *First-Order Dynamic Logic*. Lectures Notes in Computer Science 68, Springer, Berlin, 1979.
- Hoare, C.A.R. (1969) An Axiomatic Basis for Computer Programming. *Comm. ACM* 12, 1969, pp. 576–580.
- van der Hoek, W., Meyer, J.-J. Ch., and van Schagen, J.W. (1999) Formalizing Potential of Agents – The KARO Framework Revisited. In: *Proc. of the 7th CSLI Workshop on Logic, Language and Computation* (M. Faller, S. Kaufmann M. Pauly, eds.), CSLI Publications, Stanford, 1999, to appear.
- Huang, Z., Masuch, M., and Pólos, L. (1996) ALX, an Action Logic for Agents with Bounded Rationality. *Artificial Intelligence* 82, 1996, pp. 75–127.
- Hustadt, U., Dixon, C., Schmidt, R.A., Fisher, M., Meyer, J.-J. Ch., and van der Hoek, W. (2000) Verification within the KARO Agent Theory, submitted.
- Jennings, N.R., Sycara, K. and Wooldridge, M. (1998) A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems* 1, 1998, pp. 7–38.
- Kartha, G.N. and Lifschitz, V. (1994) Actions with Indirect Effects (Preliminary Report). In: *Proc. KR'94*, 1994, pp. 341–350.
- Kozen, D. and Tiuryn, J. (1990) Logics of Programs. In: J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics*, Elsevier, Amsterdam, 1990, pp. 789–840.
- van Linder, B., van der Hoek, W. and Meyer, J.-J. Ch. (1995) Actions that Make You Change Your Mind: Belief Revision in an Agent-Oriented Setting. In: *Knowledge and Belief in Philosophy and Artificial Intelligence* (A. Laux & H. Wansing, eds.), Akademie Verlag, Berlin, 1995, pp. 103–146.
- van Linder, B., van der Hoek, W. and Meyer, J.-J. Ch. (1996) Formalizing Motivational Attitudes of Agents: On Preferences, Goals and Commitments. In: Intelligent Agents Volume II – Agent Theories, Architectures, and Languages (M. Wooldridge, J.P. Müller & M. Tambe, eds.), Lecture Notes in Computer Sciences (Subseries LNAI) 1037, Springer-Verlag, 1996, pp. 17–32.
- van Linder, B., van der Hoek, W. and Meyer, J.-J. Ch. (1997) Seeing is Believing (And So Are Hearing and Jumping). *Journal of Logic, Language and Information* 6, 1997, pp. 33–61.

- van Linder, B., van der Hoek, W. and Meyer, J.-J. Ch. (1998) Formalizing Abilities and Opportunities of Agents. *Fundamenta Informaticae* 34(1, 2), 1998, pp. 53-101.
- Lomuscio, A. (1999) Knowledge Sharing among Ideal Agents. Ph.D. Thesis, University of Birmingham, 1999.
- Lukaszewicz, W. and Madalińska-Bugaj, E. (1995) Reasoning about Action and Change using Dijkstra's semantics for Programming Languages: Preliminary Report. In *Proc. IJCAI-95*, Montreal, 1995, pp. 1950-1955.
- Mally, E. (1926) *Grundgesetze des Sollens, Elemente der Logik des Willens*. Leuschner & Lubensky, Graz, 1926.
- Manchanda, S. and Warren, D.S. (1988) A Logic-Based Language for Database Updates. In: *Foundations of Deductive Databases and Logic Programming* (J. Minker, ed.), Morgan Kaufmann, Los Altos, CA, 1988, pp. 363-394.
- van der Meyden, R. (1990) The Dynamic Logic of Permission. *Proc. 5th IEEE Conf. on Logic in Computer Science*, Philadelphia, 1990, pp. 72-78.
- Meyer, J.-J. Ch. (1998) A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic*, vol.29, pp. 109-136, 1988.
- Meyer, J.-J. Ch. and Doherty, P. (1999) Preferential Action Semantics (Preliminary Report). In: *Formal Models of Agents* (J.-J. Ch. Meyer & P.Y. Schobbens, eds.), Springer, to appear.
- Meyer, J.-J. Ch. and van der Hoek, W. (1995) *Epistemic Logic for AI and Computer Science*. Cambridge University Press, 1995.
- Meyer, J.-J. Ch., van der Hoek, W., and van Linder, B. (1999) A Logical Approach to the Dynamics of Commitments. *AI Journal* 113, 1999, 1-40.
- Meyer, J.-J. Ch., Wieringa, R.J., and Dignum, F.P.M. (1998) The Role of Deontic Logic in the Specification of Information Systems. In: *Logics for Databases and Information Systems* (J. Chomicki & G. Saake, eds.), Kluwer, Boston/Dordrecht, 1998, pp. 71-115.
- Moore, R.C. (1985) A Formal Theory of Knowledge and Action. In: J.R. Hobbs & R.C. Moore (eds.), *Formal Theories of the Commonsense World*, Ablex, Norwood NJ, 1985, pp. 319-358.
- Pratt, V. (1976) Semantical Considerations on Floyd-Hoare Logic. In *Proc. 17th IEEE Symp. on Foundations of Computer Science*, 1976, pp. 109-121.
- Royakkers, L.M.M. (1998) *Extending Deontic Logic for the Formalization of Legal Rules*. Kluwer Academic Publishers, Dordrecht/Boston, 1998.
- Sandewall, E. (1994) *Features and Fluents: A Systematic Approach to the Representation of Knowledge about Dynamical Systems*. Oxford University Press, Oxford, 1994.
- Sandewall, E. and Shoham, Y. (1994) Nonmonotonic Temporal Reasoning. In: *Handbook of Logic in Artificial Intelligence and Logic Programming Vol.4 (Epistemic and Temporal Reasoning)* (D.M. Gabbay, C.J. Hogger & J.A. Robinson, eds.), Oxford University Press, Oxford, 1994, pp. 439-498.
- Shanahan, M.P. (1996a) Noise and the Common Sense Informatic Situation for a Mobile Robot, *Proc. AAAI'96*, 1996, pp. 1098-1103.
- Shanahan, M.P. (1996b) Robotics and the Common Sense Informatic Situation, *Proc. ECAI'96*, 1996, pp. 684-688.

- Sierra, C., Godo, L., López de Mántaras, R. and Manzano, M. (1996) Descriptive Dynamic Logic and Its Application to Reflective Architectures. *Future Generation Computer Systems* 12, 1996, pp. 157-171.
- Soeteman, A. (1989) *Logic in Law*. Kluwer Academic Publishers, Dordrecht / Boston, 1989.
- Spruit, P.A., Wieringa, R.J., and Meyer, J.-J. Ch. (1995) Axiomatization, Declarative Semantics and Operational Semantics of Passive and Active Updates in Logic Databases. *Journal of Logic and Computation* 5(1), 1995, pp. 27-70.
- Spruit, P., Wieringa, R.J., and Meyer, J.-J. Ch. (1999) Regular Database Update Logics. 1999, submitted.
- Stirling, C. (1992) Modal and Temporal Logics, in: S. Abramsky, D.M. Gabbay & T.S.E. Maibaum (eds.). *Handbook of Logic in Computer Science, Vol. II*, Clarendon Press, Oxford, 1992, pp. 477-563.
- Veltman, F. (1996) Defaults in Update Semantics. *Journal of Philosophical Logic* 25, 1996, pp. 221-261.
- de Weerdt, M., de Boer, F.S., van der Hoek, W. and Meyer, J.-J. Ch. (1999a) Imprecise Observations of Mobile Robots Specified by a Modal Logic. In: *Proc. of the fifth annual conference of the Advanced School for Computing and Imaging (ASCI'99)*, (M. Boasson, J.A. Kaandorp, J.F.M. Tonino & M.G. Vosselman, eds.), Heijen, The Netherlands, June 15-17, 1999, pp. 184-190.
- de Weerdt, M., de Boer, F.S., van der Hoek, W. and Meyer, J.-J. Ch. (1999b) Specifying Uncertainty of Mobile Robots by Means of a Modal Logic. in preparation.
- Wieringa, R.J. and Meyer, J.-J. Ch. (1993) Applications of Deontic Logic in Computer Science: A Concise Overview. In: *Deontic Logic in Computer Science: Normative System Specification*, (J.-J. Ch. Meyer & R.J. Wieringa, eds.), John Wiley & Sons Ltd., Chichester, 1993, pp. 17-40.
- von Wright, G.H. (1951) Deontic Logic. *Mind* 60, 1951, pp. 1-15.
- von Wright, G.H. (1964) A New System of Deontic Logic. *Danish Yearbook of Philosophy* 1, 1964, pp. 173-182.
- Wooldridge, M. and Jennings, N. (1995) Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review* 10(2), 1995, pp. 115-152.

IX

INDUCTIVE REASONING

Chapter 14

LOGIC-BASED MACHINE LEARNING

Stephen Muggleton and Flaviu Marginean

*Department of Computer Science
University of York
Heslington, York, YO1 5DD
United Kingdom*

Abstract The last three decades has seen the development of Computational Logic techniques within Artificial Intelligence. This has led to the development of the subject of Logic Programming (LP), which can be viewed as a key part of Logic-Based Artificial Intelligence. The subtopic of LP concerned with Machine Learning is known as “Inductive Logic Programming” (ILP), which again can be broadened to Logic-Based Machine Learning by dropping Horn clause restrictions. ILP has its roots in the ground-breaking research of Gordon Plotkin and Ehud Shapiro. This paper provides a brief survey of the state of ILP applications, theory and techniques.

Keywords: Inductive logic programming, machine learning, scientific discovery, protein prediction, learning of natural language

1 INTRODUCTION

As envisaged by John McCarthy in 1959 (McCarthy, 1959), Logic has turned out to be one of the key knowledge representations for Artificial Intelligence research. Logic, in the form of the First-Order Predicate Calculus provides a representation for knowledge which has a clear semantics, together with well-studied sound rules of inference. Interestingly, Turing (Turing, 1950) and McCarthy (McCarthy, 1959) viewed a combination of Logic and Learning as being central to the development of Artificial Intelligence research. This article is concerned with the topic of Logic-Based Machine Learning (LBML). The modern study of LBML has largely been involved with the study of the learning of logic programs, or Inductive Logic Programming (ILP) (Muggleton, 1991). Many of the foundational results of both Gordon Plotkin (Plotkin, 1971) and Ehud Shapiro (Shapiro, 1983) are still central to the study and conceptual framework of ILP. This is true despite the fact that Plotkin did not employ Horn clause restrictions.

ILP is a general form of Machine Learning which involves the construction of logic programs from examples and background knowledge. The subject has inherited its logical tradition from Logic Programming and its empirical orientation from Machine Learning. Robert Kowalski (Kowalski, 1980) famously described Logic Programming (LP) with the following equation.

$$\text{LP} = \text{Logic} + \text{Control}$$

The equation emphasizes the role of the programmer in providing sequencing control when writing Prolog programs. In a similar spirit we might describe ILP as follows.

$$\text{ILP} = \text{Logic} + \text{Statistics} + \text{Computational Control}$$

The logical part of ILP is related to the formation of hypotheses while the statistical part is related to evaluating their degree of belief. As in LP the Computational Control part is related to the sequencing of search carried out when exploring the space of hypotheses.

The structure of the paper is as follows. Section 2 introduces the key elements of ILP, provides a formal framework (Section 2.1) for the later discussion and discusses how Bayesian inference (Section 2.3) is used as a preference mechanism. Section 3 describes applications of ILP to problems related to discovery of biological function. ILP has a strong potential for being applied to Natural Language Processing (NLP). The resultant research area of Learning Language in Logic (LLL) is described in Section 4 together with some encouraging preliminary results. Conclusions concerning ongoing ILP research are given in Section 5.

2 ILP

ILP algorithms take examples E of a concept (such as a protein family) together with background knowledge B (such as a definition of molecular dynamics) and construct a hypothesis h which explains E in terms of B . For example, in the protein fold domains (Section 3.2.2), E might consist of descriptions of molecules separated into positive and negative examples of a particular fold (overall protein shape). This is exemplified in Figure 14.1 for the fold “4-helical-up-and-down-bundle”. A possible hypothesis h describing this class of proteins is shown in Figure 14.2. The hypothesis is a definite clause consisting of a *head* (*fold*(...,..)) and a *body* (the conjunction *length*(..), .. *helix*(..)). In this case “fold” is the predicate involved in the examples and hypothesis, while “length”, “position”, etc. are defined by the background knowledge. A logic program is simply a set of such definite clauses. Each of E , B and h are logic programs.

In the context of knowledge discovery a distinct advantage of ILP over black box techniques, such as neural networks, is that a hypotheses such as that shown in Figure 14.2 can, in a straightforward manner, be made readable by translating it into the following piece of English text.

The protein P has fold class “Four-helical up-and-down bundle” if it contains a long helix H1 at a secondary structure position between 1 and 3, and H1 is followed by a second helix H2.

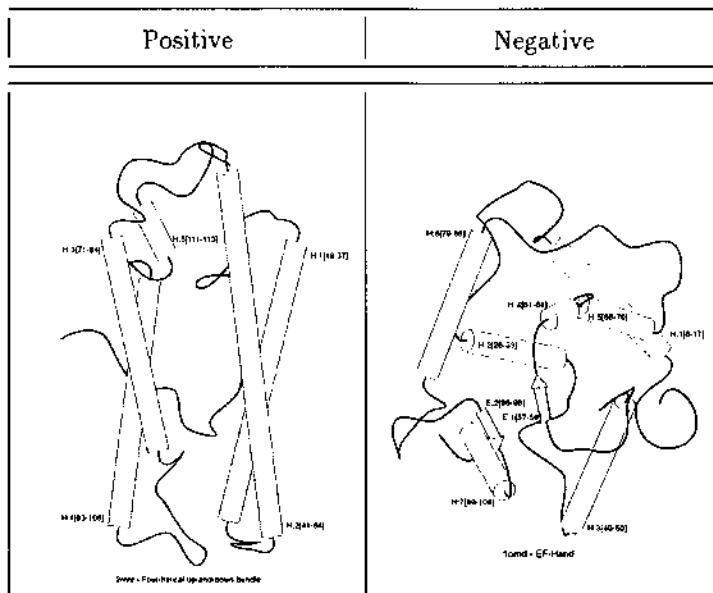


Figure 14.1 A positive and a negative example of the protein fold “4-helical-up-and-down-bundle”. 3-D arrangement of secondary structure units is shown for α -helices (cylinders) and β -sheets (arrows). Each secondary structure unit is labelled according to the index of its first and last amino acid residue.

```

fold('Four-helical up-and-down bundle',P) :-  

    helix(P,H1),  

    length(H1,hi),  

    position(P,H1,Pos),  

    interval(1 ≤ Pos ≤ 3),  

    adjacent(P,H1,H2),  

    helix(P,H2).

```

Figure 14.2 An hypothesised definite clause for 4-helical-up-and-down-bundles

Such explicit hypotheses are required for experts to be able to use them within the familiar human scientific discovery cycle of debate, criticism and refutation.

2.1 FORMAL FRAMEWORK FOR ILP

The normal framework for ILP (Muggleton and De Raedt, 1994; Nienhuys-Cheng and de Wolf, 1997) is as follows. As exemplified by the protein folds problem, described in the last sub-section, the learning system is provided with background knowledge B , positive examples E^+ and negative examples E^- and constructs an hypothesis h . B , E^+ , E^- and h are each logic programs. A logic program (Lloyd, 1987) is a set of definite clauses each having the form

$$h \leftarrow b_1, \dots, b_n$$

where h is an atom and b_1, \dots, b_n are atoms. Usually E^+ and E^- consist of ground clauses, those for E^+ being definite clauses with empty bodies and those for E^- being clauses with head 'false' and a single ground atom in the body.

In the text below the logical symbols used are: \wedge (logical and), \vee (logical or), \models (logical entailment), \square (Falsity). The conditions for construction of h are as follows.

Necessity: $B \not\models E^+$

Sufficiency: $B \wedge h \models E^+$

Weak consistency: $B \wedge h \not\models \square$

Strong consistency: $B \wedge h \wedge E^- \not\models \square$

Note that neither *sufficiency* nor *strong consistency* are required for systems that deal with noise. The four conditions above capture *all* the logical requirements of an ILP system. However, for any B and E there will generally be many h 's which satisfy these conditions. Statistical preference is often used to distinguish between these hypotheses (see Section 2.3). Both *Necessity* and *Consistency* can be checked using a theorem prover. Given that all formulae involved are definite, the theorem prover used need be nothing more than a Prolog interpreter, with some minor alterations, such as iterative deepening, to ensure logical completeness.

2.2 DERIVING ALGORITHMS FROM THE SPECIFICATION OF ILP

The *sufficiency* condition captures the notion of generalizing examples relative to background knowledge. A theorem prover cannot be directly applied to derive h from B and E^+ . However, by simple application of the Deduction Theorem the *sufficiency* condition can be rewritten as follows.

Sufficiency*: $B \wedge \overline{E^+} \models \overline{h}$

This simple alteration has a profound effect. The negation of the hypothesis can now be deductively derived from the negation of the examples together with the

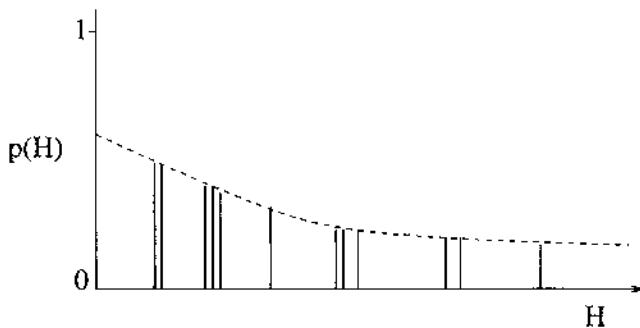


Figure 14.3 Prior over hypotheses. Hypotheses are enumerated in order of descending prior probability along the X-axis. The Y-axis is the probability of individual hypotheses. Vertical bars represent hypotheses consistent with E.

background knowledge. This is true no matter what form the examples take and what form the hypothesis takes. This approach of turning an inductive problem into one of deduction is called *inverse entailment* (Muggleton, 1995). Methods for ensuring completeness of inverse entailment have been a subject of debate recently (Yamamoto, 1997; Muggleton, 1998).

2.3 BAYESIAN FRAMEWORK

It is not sufficient to specify the ILP framework in terms of the logical relationships which must hold between E , B and h . For any given E and B there will be many (possibly infinitely many) choices for h . Thus a technique is needed for defining a preference over the various choices for h . One approach to doing so involves defining a Bayesian prior probability distribution over the learner's hypothesis space (Muggleton, 1994a). This is illustrated in Figure 14.3¹. According to Bayes' theorem the hypothesis (h_{MAP}) with maximum posterior probability in hypothesis space H is as follows.

$$\begin{aligned} h_{MAP} &= \operatorname{argmax}_{h \in H} P(h|E) \\ &= \operatorname{argmax}_{h \in H} \frac{P(E|h)P(h)}{p(E)} \\ &= \operatorname{argmax}_{h \in H} P(E|h)P(h) \end{aligned}$$

Within ILP Bayesian approaches (Muggleton, 1994b) have been used to investigate the problem of learning from positive examples only (Muggleton, 2000) and issues related to predicate invention (Khan et al., 1998) (a relational form of feature construction). Learning from positive examples and predicate

¹Note that in the case of the potentially infinite hypothesis space employed in ILP, it is not possible to have a uniform distribution which assigns non-zero prior probabilities to all hypotheses

invention are important in both natural language domains (see Section 4) and in problems involving scientific discovery (see Section 3).

3 DISCOVERY OF BIOLOGICAL FUNCTION

Understanding of a variety of metabolic processes is at the center of drug development within the pharmaceutical industry. Each new drug costs hundreds of millions of pounds to develop. The majority of this cost comes from clinical tests on efficacy and side-effects. The increasing supply of data both from the human genome project and existing drug databases is producing increasing interest in computational techniques which could reduce drug development costs by supporting automated discovery of biological functions.

Biological functions are regulated by the docking of small molecules (ligands) with sites on large molecules (proteins). Drugs, such as beta-blockers, mimic natural small molecules, such as adrenaline. Effectiveness of drugs depends on the correct shape and charge distribution of ligands. Thus beta-blockers inhibit the binding of adrenaline, and so stop over-stimulation of heart muscle in patients prone to heart attacks.

Results on scientific discovery applications of ILP are separated below between those related to small molecules (such as ligands) and those related to proteins.

3.1 SMALL MOLECULES

3.1.1 Structure-activity prediction. The majority of pharmaceutical R&D is based on finding slightly improved variants of patented active drugs. This involves laboratories of chemists synthesising and testing hundreds of compounds almost at random. The average cost of developing a single new drug is around \$300 million. In (King et al., 1992) it was shown that the ILP system Golem (Muggleton and Feng, 1992) was capable of constructing rules which accurately predict the activity of untried drugs. Rules were constructed from examples of drugs with known medicinal activity. The accuracy of the rules was found to be slightly higher than traditional statistical methods. More importantly the easily understandable rules provided insights which were directly comparable to the relevant literature concerning the binding site of dihydrofolate reductase.

3.1.2 Mutagenesis. In (King et al., 1996; Srinivasan et al., 1996) the ILP system Progol (Muggleton, 1995) was used to predict the mutagenicity of chemical compounds taken from a previous study in which linear regression had been applied. Progol's predictive accuracy was equivalent to regression on the main set of 188 compounds and significantly higher (85.7% as opposed to 66.7%) on 44 compounds which had been discarded by the previous authors as unpredictable using regression. Progol's single clause solution for the 44 compounds was judged by the domain experts to be a new structural alert for mutagenesis.

Method	Type	Accuracy	P
Ashby†	Chemist	0.77	0.29
Progol	ILP	0.72	1.00
RASH†	Biological potency analysis	0.72	0.39
TIPT†	Propositional ML	0.67	0.11
Bakale	Chemical reactivity analysis	0.63	0.09
Benigni	Expert-guided regression	0.62	0.02
DEREK	Expert system	0.57	0.02
TOPKAT	Statistical discrimination	0.54	0.03
CASE	Statistical correlation analysis	0.54	< 0.01
COMPACT	Molecular modeling	0.54	0.01
Default	Majority class	0.51	0.01

Figure 14.4 Comparative accuracies on the first round of the Predictive Toxicology Evaluation (PTE-1). Here P represents the binomial probability that Progol and the corresponding toxicity prediction method classify the same proportion of examples correctly. The “Default” method predicts all compounds to be carcinogenic. Methods marked with a † have access to short-term *in vivo* rodent tests that were unavailable to other methods. Ashby and RASH also involve some subjective evaluation to decide on structural alerts.

3.1.3 Pharmacophores. In a series of “blind tests” in collaboration with the pharmaceutical company Pfizer UK, Progol was shown (Finn et al., 1998) capable of re-discovering a 3D description of the binding sites (or pharmacophores) of ACE inhibitors (a hypertension drug) and an HIV-protease inhibitor (an anti-AIDS drug).

3.1.4 Carcinogenicity. Progol was entered into a world-wide carcinogenicity prediction competition run by the National Toxicology Program (NTP) in the USA. Progol was trained on around 300 available compounds, and made use of its earlier rules relating to mutagenicity. In the first round of the competition Progol produced the highest predictive accuracy of any automatic system entered (Srinivasan et al., 1997) (see Figure 14.4).

3.2 PROTEINS

3.2.1 Protein secondary structure prediction.. In (Muggleton et al., 1992) Golem was applied to one of the hardest open problems in molecular biology. The problem is as follows: given a sequence of amino acid residues, predict the placement of the main three dimensional sub-structures of the protein. The problem is of great interest to pharmaceutical companies involved

with drug design. For this reason, over the last 20 years many attempts have been made to apply methods ranging from statistical regression to decision tree and neural net learning to this problem. Published accuracy results for the general prediction problem have ranged between 50 and 60%, very close to majority-class prediction rates. In our investigation it was found that the ability to make use of background knowledge from molecular biology, together with the ability to describe structural relations boosted the predictivity for a restricted sub-problem to around 80% on an independently chosen test set.

3.2.2 Discovery of fold descriptions. Protein shape is usually described at various levels of abstraction. At the lower levels each family of proteins contains members with high sequence similarity. At the most abstract level folds describe proteins which have similar overall shape but are very different at the sequence level. The lack of understanding of shape determination has made protein fold prediction particularly hard. However, although there are around 300 known folds, around half of all known proteins are members of the 20 most populated folds. In (Turcotte et al., 1998) Progol was applied to discover rules governing the 20 most populated protein folds. The assignment to folds was taken from the SCOP (Structural Classification of Proteins) database (Brenner et al., 1996). Progol was used to learn rules for the five most populated folds of the four classes (alpha/alpha, beta/beta, alpha/beta and alpha+beta). The rules had an average cross-validated accuracy of $75 \pm 9\%$. The rules identified known features of folds. For instance, according to one rule the NAD binding fold where a short loop between the first beta-strand and alpha-helix is required to bind to biological cofactor molecule NAD. A questionnaire, designed by Mike Sternberg, requested named folds to be paired with fold descriptions generated by Progol. The questionnaire was sent to a selection of the world's top protein scientists. Progol successfully identified structural signatures of protein folds that were only known by the world's top expert (Dr Murzin, Cambridge).

4 LEARNING LANGUAGE IN LOGIC

The telecommunications and other industries are investing substantial effort in the development of natural language grammars and parsing systems. Applications include information extraction; database query (especially over the telephone); tools for the production of documentation; and translation of both speech and text. Many of these applications involve not just parsing, but the production of a semantic representation for a sentence.

4.1 WHY IS ILP GOOD FOR NLP?

Hand development of such grammars is very difficult, requiring expensive human expertise. It is natural to turn to machine learning for help in automatic support for grammar development. The currently dominant paradigm in grammar learning is statistically-based ((Magerman, 1995; Collins, 1996; Briscoe and Carroll, 1993; Krotov et al., 1994; Krotov et al., 1997)). This work is all, with a few recent small-scale exceptions, focussed on syntactic or

lexical properties. No treatment of semantics or contextual interpretation is possible because there are no annotated corpora available of sufficient size. The aim of statistical language modeling is, by and large, to achieve wide coverage and robustness. The necessary trade-off is that depth of analysis cannot also be achieved. Statistical parsing methods do not deliver semantic representations capable of supporting full interpretation. Traditional rule-based systems, on the other hand, achieve the necessary depth of analysis, but at the sacrifice of robustness: hand-crafted systems do not easily extend to new types of text or application.

In this paradigm disambiguation is addressed by associating statistical preferences, derived from an annotated training corpus, with particular syntactic or semantic configurations and using those numbers to rank parses. While this can be effective, it demands large annotated corpora for each new application, which are costly to produce. There is presumably an upper limit on the accuracy of these techniques, since the variety of language means that it is always possible to express sentences in a way that will not have been encountered in training material.

The alternative method for disambiguation and contextual resolution is to use an explicit domain theory which encodes in a set of logical axioms the relevant properties of the domain. While this has been done for small scale domains (e.g. (Hobbs et al., 1993)), the currently fashionable view is that it is impractical for complex domains because of the unmanageably large amount of hand-coded knowledge that would be required. However, if a large part of this domain knowledge could be acquired (semi-)automatically, this kind of practical objection could be met. From the NLP point of view the promise of ILP is that it will be able to steer a mid-course between these two alternatives of large scale, but shallow levels of analysis, and small scale, but deep and precise analyses. ILP should produce a better ratio between breadth of coverage and depth of analysis.

4.2 HOW CAN ILP BE USED FOR NLP?

In grammar learning the logical theory to be synthesised consists of grammar rules together with semantic representations. Examples are sentences from the language being learned and the background knowledge represents an existing partial grammar, perhaps supplemented with constraints on the possible forms of rules. In grammatical disambiguation and contextual interpretation the theory to be synthesised consists of a set of axioms or logical statements prescribing properties of the domain relevant to these tasks. These properties may include an ontology or type hierarchy and (perhaps default) statements about typical properties of and relations holding between the entities in the domain. The examples consist of both correct and incorrect analyses or contextual interpretations for sentences, or sentence-context pairs, where contexts can be represented as disambiguated sentences. The background knowledge may consist of a partial domain theory, or an encoding of whatever existing constraints on disambiguation or interpretation are known. The resulting theory is used as a filter on hypothesised alternative interpretations.

What is the highest point of the state with the largest area?

```
answer(P, (high-point(S,P), largest(A, (state(S), area(S,A))))).
```

What are the major cities in Kansas?

```
answer(C, (major(C), city(C), loc(C,S),
equal(S,stateid(kansas)))).
```

Figure 14.5 Form of examples

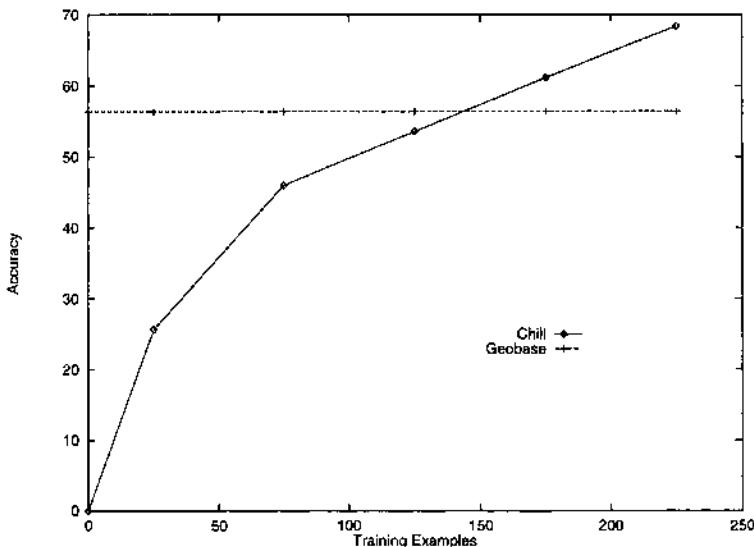


Figure 14.6 CHILL's accuracy on learning grammar and semantics

4.3 GEOGRAPHIC DATABASE QUERIES

ILP has been used for learning grammar and semantics using the CHILL system (Zelle and Mooney, 1996b). In this case, background knowledge and examples were taken from an existing database of US geographical facts. Each example consisted of a sentence paired with its semantics as shown in Figure 14.5 (figure taken from (Mooney, 1997)).

The data was gathered by asking subjects to generate appropriate questions. Each question was then paired with appropriate logical queries to give 250 examples. Figure 14.6 (taken from (Mooney, 1997)) shows CHILL's accuracy on progressively larger training sets averaged over 10 trials. The line labelled "Geobase" shows the accuracy of an existing commercially-developed hand-coded system for the same domain. CHILL outperforms the existing system when trained on 175 or more examples.

4.4 MORPHOLOGY

Mooney and Califf (Mooney and Califf, 1995) have applied ILP to learning the past tense of English verbs. Learning of English past-tense has become

Examples	Hypotheses
$\text{past}([\text{w}, \text{o}, \text{r}, \text{r}, \text{y}], [\text{w}, \text{o}, \text{r}, \text{i}, \text{e}, \text{d}]).$ $\text{past}([\text{w}, \text{h}, \text{i}, \text{z}], [\text{w}, \text{h}, \text{i}, \text{z}, \text{z}, \text{e}, \text{d}]).$ $\text{past}([\text{g}, \text{r}, \text{i}, \text{n}, \text{d}], [\text{g}, \text{r}, \text{o}, \text{u}, \text{n}, \text{d}]).$	$\text{past}(A, B) \quad :- \quad \text{split}(A, C, [r, r, y]),$ $\text{split}(B, C, [r, r, i, e, d]).$

Figure 14.7 Form of examples and hypotheses for past tense domain

a benchmark problem in the computational modeling of human language acquisition (Rumelhart and McClelland, 1986; Ling, 1994). In (Mooney and Califf, 1995) it was shown that a particular ILP system, FOIDL, could learn this transformation more effectively than previous neural-network and decision-tree methods. FOIDL's first-order default rule style representation was demonstrated by the authors as producing a predictive accuracy advantage in this domain.

However, more recently Muggleton and Bain (Muggleton and Bain, 1999) have shown that ILP prediction techniques based on Analogical Prediction (AP) produce even higher accuracies on the same data. AP is a half-way house between instance-based learning and induction. Thus AP logical hypotheses are generated on the fly for each instance to be predicted. The form of examples and hypotheses is shown in Figure 14.7. A comparison of learning curves for various systems is shown in Figure 14.8. The horizontal line labelled "Default rules" represents the following simple Prolog program which adds a 'd' to verbs ending in 'e' and otherwise adds 'ed'.

```
past(A,B) :- split(A,B,[e]), split(B,A,[d]), !.  
past(A,B) :- split(B,A,[e,d]).
```

The differences between AP and all other systems are significant at the 0.0001 level with 250 and 500 examples.

4.5 WHY IS NLP GOOD FOR ILP?

From the ILP point of view NLP has recently been recognised as a challenging application area. Some successes have been achieved in using ILP to learn grammar and semantics ((Zelle and Mooney, 1996b; Zelle and Mooney, 1993; Zelle and Mooney, 1996a; Cussens et al., 1997)). The existence within NLP problems of hierarchically defined, structured data with large amounts of relevant logically defined background knowledge provides a perfect testbed for stretching ILP technology in a way that would be beneficial in other application areas (Bratko and Muggleton, 1995; Sternberg et al., 1994; King et al., 1996; Srinivasan et al., 1996; Finn et al., 1998). The York system Progol (Muggleton, 1995) is arguably the most general purpose and widely applied ILP system. Most ILP systems concentrate on the issue of learning a single (usually non-recursive) concept and assume a set of completely and correctly defined background predicates. By contrast NLP applications need techniques

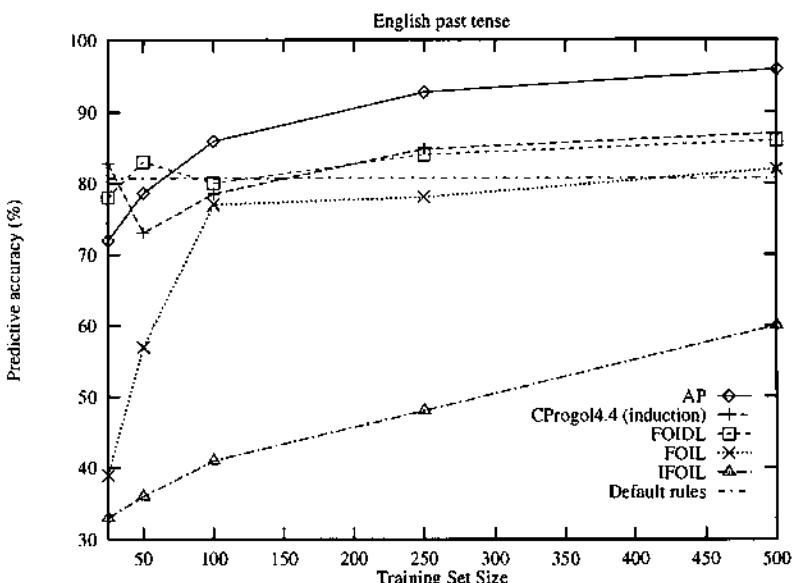


Figure 14.8 Learning curves for alphabetic English past tense. Comparisons were made between AP, CProgol4.4, FOIDL, FOIL, IFOIL and a hand-coded set of default rules. Results were averaged over 10 random chosen training sets of sizes 25, 50, 100, 250, 500 with accuracies measured over test sets of size 500.

to deal with simultaneous completion and correction of a set of related (often recursively defined) predicates. It is also expected to be necessary to implement new techniques for automatic augmentation of the set of background predicates, (Khan et al., 1998), in order to handle incompleteness of available vocabulary.

5 CONCLUSION

In his presentation of ILP biological discovery results to the Royal Society (Sternberg et al., 1994) Sternberg emphasised the aspect of joint human-computer collaboration in scientific discoveries. Science is an activity of human societies. It is the author's belief that computer-based scientific discovery must support strong integration into the existing social environment of human scientific communities. The discovered knowledge must add to and build on existing science. The ability to incorporate background knowledge and re-use learned knowledge together with the comprehensibility of the hypotheses, have marked out ILP as a particularly effective approach for scientific knowledge discovery.

In the natural language area ILP has not only been shown to have higher accuracies than various other ML approaches in learning the past tense of English (see Section 4.4) but also shown to be capable of learning accurate grammars which translate sentences into deductive database queries (Zelle and Mooney, 1996b) (see Section 4.3). In both cases, follow up studies (Thompson et al., 1997; Džeroski and Erjavec, 1997) have shown that these ILP approaches

to natural language problems extend with relative ease to various languages other than English.

The area of Learning Language in Logic (LLL) is producing a number of challenges to existing ILP theory and implementations. In particular, language applications of ILP require revision and extension of a hierarchically defined set of predicates in which the examples are typically only provided for predicates at the top of the hierarchy. New predicates often need to be invented, and complex recursion is usually involved. Similarly the term structure of semantic objects is far more complex than in other applications of ILP. Advances in ILP theory and implementation related to the challenges of LLL are already producing beneficial advances in other sequence-oriented applications of ILP. In addition LLL is starting to develop its own character as a sub-discipline of AI involving the confluence of computational linguistics, machine learning and logic programming.

Acknowledgements

Thanks are due to the funders and researchers who helped develop the technology and applications of ILP on the Esprit projects ECOLES (1989-1992), ILP I (1992-1995), ILPnet (1992-1995), ILP II (1996-1999), ILPnet2 (1998-2001), ALADIN (1998-2001) and the EPSRC Rule-based system project (1990-1993), Experimental Application and Developments of ILP (1993-1996), Distribution-based Machine Learning (1995-1998), Closed Loop Machine Learning (1992-2002). The author would also like to acknowledge the personal support he received in carrying out research into ILP under an SERC Post-doctoral Fellowship (1990-1992), an EPSRC Advanced Research Fellowship (1993-1998) and Research Fellowship from Wolfson College (1993-1997). Warm thanks are offered to the author's wife Thirza and daughter Clare for their continuous cheerful support.

References

- Bratko, I. and Muggleton, S. (1995). Applications of inductive logic programming. *Communications of the ACM*, 38(11):65-70.
- Brenner, S., Chothia, C., Hubbard, T., and Murzin, A. (1996). Understanding protein structure: using scop for fold interpretation. *Methods in Enzymology*, 266:635-643.
- Briscoe, T. and Carroll, J. (1993). Generalized probabilistic lr parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25-59.
- Collins, M. (1996). A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, pages 184-191, Santa Cruz, California, USA.
- Cussens, J., Page, D., Muggleton, S., and Srinivasan, A. (1997). Using Inductive Logic Programming for Natural Logic Processing. In Daelemans, W., Weijters, T., and van der Bosch, A., editors, *ECML'97 - Workshop Notes*

- on Empirical Learning of Natural Language Tasks*, pages 25–34, Prague. University of Economics. Invited keynote paper.
- Džeroski, S. and Erjavec, T. (1997). Induction of Slovene nominal paradigms. In Lavrač, N. and Džeroski, S., editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, pages 141–148. LNAI 1297, Springer Verlag.
- Finn, P., Muggleton, S., Page, D., and Srinivasan, A. (1998). Pharmacophore discovery using the inductive logic programming system Progol. *Machine Learning*, 30:241–271.
- Hobbs, J. R., Stickel, M. E., Appelt, D. E., and Martin, P. (1993). Interpretation as abduction. *Artificial Intelligence*, 63:69–142.
- Khan, K., Muggleton, S., and Parson, R. (1998). Repeat learning using predicate invention. In Page, C., editor, *Proc. of the 8th International Workshop on Inductive Logic Programming (ILP-98)*, LNAI 1446, pages 165–174, Berlin. Springer-Verlag.
- King, R., Muggleton, S., Lewis, R., and Sternberg, M. (1992). Drug design by machine learning: The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences*, 89(23):11322–11326.
- King, R., Muggleton, S., Srinivasan, A., and Sternberg, M. (1996). Structure-activity relationships derived by machine learning: the use of atoms and their bond connectives to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences*, 93:438–442.
- Kowalski, R. (1980). *Logic for Problem Solving*. North Holland.
- Kroto, A., Gaizauskas, R., Hepple, M., and Wilks, Y. (1997). Compacting the Penn Treebank Grammar. *Proceedings of the COLING-ACL'98 Joint Conference*, pages 699–703, Association for Computational Linguistics. Also: Research Memorandum CS-97-04, Department of Computer Science, University of Sheffield.
- Kroto, A., Gaizauskas, R., and Wilks, Y. (1994). Acquiring a stochastic context-free grammar from the Penn Treebank. In *Proc. of the Third Conference on the Cognitive Science of Natural Language Processing*.
- Ling, C. (1994). Learning the past tense of english verbs: the symbolic pattern associators vs. connectionist models. *Journal of Artificial Intelligence Research*, 1:209–229.
- Lloyd, J. (1987). *Foundations of Logic Programming*. Springer-Verlag, Berlin. Second edition.
- Magerman, D. (1995). Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Cambridge, MA.
- McCarthy, J. (1959). Programs with commonsense. In *Mechanisation of thought processes*, volume 1. Her Majesty's Stationery Office, pages 75–91, London. Reprinted (with an added section on 'Situations, Actions and Causal Laws') in *Semantic Information Processing*, ed. M. Minsky (Cambridge, MA: MIT Press (1963)).

- Mooney, R. (1997). Inductive logic programming for natural language processing. In Muggleton, S., editor, *Proceedings of the Sixth International Workshop on Inductive Logic Programming*, pages 3–21. Springer-Verlag, Berlin. LNAI 1314.
- Mooney, R. and Califf, M. (1995). Induction of first-order decision lists: Results on learning the past tense of english verbs. *Journal of Artificial Intelligence Research*, 3:1–24.
- Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, 8(4):295–318.
- Muggleton, S. (1994a). Bayesian inductive logic programming. In Cohen, W. and Hirsh, H., editors, *Proceedings of the Eleventh International Machine Learning Conference*, pages 371–379, San Mateo, CA. Morgan-Kaufmann. Keynote presentation.
- Muggleton, S. (1994b). Bayesian inductive logic programming. In Warmuth, M., editor, *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, pages 3–11, New York. ACM Press. Keynote presentation.
- Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing*, 13:245–286.
- Muggleton, S. (1998). Completing inverse entailment. In Page, C., editor, *Proceedings of the Eighth International Workshop on Inductive Logic Programming (ILP-98)*, LNAI 1446, pages 245–249. Springer-Verlag, Berlin.
- Muggleton, S. (2000). Learning from positive data. *Machine Learning*. Accepted subject to revision.
- Muggleton, S. and Bain, M. (1999). Analogical prediction. In *Proc. of the 9th International Workshop on Inductive Logic Programming (ILP-99)*, pages 234–246, Berlin. Springer-Verlag.
- Muggleton, S. and Feng, C. (1992). Efficient induction of logic programs. In Muggleton, S., editor, *Inductive Logic Programming*, pages 281–298. Academic Press, London.
- Muggleton, S., King, R., and Sternberg, M. (1992). Protein secondary structure prediction using logic-based machine learning. *Protein Engineering*, 5(7):647–657.
- Muggleton, S. and De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629–679.
ftp://ftp.cs.york.ac.uk/pub/MLOORUP/Papers/lpj.ps.gz
- Nienhuys-Cheng, S.-H. and de Wolf, R. (1997). *Foundations of Inductive Logic Programming*. Springer-Verlag, Berlin. LNAI 1228.
- Plotkin, G. (1971). *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University.
- Rumelhart, D. and McClelland, J. (1986). On learning the past tense of english verbs. In *Explorations in the Micro-Structure of Cognition Vol. II*, pages 216–271. MIT Press, Cambridge, MA.
- Shapiro, E. (1983). *Algorithmic Program Debugging*. PhD thesis, Yale University, MIT Press.
- Srinivasan, A., Muggleton, R. K. S., and Sternberg, M. (1997). Carcinogenesis predictions using ILP. In Lavrač, N. and Džeroski, S., editors, *Proceedings of*

- the *Seventh International Workshop on Inductive Logic Programming*, pages 273–287. Springer-Verlag, Berlin. LNAI 1297.
- Srinivasan, A., Muggleton, S., King, R., and Sternberg, M. (1996). Theories for mutagenicity: a study of first-order and feature based induction. *Artificial Intelligence*, 85(1,2):277–299.
- Sternberg, M., King, R., Lewis, R., and Muggleton, S. (1994). Application of machine learning to structural molecular biology. *Philosophical Transactions of the Royal Society B*, 344:365–371.
- Thompson, C., Mooney, R., and Tang, L. (1997). Learning to parse natural language database queries into logical form. In *Workshop on Automata Induction, Grammatical Inference and Language Acquisition*. Paper accessible from www-univ-st-etienne.fr/eurise/pdupont.html
- Turcotte, M., Muggleton, S., and Sternberg, M. (1998). Protein fold recognition. In Page, C., editor, *Proc. of the 8th International Workshop on Inductive Logic Programming (ILP-98)*, LNAI 1446, pages 53–64, Berlin. Springer-Verlag.
- Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59(236):435–460.
- Yamamoto, A. (1997). Which hypotheses can be found with inverse entailment? In Lavrač, N. and Džeroski, S., editors, *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, pages 296–308. Springer-Verlag, Berlin. LNAI 1297.
- Zelle, J. and Mooney, R. (1993). Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 817–822, San Mateo, CA. Morgan Kaufmann.
- Zelle, J. and Mooney, R. (1996a). Comparative results on using inductive logic programming for corpus-based parser construction. In *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*, pages 355–369. Springer, Berlin.
- Zelle, J. and Mooney, R. (1996b). Learning to parse database queries using Inductive Logic Programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1050–1055, Portland, Oregon. AAAI Press/MIT Press.

X

POSSIBILISTIC LOGIC

Chapter 15

DECISION, NONMONOTONIC REASONING AND POSSIBILISTIC LOGIC

An Introductory Survey of Recent Results

Salem Benferhat, Didier Dubois, Hélène Fargier, Henri Prade,
*Institut de Recherche en Informatique de Toulouse (IRIT), Université Paul Sabatier
118 route de Narbonne
31062 Toulouse Cedex 4
France
benferha@irit.fr, dubois@irit.fr, fargier@irit.fr, prade@irit.fr*

Régis Sabbadin
*INRA, Unité de Biométrie et Intelligence Artificielle
BP 27, 31326 Castanet-Tolosan cedex
France
sabbadin@toulouse.inra.fr*

Abstract The paper surveys recent AI-oriented works in qualitative decision developed by the authors in the framework of possibility theory. Lottery-based and act-based axiomatics underlying pessimistic and optimistic criteria for decision under uncertainty are first briefly restated, when uncertainty and preferences are encoded with an ordinal scale. A logical machinery capable of computing optimal decisions in the sense of these criteria is presented. Then an approach to qualitative decision under uncertainty which does not require a commensurateness hypothesis between the uncertainty and the preference scales is proposed; this approach is closely related to nonmonotonic reasoning, but turns out to be ineffective for practical decision. Lastly, the modeling of preference as prioritized sets of goals, as sets of solutions reaching some given level of satisfaction, or in terms of possibilistic constraints is discussed briefly.

Keywords: Decision theory, nonmonotonic reasoning, possibility theory, preference, uncertainty

1 INTRODUCTION

Logic and decision belong to two different traditions; the first is concerned with consistency and inference and is oriented towards symbolic processing, while the second deals with trade-offs (and possibly with uncertainty), and is more numerically inclined. However non-classical logics developed in AI often use ordering structures, while a need is now expressed in decision analysis for more qualitative evaluations which only require ordinal scales (rather than numerical ones) for grading uncertainty or utility. Among weighted logics, possibilistic logic, based on the conjoint use of classical logic and qualitative possibility theory (Dubois and Prade, 1998) (Zadeh, 1978) offers a framework at the meeting point of the two traditions.

Possibility theory and possibilistic logic have been already shown to provide a convenient setting for modeling and handling nonmonotonic reasoning (Benferhat et al., 1992) (Benferhat et al., 1997b) (Gärdenfors and Makinson, 1994). However, the possibilistic logic framework can be used for modeling not only pieces of knowledge pervaded with uncertainty, but also preferences, as sets of prioritized goals in decision under uncertainty (Lang, 1991), (Dubois et al., 1998d), (Dubois et al., 1999). Generally speaking, decision analysis requires both the expression of the user's preferences and of the available knowledge about the world. In classical approaches, preferences are encoded by numerical value functions and knowledge by probability distributions. Since such functions and distributions are not always directly accessible, there is a need for a representation of information which is more granular, closer to human expression, and more explanation-oriented. AI representational frameworks based on (non-classical) logic formulas, on graphs or on constraints can be useful in that respect. Besides, because preferences and often knowledge are human-originated, there is also a need for approaches which are more qualitative than the ones developed in decision theory until now.

This paper provides an overview of recent work by the authors pertaining to the logical handling of qualitative decision problems in the setting of possibility theory. The paper is organized in three main sections. Starting with the motivations expressed above, a formal approach to decision under uncertainty has been proposed where preferences and uncertainty are graded on ordinal (linearly ordered) scales which are commensurate. Axiomatic underpinnings (restated in Sections 2.2 and 2.3), either a la (von Neumann and Morgenstern, 1944), or more recently a la (Savage, 1954), have been provided for a pessimistic and an optimistic qualitative counterparts to probabilistic expectation, which are defined in the framework of qualitative possibility theory. These two decision criteria are first informally presented in Section 2.1. Section 3 provides an overview of another approach which no longer requires a commensurateness hypothesis between the utility and uncertainty scales. Then it is shown that we are naturally led to a decision procedure which, as in nonmonotonic reasoning, assumes that one of the most normal situations compatible with the available information prevails. More precisely, it provides decision-theoretic foundations for nonmonotonic consequence relations obeying (Kraus et al., 1990)'s postulates, for which possibility theory-based

semantics exist as well. Lastly, Section 4 advocates a possibilistic logic handling of preferences, possibly generated from constraints, or as prioritized goals, or still by means of specified classes of examples. Then the aggregation, as well as the modification of preferences can be envisaged at the symbolic level in a logical setting. Lastly, a logical machinery encodes the maximization of the two qualitative "expectations" introduced in Section 2 when the preferences are expressed under the form of prioritized goals, and the knowledge as sets of beliefs having various levels of strength.

2 A FRAMEWORK FOR QUALITATIVE DECISION

Let S be a finite set of elements called "states". States encode such things as possible situations and states of affairs. In the (Savage, 1954) framework, the consequence of a decision depends on the state of the world in which it takes place. For instance in a restaurant one must choose our dishes without being totally sure that the food is good. If X is a set of possible consequences, a decision f is a mapping from S to X . The decision-maker has some knowledge of the actual state and some preference on the consequences of his decision. The usual approach to decision-making under uncertainty is based on representing uncertainty on states by a unique probability distribution p and the decision-maker by a utility function μ on consequences. Then decisions are selected on the basis of expected utility, defined (when S is finite) by

$$E(f) = \sum_{s \in S} p(s) \cdot \mu(f(s)).$$

An increasing interest for qualitative decision has recently appeared in the AI community : The term "qualitative decision theory" refers to more than one kind of representation. Some approaches consider only all-or-nothing notions of utility and plausibility, for instance (Bonet and Geffner, 1996); others use integer-valued functions (Tan and Pearl, 1994), (Pearl, 1993). (Boutilier, 1994) exploits preference orderings and plausibility orderings by focusing on the most plausible states.

In (Dubois and Prade, 1995b), a qualitative analog to (von Neumann and Morgenstern, 1944)'s postulates, intended for rational decision under ordinal uncertainty has been proved to be equivalent to the maximization of a qualitative utility function. This is recalled in Section 2.2. A (Savage, 1954)-like qualitative decision theory is presented in Section 2.3. Qualitative decision criteria are first presented before giving their axiomatic justifications.

2.1 PESSIMISTIC AND OPTIMISTIC DECISION CRITERIA

When uncertainty is just due to a lack of information and only the result of the present decision matters (as in the restaurant), the idea of expected utility is less attractive. For this reason a purely possibilistic approach to decision under uncertainty that addresses this kind of situation has been developed.

Possibility distributions which map a set of interpretations to a scale are a convenient way of encoding complete pre-orderings. It makes sense, if information is qualitative, to represent incomplete knowledge on the actual state by a possibility distribution π on S , the set of (mutually exclusive) states, with values in a plausibility scale L and the decision-maker's preferences on X , the set of (mutually exclusive) consequences, by means of another possibility distribution μ with values on a preference scale T . Let 0 (resp. \emptyset) and 1 (resp. 1) denote the bottom and top elements of L (resp. T). The following representational conventions are assumed for possibility distributions. $\pi(s) = 0$ means that s is definitely impossible according to what is known, and $\mu(x) = \emptyset$ that x is unacceptable as a consequence. The greater $\pi(s)$ (resp. $\mu(x)$), the more plausible s as being the real state of the world (resp. the more acceptable x as a consequence). $\pi(s) = 1$ (resp. $\mu(x) = 1$) means s is among the most plausible (normal) states and there may be several s such that $\pi(s) = 1$ (resp. x is among the most preferred consequences). The utility of a decision f whose consequence in state s is $x = f(s) \in X$ for all states s , can be evaluated by combining the plausibilities $\pi(s)$ and the utilities $\mu(x)$ in a suitable way. Two qualitative criteria that evaluate the worth of decision f have been proposed in the literature, provided that a commensurability assumption between plausibility and preference is made:

Definition 1 Pessimistic criterion

$$E_*(f) = \inf_{s \in S} \max(n(\pi(s)), \mu(f(s))), \quad (15.1)$$

where n is an order-reversing mapping from L to T (i.e. $n(0) = 1$, $n(1) = \emptyset$, n is a strictly decreasing bijection of L to T).

Definition 2 Optimistic criterion

$$E^*(f) = \sup_{s \in S} \min(m(\pi(s)), \mu(f(s))), \quad (15.2)$$

where m is order-preserving mapping from L to T (i.e. $m(0) = \emptyset$, $m(1) = 1$ and m is strictly increasing).

These criteria are nothing but the necessity and the possibility measures of fuzzy events, and are special cases of Sugeno integrals (Dubois and Prade, 1995b). Maximizing $E_*(f)$ means finding a decision f , all the highly plausible consequences of which are also highly preferred. The definition of "highly plausible" is decision-dependent and reflects the compromise between high plausibility and low utility expressed by the order-reversing map between the plausibility valuation set L and the utility valuation set T . The pessimistic utility $E_*(f)$ is small as soon as there exists a possible consequence of f which is both highly plausible and bad with respect to preferences. This is clearly a risk-averse and thus a pessimistic attitude. It generalizes the max-min (Wald, 1950) criterion, which is based on the worst possible consequence of the considered decision in the absence of probabilistic knowledge, since if π is then the set characteristic function of a subset A of states, $E_*(f)$ is the utility of the worst consequence of states in A , however unlikely they are. But the

possibilistic criterion is less pessimistic. It focuses on the idea of usuality since it relies on the worst *plausible* consequences induced by the decision (extremely unusual consequences are neglected). On the contrary, the criterion $E^*(f)$ corresponds to an optimistic attitude since it is high as soon as there exists a possible consequence of f which is both highly plausible and highly prized. It generalizes the maximax optimistic criterion. The latter evaluation can be used as a secondary criterion, for breaking ties between decisions which are equivalent w.r.t. the pessimistic criterion.

For binary acts the expression of the utilities is simple: let xAy denote a decision that gives consequence x if A occurs and y otherwise, and assume x is preferred to y ($\mu(x) > \mu(y)$). Assume $L = T$. Introducing the possibility of event A (Zadeh, 1978) and its necessity, respectively defined by

$$1. \Pi(A) = \sup_{s \in A} \pi(s),$$

$$2. N(A) = \inf_{s \notin A} \pi(s),$$

the criteria $E_*(f)$ and $E^*(f)$ can be simplified when f is a binary act. Observe that $N(A) = n(\Pi(\neg A))$, i.e., the necessity of A amounts to the impossibility of $\neg A$, and $\min(N(A), N(\neg A)) = 0$ (since $\Pi(S) = 1$ is postulated and $\Pi(S) = \max(\Pi(A), \Pi(\neg A))$, due to $S = A \cup \neg A$), so that:

$$E_*(xAy) = \max(\mu(y), \min(N(A), \mu(x))) = \min(\mu(x), \max(N(A), \mu(y))).$$

This form of the pessimistic utility is easy to understand: if the agent is sure enough that A occurs ($N(A) > \mu(x)$) then the utility of the act xAy is $\mu(x)$. If the agent has too little knowledge ($\max(N(A), N(\neg A)) < \mu(y)$), then cautiousness prevails and the utility is $\mu(y)$, the worst case. Of course the same happens if the agent is at least somewhat certain that $\neg A$ occurs. If the agent's certainty that A occurs is positive but not extreme, the utility reflects the certainty level and is equal to $N(A)$. Note that the utility of the binary decision is the median of $\{\mu(x), N(A), \mu(y)\}$, thus contrasting with expected utility, which is a mean.

Similarly, the optimistic utility of the binary act takes the simplified form (if function m in (15.2) is identity):

$$E^*(xAy) = \max(\min(\Pi(A), \mu(x)), \mu(y))$$

and can be interpreted similarly as the median of $\{\mu(x), \Pi(A), \mu(y)\}$, but here the utility is $\mu(x)$ as soon as the agent believes that obtaining x is possible enough ($\Pi(A) > \mu(x)$).

Example 1 Consider the omelette example of (Savage, 1954, pages 13 to 15). The problem is to make a six-egg omelette from a five-egg one. The new egg can be fresh or rotten. There are three feasible acts: break the egg in the omelette (BIO); break it apart in a cup (BAC) or throw it away (TA). Assume that utilities and degrees of certainty belong to the same totally ordered scale, here $L = T = \{0, a, b, c, d, 1\}$ where $0 < a < b < c < d < 1$, equipped with its involutive order-reversing map n . The set of 6 consequences is given in Table 1.

Act/State	fresh egg (F)	rotten egg (R)
BIO	6-egg omelette (1)	nothing to eat (0)
BAC	6-egg omelette, cup to wash (d)	5-egg omelette, cup to wash (b)
TA	5-egg omelette, a spoiled egg (a)	5-egg omelette (c)

Table 15.1 States, acts and consequences in Savage's omelette example.

Grades between parentheses indicate a reasonable encoding of the utility ordering of consequences. The reader can easily check that he/she agrees with this ordering. Only two states (fresh F, rotten R) are present, so we deal with binary acts. The utilities of the three acts in the egg example are given as functions of states F and R where $N(F) = n(\Pi(R))$, $N(R) = n(\Pi(F))$, $\min(N(F), N(R)) = 0$ and $\max(\Pi(F)), \Pi(R)) = 1$:

$$\begin{aligned} E_*(BIO) &= \min(1, \max(N(F), 0)) = N(F) ; & E^*(BIO) &= \Pi(F) ; \\ E_*(BAC) &= \min(d, \max(N(F), b)) ; & E^*(BAC) &= \max(\min(\Pi(F), d), b) ; \\ E_*(TA) &= \min(\max(N(R), a), c). & E^*(TA) &= \max(a, \min(\Pi(R), c)). \end{aligned}$$

The criterion E_* recommends act BAC in case of relative ignorance on the egg state, that is when $\max(N(F), N(R))$ is not high enough (less than b). In practice, it is advisable to act cautiously, and to break the egg in a spare cup in case of serious doubt. It sounds like a more realistic advice than the one prescribed on the basis of the most likely state (like in (Boutilier, 1994)). Indeed, in the latter case, in spite of R (resp. F) being slightly certain, the chosen act would be TA (resp. BIO), whose results may be not so good. In case of ignorance Boutilier's strategy gives no advice since the utility vectors $(1, 0)$ for BIO, (d, b) for BAC and (a, c) for TA are incomparable. Results would be unchanged if "nothing to eat" had a non-zero utility, strictly less than a . Here, the optimistic criterion may lead to adventurous decisions since it chooses BIO in the case of ignorance, i.e. when $\min(\Pi(F), \Pi(R))$ is high.

The use of pessimistic and optimistic criteria (15.1) and (15.2) in multi-stage decision making has been investigated by (Sabbadin et al., 1998) who developed a dynamic programming-like approach. They have also been used in scheduling problems, see (Dubois et al., 1995); in this paper, apart from genuine scheduling instances, a toy example shows how a person, who has to arrive at least approximately on time at a meeting in the morning and who dislikes to get up too early, can decide at what time (s)he has to leave home, taking into account the uncertainty about the travel time from home to the meeting. This example perfectly illustrates that the pessimistic criterion makes a tradeoff between preference and uncertainty by proposing a decision which makes an individual sure enough to arrive at least almost on time, by neglecting only very long travel times which have a very low plausibility. Moreover, possibilistic planners have been recently proposed (Guérard and Alami, 1999).

2.2 AXIOMATICS FOR POSSIBILISTIC LOTTERIES

The pessimistic criterion has been axiomatically justified by (Dubois and Prade, 1995b) and by (Dubois et al., 1998c) in a simplified form, in the style of (von Neumann and Morgenstern, 1944)'s utility theory. It has been generalized to partial preferences by (Zapico, 1999). The idea is that if the uncertainty on the state is represented by π , each decision f induces on the set of consequences X a possibility distribution such that $\pi_f(x) = \Pi(f^{-1}(x))$, where $\pi_f(x)$ is the plausibility of getting x under decision f . So ranking decisions comes down to ranking possibility distributions on X . A possibility distribution then represents the more or less plausible consequences of an act. Assume the decision-maker supplies an ordering between possibility distributions on X , thus expressing his/her attitude in front of risk, that is, in front of various possibilities of happy and unhappy consequences in X . For instance, the decision maker may dislike acts associated with possibility distributions containing bad consequences with a high plausibility. The question is to know what kind of axioms on the ordering between possibility distributions on X make it representable by the ranking of decisions according to the above pessimistic or optimistic criteria. Let $(\lambda/\pi, \mu/\pi')$ denote the "qualitative lottery" yielding π with plausibility λ and π' with plausibility μ . Of course, $\max(\lambda, \mu) = 1$, a counterpart in possibility theory of the probabilistic normalization. The following are the pessimistic axioms (a dual set of axioms can be devised for the optimistic criterion):

1. The set of possibility distributions is equipped with a *complete preordering* structure (\succeq, \sim, \succ) where \succeq , \sim , \succ denote weak preference, indifference and strict preference respectively;
2. *Independence*: $\pi_1 \sim \pi_2 \Rightarrow (\lambda/\pi_1, \mu/\pi) \sim (\lambda/\pi_2, \mu/\pi)$;
3. *Continuity*: if $\pi > \pi'$ then $\exists \lambda \in T, \pi' \sim (1/\pi, \lambda/X)$;
4. *Reduction of lotteries*: $(\lambda/(\alpha/\pi, \beta/\pi'), \mu/\pi') \sim (\gamma/\pi, \delta/\pi')$ where $\gamma = \min(\alpha, \lambda)$ and $\delta = \max(\mu, \min(\lambda, \beta))$;
5. *Risk aversion*: $\pi \geq \pi' \Rightarrow \pi' \succeq \pi$.

The risk aversion axiom states that the less informative is π , the more risky is the situation. The worst epistemic state is total ignorance ($\pi(s) = 1$ for all s). Continuity says that the utility of π smoothly goes down if the uncertainty about π raises. It can be proved (Dubois et al., 1998c) that if the knowledge is represented by a subset A of possible states, then $\exists x \in A, x \sim A$. This property, violated by expected utility, suggests that contrary to it, the pessimistic utility is not based on the idea of average. The latter agrees well with repeated decisions, but the former makes sense for one-shot decisions.

The reduction of lotteries suggests that if lottery $(\alpha/\pi, \beta/\pi')$ is embedded in the lottery $(\lambda/x, \mu/\pi')$ at place x , π obtains in two sequential steps with possibility $\min(\alpha, \beta)$ and π' obtains either directly with possibility μ or in two steps with possibility $\min(\alpha, \beta)$. The reduction of lotteries property is

due to the fact that the whole setting relies on a qualitative counterpart of the structure of probabilistic mixtures that underlies the von Neumann and Morgenstern approach. This is due to the closure of the set of possibility measures under weighted maxima: if Π and Π' are possibility measures then, $\max(f(\Pi), f'(\Pi'))$ is a possibility measure again, where f and f' are monotonic transformations of the possibility scale, with $f(0) = f'(0) = 0$, and $\max(f(1), f'(1)) = 1$. This is the only aggregation that preserves possibility measures. See (Dubois et al., 1996) on possibilistic mixtures.

2.3 ACT-BASED JUSTIFICATION OF POSSIBILITY THEORY

In Savage decision theory, a decision problem is cast in the framework of a set of states (of the world) and a set of potential consequences of decisions. An act is viewed as a mapping from the state space S to the consequence set X , namely, in each state, an act produces a well-defined result. Savage is interested in the following problem: Starting from a user-driven preference relation over acts, and axioms that such a preference relation should obey if the decision-maker is “rational”, construct a partial ordering relation representing uncertainty on events in the state space, and a preference relation on the consequences. If the preference on acts satisfies suitable properties then the uncertainty on events should be represented by a probability distribution, the preference relation on consequences by a numerical utility function, and the preference relation on acts by expected utility: an act is preferred to another if and only if it has a better expected utility.

An axiomatic justification of possibility has also been developed in the style of Savage (Dubois et al., 1998f) (Dubois et al., 1998e). The problem is one of representing the preference relation of an agent over the set $D = X^S$ of potential decisions by means of a utility function whose form is dictated by the properties of this preference relation \succeq . Let fAf' denotes a decision that coincides with f if A occurs and f' if not. A set of axioms for the pessimistic utility is as follows:

1. (D, \succeq) is a complete preorder;
2. There exist two decisions f and g where f is strictly preferred to g ($f \succ g$);
3. If x and y denote *constant acts* that yield consequence x and y for any state, then for any decision f and any event A : $x \succeq y$ implies $xAf \succeq yAf$;
4. If $f \succ x$ and $f \succ g$ then $f \succ x \vee g$, where x is a constant act;
5. If $g \succ f$ and $h \succ f$ then $g \wedge h \succ f$.

Theorem 1 If the preference relation over D satisfies the above 5 axioms then there exists a possibility distribution on S , a preference function on X and an

order-reversing function between plausibility and preference scales such that $f \succ f'$ if and only if $E_*(f) > E_*(f')$ using the pessimistic possibilistic utility.

The optimistic criterion can be justified likewise, moving the constant act condition from axiom 4 to axiom 5. In fact axioms 1, 2, and 3 are enough to show that the uncertainty on states is captured by a monotonic set-function. Axiom 5 ensures the min-decomposition of the uncertainty measure (which is then a necessity function). The pessimistic nature of axiom 5 can be guessed from its equivalent form: $f' Af \succ f$ implies $f \succeq f' Af$. If decision f is improved by f' when A occurs it means that A is sure enough to occur and there is no way of improving decision f by changing it in case A does not occur (the agent neglects this possibility of non-occurrence).

3 QUALITATIVE DECISION WITHOUT COMMENSURATENESS

The commensurateness of uncertainty and preference scales may be found very demanding. In this section we survey what can be done without this hypothesis, and then explain the relation of this approach to decision with nonmonotonic reasoning.

3.1 THE LIKELY DOMINANCE RULE

In this sub-section, we consider the following problem in the Savage setting (to which Section 2.3 already referred). An uncertainty relation on events is assumed to be given, and a preference relation on consequences is independently given as well. How can a preference relation on acts be recovered with a *purely symbolic approach*? Indeed, decision makers are not necessarily capable of describing their state of uncertainty by means of a probability distribution, nor may they be able to quantify their preferences on consequences. Suppose a decision-maker only supplies a relative likelihood ordering on events that describes some knowledge about the state of the world, and a preference ordering on the consequences of feasible acts. A natural, intuitively appealing lifting technique that solves the act-ranking problem is as follows: namely, an act f is preferred to another g if the (relative) likelihood that f outperforms g statewisely is greater than the likelihood that g outperforms f .

Let $(2^S, >_L)$ be a "likelihood" relation on events viewed as sets of states in S . We assume that $>_L$ is a strict partial order ($>_L$ is transitive and asymmetric), is non-trivial ($S >_L \emptyset$), and faithful to deductive inference:

$$A \subseteq B \Rightarrow \neg(A >_L B) \text{ (inclusion-monotonicity).}$$

The above property states that if A implies B , then B cannot be less likely than A . Let X be another finite set that represents consequences of acts. The preference on X is supposed to be just a weak order \geq_P (reflexive, complete, and transitive). It is assumed that X has at least two elements x and y s.t. $x >_P y$. An act is again viewed as a mapping $f : S \rightarrow X$.

If no commensurateness assumption can be made between the likelihood relation on events and the preference relation on consequences, a natural way of lifting the pair $(>_L, \geq_P)$ to X^S is as follows: an act f is more promising than an act g if and only if the event formed by the disjunction of states in which f gives results at least as good as g , is more likely than the event formed by the disjunction of states in which g gives results at least as good as f . A state s is at least as promising a state for f as for g iff $f(s) \geq_P g(s)$. Hence, we define the strict preference \succ between acts by the likely dominance rule:

$$f \succ g \text{ iff } [f \geq_P g] >_L [g \geq_P f] \text{ where } [f \geq_P g] = \{s, f(s) \geq_P g(s)\};$$

This lifting principle¹, however intuitive it may look, turns out to be incompatible with a general probabilistic representation of uncertainty. Indeed, applying this technique to a comparative probability ordering on events may lead to a non-transitive strict preference relation on acts. The difficulty is very similar to the one faced in voting theories where a priori natural procedures, like the Condorcet pairwise majority rule (where a candidate C_1 is preferred to another C_2 if there are more voters who prefer C_1 to C_2 than voters who prefer C_2 to C_1), exhibit counter-intuitive intransitivities (see e.g., Moulin, 1988). In fact, in a probabilistic setting, the lifting principle exhibits a Condorcet effect proper. Thus, a decision procedure satisfying the lifting axiom cannot be founded on a comparative probability (except in a very special case). This is not the case with comparative necessity relations recalled in the Appendix. It can be shown (Dubois et al., 1997a) that, starting with $(2^S, >_N)$ and (X, \geq_P) , the strict preference structure (X^S, \succ) induced via the likely dominance rule:

$$f \succ g \Leftrightarrow [f \geq_P g] >_N [g \geq_P f] \Leftrightarrow [f >_P g] >_\Pi [g >_P f] \quad (\text{L}\Pi)$$

is transitive.

3.2 AXIOMATICS OF ORDINAL DECISION UNDER UNCERTAINTY

In order to accommodate the likely dominance rule and motivate it by a purely ordinal approach, we have proposed the following axiomatic framework, that adheres to Savage's as much as possible. Let X^S be a set of acts equipped with a strict preference relation \succ and let \succeq be the transposed complement of \succ (i.e., $f \succeq g$ if and only if $\text{not}(g \succ f)$). The modified set of axioms is (axioms S'_i are variants of Savage axioms S_i ; others are new):

S'_1 : (X^S, \succ) is a transitive, asymmetric, partially ordered set.

Note that this is weaker than the Savage requirement that \succeq be a weak order. We could have started from a reflexive and transitive relation \succeq . Then we

¹Lifting principles are commonly used in AI for lifting a preorder on a set to its power set, see, e.g. (Halpern, 1997). Here the situation is more general, since we lift two preorders on X and S to X^S .

could have distinguished between indifference ($f \succeq g$ and $g \succeq f$) and incomparability ($\neg(f \succeq g)$ and $\neg(g \succeq f)$). Here starting from the strict preference, this distinction is not made and the weak preference \succeq is not transitive.

S_2 : (X^S, \succeq) satisfies the sure-thing principle: $fAh \succeq gAh$ iff $fAh' \succeq gAh'$.

The sure-thing principle says that the preference between fAh and gAh does not depend on their common part h . It enables two notions to be simply defined, namely conditional preference and null events. Act f is said to be weakly preferred to g , *conditioned* on an event A if and only if $fAh \succeq gAh$ for some h . This is denoted by $(f \succeq g)_A$. An event A is said to be *null* if and only if $\forall f, \forall g, (f \succeq g)_A$ holds.

U : $(f \succeq g)_A$ and $(f \succeq g)_{\bar{A}} \Rightarrow f \succeq g$ (unanimity).

where \bar{A} denotes the complement of A in S . This axiom is a consequence of the Savage framework, but must be added here due to the weakness of S'_1 . Among acts in X^S are *constant acts* such that: $\exists x \in X, \forall s \in S, f(s) = x$. Such an act is denoted x . It seems reasonable to identify the set of constant acts $\{x, x \in X\}$ and X . The preference on X can be induced from (X^S, \succeq) as follows. Given (X^S, \succeq) , the preference relation \geq_P on X is of the form $\forall x, y \in X, x \geq_P y$ if and only if $x \succeq y$.

S_3 : $\forall A \subseteq S, A$ not null, $(x \succeq y)_A$ if and only if $x \geq_P y$.

Projecting X^S on 2^S yields a likelihood relation, that is, a binary relation $>_L$ among events defined by $A >_L B$ iff $xAy \succeq xBy$ for some $x >_P y$. Axiom S_4 ensures that it does not depend on x and y .

S_4 : $\forall x, y, x', y' \in X$ s.t. $x >_P y, x' >_P y', xAy \succeq xBy \Leftrightarrow x'Ay' \succeq x'By'$.

S'_5 : $\exists x, y, z$ three constant acts such that $x \succ y \succ z$.

The axiom *LD* below enables the partial ordering on acts (\succ) to be reconstructed from the likelihood relation on events ($>_L$) associated with (\succ) and the preference relation on consequences (\geq_P) associated with \succeq :

LD: (*likely dominance*) $f \succ g \Leftrightarrow [f \geq_P g] >_L [g \geq_P f]$ where \geq_P is the projection of \succeq on X (due to S_3) and $>_L$ is the projection of \succ on events (due to S_4).

Based on this set of axioms, the following representation theorem has been established (Dubois et al., 1998a):

Theorem 2 If a partially ordered set of acts (X^S, \succeq) satisfies $(S'_1, S_2, S_3, S_4, S'_5, U)$, and the likely dominance axiom *L*, then the relation \geq_P on consequences is a weak order, and there is a family \mathcal{F} of possibility orderings $>_\Pi$ on S and

a relation \geq_P on X such as:

$$f > g \Leftrightarrow \forall >_{\Pi} \in \mathcal{F}, [f >_P g] >_{\Pi} [g >_P f]$$

where $>_{\Pi}$ is the strict part of a possibility ordering \geq_{Π} .

See the Appendix for a refresher on possibility orderings. They are orderings of events which can only be represented by possibility measures. Moreover the likely dominance principle *LD* can be recovered as the only possible decision rule, if we add to $S'_1, U, S_2, S_3, S_4, S'_5$, an ordinal invariance axiom (Fargier and Perny, 1999). The following definition is needed for stating this axiom. Two pairs of acts (f, g) and (f', g') are called *statewise order equivalent* if and only if $\forall s \in S, f(s) \geq_P g(s)$ if and only if $f'(s) \geq_P g'(s)$. This is denoted $(f, g) \equiv (f', g')$. Then, the *Ordinal Invariance* axiom, which fully embodies the requirement of a purely ordinal approach to decision-making under uncertainty, reads:

$$OI: \forall f, f', g, g' \in X^S, \text{ if } (f, g) \equiv (f', g') \text{ then } f \succeq g \Leftrightarrow f' \succeq g'.$$

This axiom does express that what matter for preference between two acts are the relative positions of consequences of acts for each state, not the consequences themselves, nor the positions of these acts relative to other acts. It emphasizes the purely ordinal nature of the approach based on the likely dominance rule, whereby the idea of scaling preference is fully given up.

3.3 DECISION THEORY BASED ON NONMONOTONIC REASONING

It can be proved (Dubois et al., 1997a; Dubois et al., 1998b) that under $S'_1, S_2, S_3, U, S_4, S'_5$ and the lifting axiom *L*, the induced likelihood ordering verifies the following principles:

- for A, B, C disjoint, $A \cup C >_L B$ and $B \cup C >_L A$ imply $C >_L A \cup B$
- $A >_L \bar{A}$ and $A \subseteq B$ imply $B >_L \bar{B}$.

These properties are satisfied by so-called discriminax likelihood relations (see the Appendix), and are characteristic of nonmonotonic reasoning in the sense of Kraus, Lehmann and Magidor (Friedman and Halpern, 1996).

Consider nonmonotonic inferences of the form $A \mid\sim B$ which means that if all is known by an agent is that event A obtains, then the agent will reason as if B obtained as well. In other words, in the context where the agent knows A only, he accepts B . Basic properties of nonmonotonic inference have been advocated by (Kraus et al., 1990):

Right weakening: $A \mid\sim B$ and $B \subseteq C$ imply $A \mid\sim C$;

AND: $A \mid\sim B$ and $A \mid\sim C$ imply $A \mid\sim B \cap C$;

OR: $A \mid\sim C$ and $B \mid\sim C$ imply $A \cup B \mid\sim C$;

Cautious monotony (CM): $A \mid\sim B$ and $A \mid\sim C$ imply $A \cap B \mid\sim C$;

Cut: $A \mid\sim B$ and $A \cap B \mid\sim C$ imply $A \mid\sim C$.

Together with axiom $A \mid\sim A$, the above inference rules constitute what is called System P (for ‘preferential’) by these authors. The above rules of inference embody the notion of plausible inference in the presence of incomplete information. Namely, they describe the properties of deduction under the assumption that the state of the world is as normal as can be. Possibility theory is closely related to preferential nonmonotonic consequence relations, as defined by the above postulates.

Indeed a default rule “if p then q , generally”, denoted $p \rightsquigarrow q$, is understood as a constraint of the form $\Pi(p \wedge q) > \Pi(p \wedge \neg q)$ where Π is a possibility measure (i.e. Π is max-decomposable under disjunction; see the Appendix). Such a constraint means that in the context where p is true, “ q true” is more possible or plausible than “ q false”. Then, given a set of defaults represented by constraints of this form, looking for the greatest possibility measure which satisfies the set of constraints, provides a way for rank-ordering the defaults according to their specificity in a way which fully parallels (Pearl, 1990)’s Z procedure. A set of defaults $\Delta = \{p_i \rightsquigarrow q_i, i = 1 \dots k\}$ can then be encoded by possibilistic logic formulas of the form $(\neg p_i \vee q_i, \alpha_i)$ where α_i encodes a priority level reflecting the ordering. Then, these formulas are processed through an extended logical machinery based on the possibilistic logic resolution rule² and a refutation method (Dubois et al., 1994). Adding the factual information on the considered case with the highest priority level, and applying possibilistic inference has been proved to be equivalent to the rational closure entailment of (Lehmann and Magidor, 1992). Working with all the possibility measures satisfying the set of constraints Δ (Dubois and Prade, 1995a) instead of selecting the greatest one and building an ordering, provides a representation of the system P of postulates for nonmonotonic consequence relations proposed by (Kraus et al., 1990) and leads to a more cautious inference system.

Assume no null events. Then, a nonmonotonic consequence relation $\mid\sim$ is preferential if and only if there exists a set \mathcal{F} of possibility orderings such that if $A \neq \emptyset$ (Dubois and Prade, 1995a):

$$A \mid\sim B \text{ iff } \forall >_{\Pi} \in \mathcal{F}, A \cap B >_{\Pi} A \cap \overline{B}.$$

$A \mid\sim B$ intuitively means that B is more likely than \overline{B} when A is true. Note that here the relation $>_{\Pi}$ is between disjoint events as in the representation theorem of Section 3.2, which thus provides the basis for decision-theoretic foundations for nonmonotonic consequence relations.

A decision-maker attitude in the setting of Section 3.2 is often either adventurous or very indecisive. For instance, if $>_L$ results in a total ordering of states then $f > g$ if and only if $f(s^*) \geq_P g(s^*)$, where s^* is the most plausible state. When the likelihood relation $>_L$ cannot discriminate between

²The possibilistic resolution rule reads $(\neg p \vee q, \alpha), (p \vee r, \beta) \vdash (q \vee r, \min(\alpha, \beta))$ in the propositional case. Soundness and completeness results between the syntactic and semantic parts of possibilistic logic can be found in (Dubois et al., 1994).

states, then it coincides with proper set inclusion; then $f \succ g$ if and only if $\forall s f(s) \geq_P g(s)$ (i.e. Pareto dominance) which is quite indecisive. This decision attitude, in both cases, looks hardly satisfactory, thus casting doubts on the possibility of a purely ordinal solution to the decision problem under uncertainty in the framework of even weakly transitive preference relations on acts. This situation is related to Arrow's impossibility theorem for the aggregations of complete preorders in social choice theory, see (Dubois et al., 1997a) (Dubois et al., 1998a). (Doyle and Wellman, 1991) have already discussed Arrow's theorem in the context of default reasoning when aggregating several partial preference preorders; see also (Fargier and Perny, 1999). This indecisiveness can be illustrated by considering the omelette example again.

Example 2 (continued)

Since only two states are present, the ordering of events can always be represented by only one probabilistic ordering. Let us apply the likely dominance rule. If fresh egg is more likely than rotten egg, then $[BIO >_P BAC] = [BIO >_P TA] = [BAC >_P TA] = \{\text{fresh}\} >_L [BAC >_P BIO] = [TA >_P BIO] = [TA >_P BAC] = \{\text{rotten}\}$ and the best act is clearly BIO . If the decision-maker thinks the egg is rotten, then the best act is TA . In case of total ignorance, the three acts are indifferent. So the decision making attitude induced by the approach is: break the egg in the omelette if you think the egg is fresh, throw it away if you think it is rotten, and do anything you like if you have no opinion (all acts being equally preferred, then). Clearly, this may result in many starving days, and garbage cans filled with lots of spoiled fresh eggs, in the case when the state of the egg is usually hard to tell. The pessimistic probabilistic criterion seems to deliver a more realistic advice than the one prescribed by the likely dominance rule which does not presuppose commensurability between uncertainty and utility. In the general case, the likelihood ordering is induced by a family of possibility distributions. This is still less discriminating than when this ordering is based on a single one from the family. So, the likely dominance principle does not look more attractive in the presence of a family of possibility orderings.

4 LOGICAL HANDLING OF PREFERENCES

Utility functions valued on ordinal scales can be viewed as a set of prioritized propositions representing goals. Other specifications of preference in terms of levels of satisfaction of prototypical solutions, or by means of constraints are considered. The logical counterpart of the combination of utility functions corresponding to different criteria is also described in this section (Benferhat et al., 1999).

4.1 PREFERENCES PROFILES AS PRIORITIZED SETS OF GOALS

Let U be a finite set of possible candidates. A utility function μ_C , associated with some criterion C , is a mapping from U to some valuation scale L . In many practical situations, a finite valuation scale $L = \{\alpha_0 = 0 < \alpha_1 < \dots < \alpha_m = 1\}$

is enough, first because the set of candidates is finite, and moreover humans are often only able to differentiate candidates through a small number of valuations. Criterion C is then viewed as a fuzzy set of candidates (Bellman and Zadeh, 1970).

Then the fuzzy set C defined by its membership function μ_C can be equivalently seen as a finite family of nested α -level cuts $C_\alpha = \{u \in U, \mu_C(u) \geq \alpha\}$ corresponding here to crisp (i.e., non-fuzzy) constraints or objectives. Note that $\alpha \leq \beta \Rightarrow C_\alpha \supseteq C_\beta$. C is equivalently represented by the set of constraints $N(C_{\alpha_i}) \geq n(\alpha_{i-1})$ for $i = 1, \dots, m$, where N is the necessity measure defined from $\pi = \mu_C$ (see Section 2.1), C_{α_i} is the α_i -cut of μ_C which represents the set of possible candidates having a degree of satisfaction at least equal to α_i and n is the order-reversing map of scale L , i.e. $n(\alpha_i) = \alpha_{m-i}$. The greater α_i , the smaller C_{α_i} . Note that if $C_{\alpha_i} = C_{\alpha_{i+1}}$ then the constraint $N(C_{\alpha_{i+1}}) \geq n(\alpha_i)$ is redundant, and can be ignored. This can be also reinterpreted in terms of priority: the goal of picking a candidate in C_{α_i} has priority $n(\alpha_{i-1})$, and the larger the α -cut, the more important the priority; in particular it is imperative that the chosen u has a non-zero degree of satisfaction, so C_{α_1} has priority 1. Indeed, $N(C_{\alpha_1}) = 1$.

This gives birth to a possibilistic knowledge base of the form $K = \{(c_{\alpha_i}, n(\alpha_{i-1})), i = 1, m\}$ where c_{α_i} denotes the proposition whose set of models is C_{α_i} . Preferences are thus expressed in terms of sets of crisp (nested) goals C_{α_i} with their respective levels of priority. Decisions violating goals with priority 1 have a level of acceptability equal to 0. This representation plays a basic role in the manipulation of qualitative preferences.

Conversely, a set of crisp goals (not necessarily nested) $K = \{(p_j, \rho_j), \rho_j \in L, j = 1, k\}$ with different levels of priority can always be represented in terms of a possibility distribution. Such a set of prioritized goals expresses a preference profile. The possibilistic logic semantics (Dubois et al., 1994) of K is given by the function π_K from the set of interpretations U to L , such that all the interpretations satisfying all the propositions in K get the highest possibility degree, namely 1, and the others are ranked w.r.t. the strongest proposition that they falsify, namely we get:

$$\pi_K(u) = \min_{j=1,k} \max(v_u(p_j), n(\rho_j)) \quad (15.3)$$

where $v_u(p_j) = 1$ if u is a model of p_j (i.e., an interpretation which makes it true) and $v_u(p_j) = 0$ if u falsifies p_j . $\pi_K(u)$ is all the smaller as u violates goals with higher priority.

It is worth noticing that (15.3) results from the application of the minimal specificity principle to the set of constraints (Dubois et al., 1994):

$$N(p_j) \geq \rho_j \text{ for } j = 1, k. \quad (15.4)$$

Indeed, (15.4) implicitly specifies a set of possibility distributions and the minimal specificity principle consists in choosing the greatest possibility distribution satisfying the constraints. This distribution allocates the greatest possibility level to each interpretation in agreement with the constraints. Associated with K is the possibility degree $\pi_K(u)$ defined by (15.3) which

expresses how plausible is the interpretation u . Clearly, π_K computed by (15.3) for $K = \{(c_\alpha, n(\alpha_{i-1}))\}, i = 1, m\}$ is equal to μ_C .

Example 3 Let $K = \{(c_1, 1), (c_2, \rho_2), (c_3, \rho_3)\}$, with $1 > \rho_2 > \rho_3$. The semantics of K obtained by applying (15.1) can be put under the form

$$\pi_K(u) = \min(\mu_{C_1}(u), \max(\mu_{C_2}(u), n(\rho_2)), \max(\mu_{C_3}(u), n(\rho_3))). \quad (15.5)$$

It is a weighted min aggregation which reflects the idea that we are completely happy if C_1 , C_2 and C_3 ($\pi_K(u) = 1$) are completely satisfied, we are less happy ($\pi_K(u) = n(\rho_3)$) if C_1 and C_2 only are satisfied, and we are even less happy if only C_1 ($\pi_K(u) = n(\rho_2)$) is satisfied.

Expressions (15.3) and (15.5) correspond to a conjunctive normal form (i.e., it is a min of max). It can be turned into a disjunctive normal form (max of min) and then provide a description of the different classes of candidates ranked according to their level of preference, as seen in the example below (all the candidates in a class reach the same level of satisfaction).

Example 4 Let us consider the following three criteria-based evaluation: "if u satisfies A and B , then u is completely satisfactory, and if A is not satisfied, then solutions should at least satisfy C ". Such an evaluation function can be encountered in multiple criteria problems for handling "special" cases (here situations where A is not satisfied) which coexist with a normal case (here situation where both A and B can be satisfied). It can be directly represented by the disjunctive form:

$$\mu_D(u) = \max(\min(\mu_A(u), \mu_B(u)), \min(\mu_C(u), n(\mu_A(u)), n(\rho)))$$

with $\rho < 1$ and μ_A, μ_B, μ_C are the characteristic functions of ordinary subsets. The reading of this expression is easy. Either the candidate satisfies both A and B , or if it falsifies A , it satisfies C , which is less satisfactory. This expression of μ_D obtained as the weighted union of the different classes of more or less acceptable solutions is a max of min-terms and can be transformed into an equivalent conjunctive form like (15.3) which is a min of max-terms. It can be checked that this conjunctive form corresponds to the base $K = \{(a \vee c, 1), (\neg a \vee b, 1), (a, \rho)\}$, applying (15.3), where A, B, C are the sets of models of a, b and c respectively. Indeed, according to possibilistic semantics, the models of $a \wedge b$ receive the degree of possibility 1 since they violate no formula in K , those of $\neg a \wedge c$ receive the degree $n(\rho)$ since they violate (a, ρ) , and the other interpretations which are models of $(\neg b \wedge a) \vee (\neg a \wedge \neg c)$ receive degree 0, since they violate $(a \vee c, 1)$ or $(\neg a \vee b, 1)$. Note that using the possibilistic resolution rule² K entails $(b \vee c, 1)$ and (b, ρ) as expected. K provides a logical, equivalent description of the evaluation process in terms of prioritized requirements to be satisfied by acceptable solutions.

Example 4 points out that the clausal form corresponding to the possibilistic logic base K may be sometimes less natural for expressing the goals than the normal disjunctive form μ_D which directly provides a reading of the levels of

satisfaction reached according to the candidate u which is chosen. Example 3 (corresponding to equation 15.5) illustrates the converse situation since K is then easy to read. Generally speaking, the normal disjunctive form provides a logical description of the different subsets of solutions each with its level of acceptability. On the contrary, a possibilistic logic base (which can always be put under the form of a conjunction of possibilistic clauses) corresponds to a prioritized set of goals as explained in 4.1.

Discounting and thresholding, specified below, are two elementary operations that can be performed on a preference profile. Indeed the discounting of a preference profile μ_C , associated with a criterion C by a level of importance $w \in L$ in a qualitative setting amounts to modifying $\mu_C(u)$ into $\max(\mu_C(u), n(w))$ for each u . It expresses that even if the candidate u is not at all satisfactory w.r.t. the initial criterion C ($\mu_C(u) = 0$), the candidate is no longer completely rejected w.r.t. the discounted criterion, and receives a value which is all the greater as the level of importance w is smaller, i.e., as the discounting is stronger. Thresholding a preference profile by a threshold value $\theta \in L$ in a qualitative setting amounts to modifying $\mu_C(u)$ into $\theta \rightarrow \mu_C(u) = 1$ if $\mu_C(u) \geq \theta$ and $\theta \rightarrow \mu_C(u) = \mu_C(u)$ if $\mu_C(u) < \theta$. In other words, as soon as the candidate u reaches the satisfaction level θ w.r.t. μ_C , it is regarded as fully satisfactory w.r.t. to the thresholded criterion, otherwise the satisfaction level remains unchanged.

These two operations are easy to perform on the representation in terms of prioritized goals. Indeed,

- the importance weighting operation $\max(\mu_C(u), n(w))$ translates into the suppression of the most prioritary goals $(c_{\alpha_i}, n(\alpha_{i-1}))$ such that $n(\alpha_{i-1}) > w$. When $w = 1$ no modification occurs, while when $w = 0$ all the goals disappear.
- the thresholding operation defined by $\theta \rightarrow \mu_C(u)$ translates into the suppression of the least prioritary goals $(c_{\alpha_i}, n(\alpha_{i-1}))$ such that $\alpha_i > \theta$. As in the previous case, if $\theta = 1$ no modification occurs, while when $\theta = 0$, all the goals disappear.

4.2 OPERATIONS ON PRIORITIZED SETS OF GOALS

More generally, the conjunctive aggregation of fuzzy preference profiles (which may be discounted or thresholded) can be interpreted in terms of conjunctions of crisp goals having different levels of priority, thus providing an expression of preferences in a possibilistic logic form. The pointwise aggregation of two fuzzy preference profiles C_1 and C_2 defined by means of the min operation can be easily interpreted in the prioritized goals framework. It corresponds to the union of the two sets of possibilistic logic formulas $\{(c_{1\alpha_i}, n(\alpha_{i-1}))\}$ and $\{(c_{2\alpha_j}, n(\alpha_{j-1}))\}$. This is a particular case of the syntactic fusion of possibilistic pieces of information (Benferhat et al., 1997a).

Aggregation operations other than min can be also accommodated in a syntactic manner. Indeed reinforcement and compensation operators, such as

the product and the average respectively, can also be interpreted in terms of operations on prioritized goals. Let (a, α) and (b, β) be two crisp constraints with priorities α and β belonging to L and $*$ be any binary monotonically increasing aggregation operation such as $1 * 1 = 1$. The aggregation of the possibilistic logic formulas (a, α) and (b, β) is expressed pointwisely at the semantical level by:

$$\max(\mu_A(u), n(\alpha)) * \max(\mu_B(u), n(\beta))$$

which in turn can be easily retranslated in terms of prioritized goals. As it can be shown (Benferhat et al., 1997a; Benferhat et al., 1999), this aggregation symbolically denoted by $(a, \alpha) * (b, \beta)$ is equivalent to the (min) conjunction of the three prioritized goals:

$$(a, n(n(\alpha) * 1)), (b, n(1 * n(\beta))), (a \vee b, n(n(\alpha) * n(\beta))).$$

Note that the combination amounts to adding the goal $a \vee b$ with a level of priority higher than the ones of a and b . Indeed, provided that $*$ is an increasing operation, $n(n(\alpha) * n(\beta))$ is greater or equal to $n(n(\alpha) * 1)$ and $n(1 * n(\beta))$. If $* = \min$, the third prioritized clause above is redundant³ w.r.t. the two others.

This can be generalized to preference profiles modeled by fuzzy sets A and B . It can be shown (Benferhat et al., 1997a) that $A * B$ defined pointwisely by $\mu_A * \mu_B$ is semantically equivalent to the conjunction of the following sets of prioritized goals:

$$\{(a_{\alpha_i} \vee b_{\beta_k}, n(\alpha_{i-1} * \beta_{k-1})) \text{ for all } (i, k)\}, \\ \{(a_{\alpha_i}, n(\alpha_{i-1} * 1)) \text{ for all } i\} \text{ and } \{(b_{\beta_k}, n(1 * \beta_{k-1})) \text{ for all } k\}.$$

Example 5 (continued) The goal base $K = \{(a \vee c, 1), (\neg a \vee b, 1), (a, \rho)\}$ can be retrieved by combination of its syntactic components $\{(a, 1), (b, 1)\}$ and $\{(c, 1), (\neg a, 1), (\perp, \rho)\}$ for $* = \max$ (where (\perp, ρ) is an unreachable goal expressing discounting).

It should be emphasized that the translation of aggregation $*$ into a possibilistic propositional logic base is done at the expense of the introduction of new levels in the scale. Indeed operation $*$ is not closed on the finite scale $L = \{\alpha_0 = 0 < \alpha_1 < \dots < \alpha_m = 1\}$ generally. For instance, if $*$ is the product and L is the numerical set $\{0, .1, .2, \dots, .9, 1\}$ then $.8 * .9 = .72$ does not belong to L . Moreover, note that the symmetry of $*$ is not required.

4.3 SPECIFYING PREFERENCES BY MEANS OF DEFAULT-LIKE CONSTRAINTS

The possibilistic framework can be also useful in qualitative preference profile elicitation from a set of constraints specifying them in a granular way. For

³In possibilistic logic (a, α) subsumes (a, β) when $\beta \leq \alpha$.

instance, a preference in favor of a binary property q w.r.t. $\neg q$ can be expressed by a constraint of the form

$$\Pi(q) > \Pi(\neg q) \quad (15.6)$$

which is equivalent to say that there exists at least one decision value in the set of models of q which is better than all the decision values in the set of models of $\neg q$. This is rather a weak manner for expressing the preference about q . Indeed, due to the definition of a possibility measure, (15.6) expresses that the most satisfactory candidate satisfying q , is preferred to the most satisfactory candidate (hence to all candidates), not satisfying q .

Such a constraint can be easily made context dependent: the requirement that if p is satisfied, q is preferred to $\neg q$, can be expressed by the constraint

$$\Pi(p \wedge q) > \Pi(p \wedge \neg q). \quad (15.7)$$

More generally, a collection of such requirements gives birth to possibilistic constraints, whose greatest solution π^* (in the sense that $\pi^* \geq \pi$ for any solution π) can be computed and represents a preference profile agreeing with the requirements. The minimal specificity principle expresses that any candidate is satisfactory inasmuch it complies with the constraints. It embodies the following default preference assignment: any state of affairs that has not been considered as explicitly bad by a decision-maker is regarded as good. However, there may exist other selection procedures worth-considering of a particular possibility distribution satisfying the set of constraints; this is open to discussion.

This approach is formally the same as the possibilistic treatment of default rules. Indeed, a default rule "if p then generally q " is translated into the constraint $\Pi(p \wedge q) > \Pi(p \wedge \neg q)$ which expresses that p true is strictly more plausible than p true and q false as recalled in Section 3.3. A set of consistent default rules of the form "if p_i then generally q_i " is thus represented by a set of constraints like (15.5) which implicitly defines a set of possibility measures. The greatest solution Π^* of this set always exists. Then, applying the minimal specificity principle (which here amounts to keeping the level of normality for each possible state of the world as great as permitted by the available knowledge), it induces a plausibility ordering on the interpretations encoded by the associated possibility distribution π^* . This ordering can be encoded at the formula level by constraints of the form $N^*(\neg p \vee q) \geq \alpha$ i.e., by a possibilistic logic base, where N^* is the dual measure associated with Π^* . See (Benferhat et al., 1992) (Benferhat et al., 1997b) for details.

In Example 3, the set of prioritized goals C_1, C_2, C_3 was directly given. However, priority orderings can be obtained from the possibility distribution selected from a set of constraints of the forms (15.6)-(15.7). For instance, an agent expresses that he would like coffee, and if coffee is not available he would like tea. This corresponds to the possibilistic base $K = \{(c \vee t, 1), (c, \alpha)\}$ with $\alpha < 1$. K can be retrieved as the least specific solution of the two following constraints $\Pi(c) > \Pi(\neg c)$ and $\Pi(\neg c \wedge t) > \Pi(\neg c \wedge \neg t)$, where $c = \text{coffee}$ and $t = \text{tea}$. The possibility distribution underlying the possibility measure Π can

be used for describing the set of alternatives with their level of satisfaction: *coffee* with $\Pi(c) = 1$; *tea* (and no *coffee*) with $\Pi(\neg c \wedge t) = n(\alpha)$; no *tea* (and no *coffee*) with $\Pi(\neg c \wedge \neg t) = 0$. This simple example shows the interrelationships between the three representations which play important roles in preference representation and elicitation, namely prioritized goals, preference constraints, and sets of alternatives with their levels of satisfaction.

Other types of constraints can be introduced, for instance for expressing indifference between q and $\neg q$ in context p as $\Pi(p \wedge q) = \Pi(p \wedge \neg q)$, or for expressing forms of independence as in (Dubois et al., 1997b). Besides, a stronger counterpart of (15.6) is:

$$\Delta(q) > \Pi(\neg q) \quad (15.8)$$

where $\Delta(q) = \min_{u: u \text{ model of } p} \pi(u)$ is the guaranteed possibility function (Dubois and Prade, 1998). (15.8) expresses that any candidate satisfying q is preferred to any candidate satisfying $\neg q$. This is stronger than the *ceteris paribus* principle which amounts to assessing that q is preferred to $\neg q$ in any context. See (Boutilier et al., 1997) for a constraint-based approach to preference modeling, although not expressed in the possibility theory framework.

4.4 LOGICAL HANDLING OF QUALITATIVE DECISION UNDER UNCERTAINTY

In decision under uncertainty, possibilistic logic can be used for modeling the available information about the world on the one hand, and the preferences on the other hand. This section summarizes a recent proposal by (Dubois et al., 1997c) (Dubois et al., 1998d) for designing a logical decision machinery. We distinguish between two possibilistic logic bases. The first one $K = \{(p_j, \rho_j); j = 1, k\}$ represents the available knowledge about the world. Namely (p_j, ρ_j) encodes that the piece of knowledge " p_j is true" holds as certain at least at level ρ_j where ρ_j belongs to a linearly ordered valuation set L . The second possibilistic logic base $P = \{(q_i, \lambda_i); i = 1, m\}$ represents the preferences of the decision-maker under the form of a prioritized set of goals. λ_i is the level of priority for getting goal q_i satisfied.

The propositional language contains Boolean decision variables and Boolean state variables. In this setting a decision is a conjunction of decision literals denoted d . Each potential decision d is represented by a formula $(d, 1)$ to be added to K if the decision is chosen. Let $K_d = K \cup \{(d, 1)\}$ be the description of what is known about the world when d is applied. Associated with K_d is the possibility distribution π_{K_d} defined using (15.3), which rank-orders the more or less plausible states of the world when d is chosen.

Associated with the layered set of goals P is the ordinal utility function

$$\mu_P(u) = \min_{i=1, m} \max(v_u(q_i), n(\lambda_j)), \quad (15.9)$$

which rank-orders the different states according to their acceptability. From π_{K_d} and μ_P the pessimistic qualitative utility (15.1) can be computed

$$E_*(d) = \min_u \max(\mu_P(u), n(\pi_{K_d}(u))). \quad (15.10)$$

$E_*(d)$ is all the greater as all the plausible states u according to π_{K_d} are among the most preferred states according to μ_P .

It is possible to compute $E_*(d)$ by only using a classical logic machinery on α -level cuts of K_d and P . Indeed it has been shown in (Dubois et al., 1997c) that $E_*(d)$ can be computed as the maximal value of the α such that

$$(K_d)_\alpha \text{ entails } (P)_{n(\alpha)} \quad (15.11)$$

where $(B)_\alpha$, resp. $(B)_{\underline{\alpha}}$ is the set of classical propositions in a possibilistic logic base B with level greater or equal to α , resp. strictly greater than α . As seen in (15.11), $E_*(d)$ is equal to 1 ($\alpha = 1$) if the completely certain part of K_d entails the satisfaction of all the goals, even the ones with low priorities, since P_0 is just the set of all the propositions in P with a non-zero priority level. In (Dubois et al., 1999) a computation procedure using an Assumption-based Truth Maintenance System is given for computing the best decision in the sense of (15.10)-(15.11).

The optimistic qualitative criterion (15.2) is given by

$$E^*(d) = \max_u \min(\mu_P(u), \pi_{K_d}(u)). \quad (15.12)$$

In logical terms, $E^*(d)$ has been shown to be equal to the greatest α such that $(K_d)_{n(\alpha)}$ and $(P)_{n(\alpha)}$ are logically consistent together. $E^*(d)$ is equal to 1 as soon as one fully acceptable choice u (i.e., such that $\mu_P(u) = 1$) is also completely plausible. The treatment of Example 1.1 using (15.11) and its optimistic counterpart can be found in (Dubois et al., 1999) where an ATMS-based computation procedure is presented.

5 CONCLUDING REMARKS

Bridging the gap between logic and decision is of interest for several types of reasons. First, at the methodological level, there is a tradition in decision analysis for normative approaches, which is not as much developed for reasoning problems, if we except (Kraus et al., 1990)' approach to exception-tolerant reasoning. Interestingly enough, we have been able to connect the Savage-like decision theoretic approach of Section 3, with this normative framework for plausible reasoning.

Second, logical expressions of the available knowledge, of the preferences may be felt more natural. The possibilistic framework makes it possible to reason about preferences, or even to revise them dynamically when new goals appear (see (Bensherbat et al., 1999) for a preliminary discussion). Because of the logical framework, it would be also possible to develop inference systems for providing explanations of the decisions proposed to the user.

The paper has also advocated the particular role that possibility theory and possibilistic logic can play for providing qualitative tools for decision under

uncertainty and preference modeling. In that respect, the cognitive validation of possibility theory is also an important issue which has received preliminary investigation (Raufaste and Da Silva Neves, 1998).

Appendix: Qualitative Possibilities Orderings

Let S be a set of states. A , B and C are subsets of S , called events. A possibility measure Π is mainly characterized by its max-decomposability under disjunction: $\Pi(A \cup B) = \max(\Pi(A), \Pi(B))$. Π is assumed to take the maximal value on the scale for tautologies. A dual measure of necessity N is associated to Π , i.e., $N(A) = n(\Pi(\bar{A}))$ where n is an order-reversing map of the scale on which Π takes its values (i.e., n is a one-to-one decreasing function which is involutive: $n(n(x)) = x$). N has the characteristic property $N(A \cap B) = \min(N(A), N(B))$. Possibility and necessity measures are the only graded counterparts of qualitative possibility and necessity relations (see e.g., Dubois and Prade, 1998).

A qualitative possibility ordering \geq_{Π} satisfies the following properties

1. \geq_{Π} is complete and transitive,
2. $S >_{\Pi} \emptyset$,
3. $\forall A, A \geq_{\Pi} \emptyset$,
4. $B \geq_{\Pi} C \Rightarrow A \cup B \geq_{\Pi} A \cup C$.

$B \geq_{\Pi} C$ reads B is at least as possible (i.e., plausible) as C . Generally, relation \geq_P is a weak order unless there is only one non null state.

Possibility and necessity orderings are dual in the sense that for a given possibility ordering \geq_{Π} the relation defined by $A \geq_N B$ iff $\bar{B} \geq_{\Pi} \bar{A}$ is a necessity ordering, and conversely. Thus, \geq_N is a necessity ordering iff

1. \geq_N is complete and transitive,
2. $S >_N \emptyset$,
3. $\forall A, S \geq_N A$,
4. $B \geq_N C \Rightarrow A \cap B \geq_N A \cap C$.

where $B \geq_N C$ reads “ B is at least as necessary as C ”.

Necessity orderings are also epistemic entrenchments in the sense of (Gärdenfors, 1988) (see (Dubois and Prade, 1991)), and are dual of possibility orderings first introduced by (Lewis, 1973). $B \geq_N C$ is intuitively understood as “ B is at least as certain as C ”, in the sense that arguments supporting B are at least as compelling as the arguments supporting C . The dual ordering $B \geq_{\Pi} C$ means that B is at least as plausible as C where plausibility refers to consistency with an agent’s knowledge: B is plausible when it does not contradict the agent’s beliefs. Any necessity ordering on a finite set can be represented by a necessity measure on a linearly ordered scale L .

A discrimax likelihood relation $>_{\Pi L}$ on 2^S , which is different from, although closely related to the original necessity and possibility orderings modeled by N (and its dual Π) is defined by the partial ordering

$$A >_{\Pi L} B \text{ iff } \overline{B} \cap A >_{\Pi} B \cap \overline{A}.$$

$>_{\Pi L}$ is auto-dual and is a refinement of both possibility and necessity orderings. Note that only disjoint events are compared via $>_{\Pi}$. The name ‘discrimax’ comes from the fact that events A and B are discriminated without taking into account their common part $A \cap B$ in the possibility evaluation (which is maxitive); see (Dubois et al., 1998b) for details. Besides, this relation is closely related to the modeling of default knowledge by nonmonotonic inferences $A \mid\sim B$ (see Section 3.3), so that the likelihood $>_{\Pi L}$ coincides with the possibilistic one on such pairs of events.

References

- Bellman, R. and Zadeh, L. (1970). Decision making in a fuzzy environment. *Management Science*, 17:B141–B164.
- Benferhat, S., Dubois, D., and Prade, H. (1992). Representing default rules in possibilistic logic. In B. Nebel, C. Rich, W. S., editor, *Proc. 3rd Inter. Conf. on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 673–684. Morgan and Kaufmann.
- Benferhat, S., Dubois, D., and Prade, H. (1997a). From semantic to syntactic approaches to information combination in possibilistic logic. In Bouchon-Meunier, B., editor, *Aggregation and Fusion of Imperfect Information*, pages 141–161. Physica Verlag, Heidelberg.
- Benferhat, S., Dubois, D., and Prade, H. (1997b). Nonmonotonic reasoning, conditional objects and possibility theory. *Artificial Intelligence*, 92:259–276.
- Benferhat, S., Dubois, D., and Prade, H. (1999). Towards a possibilistic handling of preferences. In *Proc. 16th Int. Joint Conf. on Artif. Intellig. (IJCAI-99)*, pages 1370–1375, Stockholm, Sweden.
- Bonet, B. and Geffner, H. (1996). Arguing for decisions: A qualitative model of decision making. In E. Horwitz, F. J. e., editor, *Proc. 12th Con. on Uncertainty in Artificial Intelligence (UAI'96)*, pages 98–105, Portland, Oregon.
- Boutilier, C. (1994). Toward a logic for qualitative decision theory. In J. Doyle, E. Sandewall, P. T. e., editor, *Proc. 4th Inter. Conf. on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 75–86, Bonn, Allemagne.
- Boutilier, C., Brafman, R., Geib, C., and Poole, D. (1997). A constraint-based approach to preference elicitation and decision making. In *Working Notes of the AAAI'97 Spring Symp. Series on Qualitative Preferences in Deliberation and Practical Reasoning*, pages 19–28, Stanford, CA.
- Doyle, J. and Wellman, M. P. (1991). Impediments to universal preference-based default theories. *Artificial Intelligence*, 49:97–128.
- Dubois, D., Fargier, H., and Prade, H. (1995). Fuzzy constraints in job-shop scheduling. *J. of Intelligent Manufacturing*, 64:215–234.

- Dubois, D., Fargier, H., and Prade, H. (1997a). Decision-making under ordinal preferences and comparative uncertainty. In Geiger, D. and Shenoy, P., editors, *Proc. 13th Conf. on Uncertainty in Artificial Intelligence*, pages 157–164, San Francisco, CA. Morgan and Kaufmann.
- Dubois, D., Fargier, H., and Prade, H. (1998a). Choice under uncertainty with ordinal decision rules: A formal investigation. Technical Report 98-03-R, IRIT.
- Dubois, D., Fargier, H., and Prade, H. (1998b). Possibilistic likelihood relations. In *Proc. 7th Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-based Systems*, pages 1196–1203, Paris. Editions EDK.
- Dubois, D., Fariñas del Cerro, L., Herzog, A., and Prade, H. (1997b). Qualitative relevance and independence: a roadmap. In *Proc. 15th Int. Joint Conf. on Artif. Intellig. (IJCAI-97)*, pages 62–67, Nagoya, Japan. Morgan Kaufman, San Francisco, CA.
- Dubois, D., Fodor, J., Prade, H., and Roubens, M. (1996). Aggregation of decomposable measures with application to utility theory. *Theory and Decision*, 41:59–95.
- Dubois, D., Godo, L., Prade, H., and Zapico, A. (1998c). Making decision in a qualitative setting: from decision under uncertainty to case-based decision. In *Proc. 6th Inter. Conf. Principles of Knowledge Repres. and Reasoning (KR'98)*, pages 594–605. Morgan Kaufmann, San Francisco, CA.
- Dubois, D., Lang, J., and Prade, H. (1994). Possibilistic logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 3, pages 439–513. D.M. Gabbay et al. eds., Oxford University Press.
- Dubois, D., Le Berre, D., Prade, H., and Sabbadin, R. (1998d). Logical representation and computation of optimal decisions in a qualitative setting. In *Proc. 15th Nat. Conf. on Artif. Intellig. (AAAI-98)*, pages 588–593, Madison, WI. AAAI Press/MIT Press, Menlo Park, CA.
- Dubois, D., Le Berre, D., Prade, H., and Sabbadin, R. (1999). Using possibilistic logic for modeling qualitative decision: Atms-based algorithms. *Fundamenta Informaticae*, 37:1–30.
- Dubois, D. and Prade, H. (1991). Epistemic entrenchment and possibilistic logic. *Artificial Intelligence*, 50:223–239.
- Dubois, D. and Prade, H. (1995a). Conditional objects, possibility theory and default rules. In Crocco, G., del Cerro, L. F., and Herzog, A., editors, *Conditional: From Philosophy to Computer Science*, pages 311–346. Oxford University Press.
- Dubois, D. and Prade, H. (1995b). Possibility theory as a basis for qualitative decision theory. In *14th Inter. Joint Conf. on Artificial Intelligence (IJCAI'95)*, pages 1924–1930, Montréal. Morgan Kaufmann, San Mateo, CA.
- Dubois, D. and Prade, H. (1998). Possibility theory: qualitative and quantitative aspects. In Smets, P., editor, *Quantified Representations of Uncertainty and Imprecision*, volume Vol. 1 of the Handbook of Defeasible Reasoning and Uncertainty Management Systems (D. M. Gabbay, P. Smets, eds.), pages 169–226. Kluwer Acad.

- Dubois, D., Prade, H., and Sabbadin, R. (1997c). A possibilistic logic machinery for qualitative decision. In *Working Notes AAAI'97 Workshop on Qualitative Preferences in Deliberation and Practical Reasoning*, pages 47–54, Stanford. AAAI Press, Menlo Park.
- Dubois, D., Prade, H., and Sabbadin, R. (1998e). Decision-theoretic foundations of qualitative possibility theory. Invited talk, 16th Europ. Conf. on Operational Research, Brussels, Belgium. To appear in *European Journal of Operations Research*.
- Dubois, D., Prade, H., and Sabbadin, R. (1998f). Qualitative decision theory with sugeno integrals. In *Proc. 14th Conf Uncertainty in Artificial Intelligence*, pages 121–128. Morgan Kaufmann, San Francisco, CA.
- Fargier, H. and Perny, P. (1999). Qualitative models for decision under uncertainty without the commensurability assumption. In *Proc. 15th Uncertainty in Artificial Intelligence, Stockholm*, pages 188–195. Morgan Kaufmann, San Francisco, CA.
- Friedman, N. and Halpern, J. (1996). Plausibility measures and default reasoning. In *Proc. 13th National Conf. on Artificial Intelligence (AAAI'96)*, pages 1297–1304, Portland. To appear in *J. Assoc. for Comp. Mach.*
- Gärdenfors, P. (1988). *Knowledge in Flux*. MIT Press, Cambridge.
- Gärdenfors, P. and Makinson, D. (1994). Nonmonotonic inference based on expectations. *Artificial Intelligence*, 65:197–245.
- Guéré, E. and Alami, R. (1999). A possibilistic planner that deals with non-determinism and contingency. In *Proc. 16th Int. Joint Conf. on Artif. Intellig. (IJCAI-99)*, pages 996–1001, Stockholm, Sweden.
- Halpern, J. (1997). Defining relative likelihood in partially-ordered preferential structures. *Journal of Artificial Intelligence Research*, 7:1–24.
- Kraus, K., Lehmann, D., and Magidor, M. (1990). Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44:167–207.
- Lang, J. (1991). Possibilistic logic as a logical framework for min-max discrete optimisation problems and prioritized constraints. In Jorrand, P. and Kelemen, J., editors, *Fundamentals of Artificial Intelligence Research (FAIR'91)*, number 535 in L.N.C.S., pages 112–126. Springer Verlag.
- Lehmann, D. and Magidor, M. (1992). What does a conditional knowledge base entail? *Artificial Intelligence*, 55:1–60.
- Lewis, D. (1973). *Counterfactuals*. Basil Blackwell, London.
- Moulin, H. (1988). *Axioms of Cooperative Decision Making*. Wiley, New York.
- Pearl, J. (1990). System Z: A natural ordering of defaults with tractable applications to default reasonning. In *Proc. 3rd Conf. on Theoretical Aspects of Reasoning about Knowledge (TARK'90)*, pages 121–135. Morgan Kaufmann.
- Pearl, J. (1993). From conditionnal oughts to qualitative decision theory. In Heckerman, D. and Mamdani, A., editors, *Proc. of the 9th Conf. on Uncertainty in Artificial Intelligence (UAI'93)*, pages 12–20. Morgan Kaufmann, San Mateo, CA.
- Raufaste, E. and Da Silva Neves, R. (1998). Empirical evaluation of possibility theory in human radiological diagnosis. In Prade, H., editor, *Proc. 13th*

- European Conference on Artificial Intelligence (ECAI-98)*, pages 125–128, Brighton, U.K. Wiley, Chichester.
- Sabbadin, R., Fargier, H., and Lang, J. (1998). Towards qualitative approaches to multi-stage decision making. *Inter. J. Approx. Reas.*, 19:441–471.
- Savage, L. (1954). *The Foundations of Statistics*. Dover, New York. Reprinted by Dover, 1972.
- Tan, S. W. and Pearl, J. (1994). Qualitative decision theory. In *Proc. 11th Nat. Conf. on Artificial Intelligence (AAAI'94)*, pages 928–933, Seattle, WA.
- von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press.
- Wald, A. (1950). *Statistical Decision Functions*. Wiley and Sons, New York.
- Zadeh, L. (1978). Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28.
- Zapico, A. (1999). Axiomatic foundations for qualitative/ordinal decisions with partial preferences. In *Proc. 16th Int. Joint Conf. on Artif. Intell. (IJCAI-99)*, pages 132–137, Stockholm, Sweden. Morgan Kaufmann, San Francisco, CA.

XI

LOGIC AND BELIEFS

Chapter 16

THE ROLE(S) OF BELIEF IN AI

Don Perlin*

University of Maryland

Institute for Advanced Computer Studies

and Department of Computer Science

College Park MD 20742

www.cs.umd.edu/users/perlin

perlin@cs.umd.edu

Abstract Beliefs play complex and sometimes confusing roles in AI. This paper surveys (i) a variety of notions of belief, (ii) formal efforts to characterize beliefs, and (iii) how beliefs are related to action, to language, and to commonsense. In addition, we will consider certain logical tensions between beliefs and consistency.

Keywords: Belief, consistency, modality, content

1 INTRODUCTION

Already by 1980 many researchers in AI were hard at work on the problem of belief, and philosophers had been at work on it even before that. But then (Nilsson, 1983) introduced the idea of robots with a lifetime of their own. This notion, which we now refer to more generally as autonomous intelligent agents, makes the role of belief central, for such an agent will have to behave in ways only explainable in terms involving belief, as will be argued below.

To clarify one point at the outset: we use the term “belief” here *not* in reference to a mere suspicion or impression that the item believed is possibly true, as in “I believe so”; rather it is in the much stronger sense: what the believing agent takes to be true, as if it were possible to look inside the agent’s “head” and see what the agent’s view of the world is. And of course, with artificial agents, it is so possible; but this leaves open the problem of which items found there are to count as beliefs (or as views of the world).

*The author would like to express thanks for support for this research, from grants funded by AFOSR and ONR.

This paper is organized as follows: Section 2 describes and contrasts a variety of notions of belief in AI; Section 3 discusses why beliefs are essential to the AI enterprise; Section 4 examines the extent to which beliefs are formal (or inference-bound), and Section 5 considers certain tensions between beliefs and consistency.

It may be that someday we will succeed in building an artificial autonomous agent that, by all routine standards, is intelligent. And yet if we do not understand how that agent works, if we cannot explain its behavior in a way that sheds light on the nature of intelligence, then we shall have succeeded as engineers but not as scientists.¹

One key component of the scientific enterprise of artificial intelligence is the understanding of the nature of beliefs. This paper discusses that component.

2 WHAT ARE BELIEFS?

What constitutes a belief is controversial. A number of distinct characterizations have been offered, serving best perhaps to underscore the notion that this common term of everyday usage masks a number of highly distinct phenomena. Early on there appears to have been a conflation of belief, knowledge, and data, as in the expressions "knowledge base" and "belief base" used in reference to a "database" of "facts". In addition, much early work identified an agent's beliefs with inferential consequences of its database.²

From a programmer's or logician's point of view, a belief can be thought of simply as a piece of data that has a truth value (it is true or false), and that can in some form be available to an agent to conduct its reasoning, much like an axiom or theorem. But in more cognitive terms, an agent's belief base can be thought of as its view of what the world is like. These two encompass quite a range of differences, and each of them admits of further distinctions as well. For instance, axioms are sentences in a formal language; yet beliefs are sometimes taken by the logic community to be abstract propositions rather than sentences. This will be discussed below in terms of modal versus sentential representations.

Partly in an effort to lend some clarity to this confusion, (Levesque, 1984) has emphasized the difference between implicit and explicit belief: the former includes all consequences of the agent's database, and the latter only those that the agent has in fact obtained. The latter of course can (and usually does) change over time.³

¹It is said that Henry Bessemer, who invented the Bessemer process for making steel, had no principled understanding of how his process worked, but that he tried many approaches until one finally did the job. Thus the above remark should in no way be read as disparaging of engineering; it is simply a matter of differing aims.

²(Gettier, 1963) has shown that knowledge is a very tricky notion, and many have simply given up on knowledge as a useful notion at all, in favor of belief. However, belief is not without its own pitfalls as we shall see.

³There is also (see Section 5) a notion of *evolving* explicit belief (formalized in terms of so-called active logics) which characterizes a reasoning process by which beliefs can be formed, refined, and rejected.

Levesque's *implicit* version is often stated in terms of the property of *logical omniscience*: such beliefs are closed under logical consequence. Thus if α and $\alpha \rightarrow \beta$ are agent beliefs, so is β , even if the agent in question has not in fact actually drawn that conclusion. One says the agent implicitly believes β . See (Fagin et al., 1995) for a detailed recent study of this approach.

(Konolige, 1986) has described in detail how an actual agent might construct beliefs from one beliefs in a *deductive* process akin to Levesque's explicit beliefs, forming thereby the set of all such inferred beliefs. For Konolige this process may be highly constrained due to resource limitations. On this approach it is possible to believe α and $\alpha \rightarrow \beta$ and yet fail to deduce β , thereby also failing to believe β . On the other hand, Konolige's treatment does require the belief set to be closed under all inferences available to the agent, a limited form of omniscience: the agent believes all it ever comes to believe. One can also consider a (usually) larger class, Deduct*, of those beliefs that would arise from given inferential processes without any resource bounds.⁴

It may be the case that explicit beliefs coincide with deductive beliefs; this is so if we consider the believing agent only at the end of its reasoning efforts, when it has come to believe all it will believe. And it may be that implicit beliefs coincide with the above class Deduct*; this is so if the inference rules are complete. However, all the above notions of belief presuppose some starting set of beliefs, and from those a wider set of consequent beliefs may be characterized.

Another view is that encodings of statements of purported fact, whether in a database or not, cannot be properly viewed as beliefs unless they are connected suitably to the agent's behavior. Agents can spin conclusions out of propositions as mere exercises, just as we can conclude from A and from A implies B, that B, without our believing A, B, or A implies B. Somehow a belief seems tied to our overall behavior and not just to our ability to spin conclusions: we need to be able to believe the premises, as well as the conclusion. This however is no definition, for it is circular. Even putting A in a database is little help, for one can easily have a list of disbelieved propositions or sentences (e.g., the Moon is made of green cheese, babies are brought by storks, etc) all

⁴There is however a curious phenomenon about resource-bounded belief: the very fact of boundedness can lead to beliefs that would not arise in the absence of those bounds. This is most notably so when there are reflective inference rules, i.e., ones that produce inferred beliefs reflecting past behavior including absences of certain potential beliefs. For instance, if *NotFlies(tweety)* is a logical consequence of given axioms but is not derived due to time or space limitations, $\neg K\text{now } \neg NotFlies(tweety)$ may be inferred. An unbounded (ideal) reasoner would not reach this conclusion but rather *NotFlies(tweety)* instead. A more dramatic case is that of inconsistent beliefs, which in the case of a resource-bounded reasoner may be limited in the damage they cause, but an unbounded reasoner will produce the usual disaster of *ex contradictione quodlibet*: everything follows from a contradiction. Finally, a temporal resource boundedness can, in the presence of suitable reflective rules, lead to conclusions such as *Now(i)* and even *Contra(i, P, $\neg P$)* to note the time is now i and that a contradiction has been discovered at that time. Unbounded reasoning as it is usually understood in AI does not take account of the passage of time during reasoning and thus has no way to track this evolving character of beliefs over time. See (Elgot-Drapkin and Perlin, 1990; Elgot-Drapkin et al., 1993).

kept in a database. Nor is it enough that such a database be operated on by inferences, as we just saw.

This suggests a possible "dispositional" definition as follows (Perlis, 1986): an expression E occurring in an agent's database(s) is a belief if the agent is disposed to act on E .⁵ On this view of beliefs, which I call *use-beliefs*, actions are highly dependent on beliefs. I suspect this is the notion of belief that is most closely relevant to commonsense reasoning; reasons will be given below in Section 3.

This is distinct from but related to, another dispositional account: agent G believes E if G is disposed to *assent* to E if asked "Do you believe E ?," or "Is E true?" We may then say G has assent-belief E .

Thus we have seen at least four distinct notions of belief:

- Entailment-belief: this is Levesque's notion of implicit belief: those wffs (or propositions) that are entailed by given (starting) beliefs.
- Deductive-belief: Konolige's notion of belief as those wffs that are proven by the agent in question.
- Use-belief: the notion of a belief as an item (wff or proposition) the agent is disposed to use in making decisions as to what actions to take (in conjunction of course with whatever desires and preferences the agent may have).
- Assent-belief: the notion of a belief as an item the agent is disposed to assent to when asked.

The first two do not say what it is to be a belief, but rather what it is for a belief to arise from other beliefs. The latter two attempt to define belief in more primitive terms. It is unclear how we are to assess whether an agent believes any given item, on most of these notions (assent belief may be an exception); and of course these distinct notions do not agree among themselves in general.⁶

The entailment-belief notion above leads to elegant mathematical properties, but seems of little practical relevance for physically realizable agents. Unfortunately it tends to be forced by most modal treatments which take the items of belief to be abstract propositions of some kind, rather than patterns such as sentences that can be stored in a physical medium like a computer database or a brain. This latter dichotomy can be put differently: are beliefs syntactic objects (such as sentences) in a representational format, or are they more abstract "meanings"? In modal logic beliefs are objects of modalities, i.e., of operators that apply to abstract propositions. For instance one may write $\text{Bel } P$ where P is not an argument to a predicate Bel ; here instead Bel

⁵To be disposed to do X is to have the tendency to do X , in the absence of exceptional circumstances; i.e., X is the default action.

⁶A referee has pointed out the following helpful example to distinguish the latter two: a native Chinese speaker who believes that it is raining may not assent to "Is it raining?" due to not knowing English; but he will have be able to use the belief in order to decide to take an umbrella. This is related to a puzzle discussed later under the heading of abstract content.

is a kind of modifier to P similar in spirit to a connective or negation symbol, telling us in what "mode" or "attitude" to take P , i.e., Bel tells us to take it not as true or false or possible or likely or hoped-for, but as believed (by whatever agent is understood). P itself is whatever the sentence " P " represents, which is usually taken to be an abstraction, such as the set of all wffs with the same truth value as the sentence " P ".

The above modal approach, due initially to (Hintikka, 1962), is in stark contradistinction to the "sentential" view, namely that beliefs simply are sentences (in some special relation to the agent that believes them, of course, whether assentual or implicit or deductive or use). As AI moves closer to Nilsson's notion of "robots with a lifetime of their own," we will need to be much clearer about such things. As pointed out above, the mere fact of having inferred something in no way leads to willingness to use that conclusion to make and carry out plans. At the very least this depends on how we regard the premises of the inference. So when one robot reasons about another robot's beliefs, something will be occurring that we do not as yet understand very well, and that therefore we are ill-prepared to design. Is the second robot viewed by the first robot as simply having inferred various conclusions? Or as taking a more active stand toward those conclusions? Moreover, two agents (people, robots) are unlikely to use identical representations for their beliefs, so that if B has the belief b, and C comes to believe that B so believes, C's belief may well have to represent B's belief in terms C can understand, which may not be the way B represents it. This is in part the problem of belief-ascription (Kripke, 1979), about which more below.

Yet another – very radical – view is that beliefs are an illusion of naive "folk psychology", and no such phenomenon as believing actually occurs; nor indeed will any more precise counterpart arise to take the place of a currently ambiguous and poorly-formed notion of belief. This view is called "eliminative materialism"; (Churchland, 1984) has presented a detailed defense of this view.

3 WHY ARE BELIEFS IMPORTANT?

Let us call the view that intelligence involves in an essential way the forming of beliefs – i.e., manipulable informational units that obey some sort of semantic strictures – the belief-theory. Much of the AI literature then at least tacitly assumes the belief-theory. An alternative view is that intelligent behavior can and does go on even in the absence of beliefs about the very world that the intelligent agent deals with. Thus neural networks are sometimes offered as examples of intelligent systems that can be made sense of without the belief-theory; this is the above view espoused by Churchland, for instance.

Yet I will argue now that the only account that has been given of intelligent behavior so far, and likely the only one possible, is based on the belief-theory, and in particular on use-belief as the most fundamental notion of belief.

Imagine an agent, Susie Q, who appears to be intelligent. She exhibits a range of behaviors highly sensitive to details of her surroundings, including details of interactions with other agents. In particular, she regularly plays tennis with a partner, Freddy, by advance arrangement with him.

Now, to explain why it is that Susie Q goes to the gymnasium on Tuesday evenings, one might suppose that she wants to play tennis, and that she believes she can do so at the gym, that today is Tuesday, that the gym is open Tuesday evenings, and that Freddy will be there. Moreover, one supposes that were she to be told (a) the gymnasium is closed for repairs, she will not go there; and that if she were told (b) of a new policy that gymnasium members must now bring their own towels, she will either bring a towel or will not go at all; and so on.⁷

No account of these and similar behavioral patterns has ever been given, that does not rely profoundly on the formation and use of beliefs on the part of the agents in question.⁸

One cannot escape this by appealing to a distributed representation such as a neural network: a neural net model may well account for Susie's recognizing (or even learning to recognize) the gym when it is in front of her, or for that matter recognizing when she is in need of exercise, and even that it is Tuesday. But it will not account for her ability to rapidly shift her behavior on being told either of the new statements (a) and (b) above. These latter require her to process her recognitions along with (a) or (b) in ways that are deeply sensitive to details of just the sort that beliefs are thought to have: propositional content – they can be true, false, composed of simpler beliefs, refashioned, and so on; i.e., they have an essentially symbolic or language-like character. While it is perfectly true that, in principle, a neural network or other distributed representation can be designed so as to provide this sensitivity, such a design amounts precisely to turning that representation into a belief-system! That is, it will have to process informational units according to semantic strictures such as modus ponens.

Thus it is not only the case that until someone comes up with a different and equally explanatory account of such behaviors, we are stuck with the belief-theory. The case is stronger: the behaviors we wish to account for, and to design into agents, are themselves ones of semantically-bound processing of informational units. The belief-theory appears to be true on the basis of the very nature of the problem. And therefore, if this is so, beliefs are fundamental to the AI enterprise; the entire field, one might say, amounts to the learning, inference, recall, use, communication, and manipulation of semantically bound informational units, whether in planning, scene interpretation, natural language processing, problem solving, navigation. In saying these are semantically bound, we are saying that they are sensitive in very particular ways. We turn to this below.

⁷Note that an essential ingredient in Susie's actions is that she *wants* something; it is for that reason that actual behaviors arise out of the reasoning. That is, it is not beliefs in inference rules alone, but these in conjunction with a pattern of desires and preferences, that leads to the observed behavior.

⁸There are actually two types of behavioral pattern in the above description: those behaviors ascribed to Susie, and the suppositional behaviors ascribed to us who think about Susie. Both involve belief.

4 WHY ARE BELIEFS (PARTLY) LOGICAL IN NATURE?

As mentioned above, beliefs have language-like character: they belong to a more general sort of entity that can be affirmed, denied, questioned, composed into more complex forms, and so on. Moreover, they can be used, as in the Susie Q example, to create new beliefs according to precise rules that are deeply sensitive to local formatting details, such as negations. Thus when told that the gym is not open today, she must process this as a kind of denial of her former belief that it is open. The “not” is more than a mere additional set of pixel data enriching an image; it has specific content that bears on the meaning (and hence the use) of the whole informational unit, saying in effect that the rest of that unit is not to be used.

But these are precisely what characterize a logic: precise rules of composition and formation (this is the “syntax” half) and semantics, where beliefs also do their part: as language-like entities they can have meanings, so that when Susie comes to believe that her regular gym is not open, her actions (staying home, or going to another gym) fit well into the external world. This is due to a good match between her beliefs and the world, and this match is given by the semantic content (meaning) of her beliefs. Of course, beliefs are often wrong, but this too depends on beliefs having semantic content.

Now, none of this is to say that the kinds of beliefs people (or autonomous agents in general) may come up with need follow some standard textbook inference rules. Not at all. Indeed a large part of theoretical AI is devoted to the design of new formalisms that come closer to the needs of real agents. And that is where many of the most interesting and important developments have been and, hopefully, will be: what sorts of belief-manipulations (inferences, putting beliefs together to come up with new beliefs) are appropriate for an autonomous agent? Non-monotonicity is one such development (see (Ginsberg, 1987)); probabilistic reasoning is another, recently shown to be related to nonmonotonic reasoning (see (Bacchus et al., 1993)).

5 WHY ARE BELIEFS PROBLEMATIC?

We have already seen that beliefs are complicated and controversial. But there is worse in store.

5.1 BELIEFS CAN BE INCONSISTENT

(Montague, 1963) showed that, in the case of knowledge, apparently mild assumptions about an agent’s knowledge *must* be inconsistent. This was quite a surprising result. The assumptions in question involve the agent’s knowledge of elementary arithmetic, as well as a few further conditions such as closure of the knowledge base under modus ponens. Roughly, this result exploits Gödel-like self-referential sentences that contradict themselves.

What Montague showed is that any first-order theory that includes (any reasonably expressive theory of) arithmetic (such as Robinson arithmetic) and having as theorems (*Know* α) \rightarrow α for each closed wff α , and also having as

the theorems $\text{Know } \alpha$ whenever α is a theorem, is inconsistent. (Here, and below, we suppress reifying quote marks on embedded wffs, writing $\text{Know } \alpha$ for $\text{Know } " \alpha "$, for ease of reading.)

Nevertheless, this may not be problematic in itself, since as noted earlier, knowledge may be a poor idea for AI. But (Thomason, 1980) found an equally surprising result for belief: he showed that under very similar assumptions, an agent's beliefs themselves will be inconsistent. Specifically, Thomason's result is this:

If an agent g believes (a suitable theory of) arithmetic and also g 's beliefs (given as arguments to the predicate Bel) satisfy the following conditions:

1. $\text{Bel } \alpha \rightarrow \text{Bel}(\text{Bel } \alpha)$
2. $\text{Bel}(\text{Bel } \alpha \rightarrow \alpha)$
3. $\text{Bel } \alpha$ for all valid α
4. $\text{Bel}(\alpha \rightarrow \beta) \rightarrow (\text{Bel } \alpha \rightarrow \text{Bel } \beta)$

then g is inconsistent in the sense that g will believe all wffs.

(Perlis, 1988) has provided yet another related conundrum about belief: the ability to determine whether or not one has a given belief leads, under a few general and plausible conditions, to an inconsistent belief set. Again the technical device employed is that of a self-referential sentence. Thus even a perfectly omniscient reasoner would not be able to have perfect self-knowledge of its own beliefs. It is very puzzling why this should be so; it appears straightforward that one could build up such an ideal reasoner in stages by adding first one belief, then another, to achieve a fixed point. But, as Kripke emphasized in (Kripke, 1975), beliefs can refer to one another in very complex ways, and the truth of one belief thus can depend on others not even yet formed (as in "I will never believe anything you say").

These results are often taken as reason to abandon a "sentential" treatment of belief, where beliefs are taken to be formulas or sentences stored in a database, and in favor instead of a modal or propositional treatment in which beliefs are abstractions rather than concrete representations.

That is, the above results apply most directly to so-called syntactic treatments of belief (or knowledge), where the formal representation of a belief is that of a quoted wff or term to which a predicate symbol (such as a belief-predicate) may be applied. Modal treatments in which beliefs are construed as propositions do not quite so easily succumb to these problems.

Moreover, (des Rivières and Levesque, 1986) showed that one can smuggle the seeming virtues of modal logic into FOL, thereby vitiating the impact of the Montague and Thomason results. They removed the offending (inconsistent) sentences from Montague-Thomason settings, by allowing only those that were first-order "transcriptions" of modal wffs. (Morreau and Kraus, 1998) recently extended this work to its apparent logical extreme. On the other hand, it is hard to see why an agent should be restricted to using a weaker language than it is clearly capable of.

It turns out, however, that if we allow manipulation of either beliefs or *representations* of beliefs, in ways that are quite readily available to any routinely programmed computational agent, then the above negative (inconsistency) results apply equally to sentential and to modal treatments (Perlis, 1988).

A possible lesson that can be taken from this is that omniscient agents are not possible, either in practice (this is clear, since such agents require infinite storage for infinitely many beliefs) or in theory (due to the above limitations); and that instead of the sum total of all possible inferences an agent might make over its lifetime, we should focus on the actual inferences it makes in the short term to support its ongoing behaviors. This is the view taken in the active logic approach (Elgot-Drapkin and Perlis, 1990) in which the agent's belief base is at all times an (evolving) finite set of sentences which moreover can be inconsistent, and such an inconsistency, if discovered by the agent, is itself recorded in the belief base and used to prompt a corrective response by means of special inference rules. That is, an active logic can reason about its own ongoing inference process and take steps to influence its future behavior based on what it sees itself doing.

Another response to the above negative results is based on (Perlis, 1985; Perlis, 1988), namely to slightly weaken the contradictory assumptions. There are two ideas here: (i) to acknowledge that intuitions about formalisms that allow for self-referential expressions (this is where the arithmetic comes in) can lead to unintuitive consequences and thus we should restrict our usual intuitions to "safe" cases; and (ii) a particular way to distinguish safe cases can be derived from work of (Gilmore, 1974; Kripke, 1975). (i) is also the basis for the method of (des Rivières and Levesque, 1986) but that approach takes a more limited view of what is "safe" (transcriptions of modal wffs) than that afforded by (ii). The latter approach replaces key instances of α by the variant α^* . The starred version of α is in most cases of interest simply α itself. But when α contains, say, the predicate symbol *Bel*, preceded by negation, as in $\neg Bel \beta$, then its star is $(\neg Bel \beta)^* = Bel(\neg \beta)$. This simple "trick" (Gilmore, 1974) is sufficient to reinstate consistency to all the above axiomatizations, and includes all the cases of (des Rivières and Levesque, 1986) as well.

5.2 BELIEFS MUST BE INCONSISTENT

Although several approaches to the inconsistent foundational axiomatizations have been found, as indicated above, there are other matters for concern about inconsistency, of a different nature. There are reasons to suspect that commonsense reasoning is of its nature inconsistent (Perlis, 1996), having to do largely with the unreliability of incoming data in a complex world. For one simple case consider the following: an agent receives two pieces of data: (i) John is tall, and (ii) Mr. Smith is short. Both are from generally reliable sources S_1 and S_2 , and so both (i) and (ii) are accepted as beliefs. But now the agent invokes its other beliefs, including that Mr. Smith and John are the same person, and a contradiction arises. This is not due to an error in reasoning, nor to a faulty knowledge-representation, but to a problem with incoming data. In

the present example either the incoming data contains an mistake (one of the sources was in error) or the meaning of one or more expressions was misinterpreted (e.g., the John in question is John Jones, not John Smith; or "tall" and "short" have quite different meanings to sources S_1 and S_2 .

There seems to be no formal way to enforce consistency, then, short of allowing the agent in question to interact only with other agents whose own belief bases have first been "sanitized" for mutual consistency, clearly an impossible requirement in general.

These considerations are closely related to the topic of belief revision. The latter on the face of it deals with the question of what to do when clashes are found between beliefs: which beliefs are to be revised, and how? But actual studies have so far concentrated mainly on a special case, namely given that an incoming piece of data is definitely to be accepted (what was called "recency prejudice" in (Perlis, 1996)) which older conflicting beliefs are to be sacrificed? The underlying issue was raised early in the nonmonotonic reasoning literature (Reiter, 1980b; McDermott and Doyle, 1980) but then largely ignored until the publication of (Alchourron et al., 1985). Even this, however, retained the principle of recency prejudice, so that the challenge of adjudicating between competing beliefs without advance knowledge that certain ones are to be retained, went largely unexplored.⁹

At the same time, a rather different approach to commonsense reasoning was being developed, that emphasized the evolving nature of reasoning; this was initially called "step logic" (Elgot-Drapkin and Perlis, 1990) and later "active logic" (Miller and Perlis, 1993; Gurney et al., 1997; Perlis et al., 1998; Traum and Andersen, 1999). In this approach, beliefs are viewed as coming and going as reasoning and action take place, and new beliefs must compete along with the old on equal footing, unless special circumstances dictate otherwise. Moreover, the encountering of a direct contradiction by an active logic system, say P and not-P, is dealt with by the same underlying process of evolving inference as any other reasoning; that is, there is no halt to the "normal" process of reasoning while a separate "revision" process recreates a totally consistent belief set. Instead, when a new belief not-P is formed that may directly contradict an older belief P, both are gradually assessed, and as long as the direct contradiction remains in the belief base, neither P nor not-P is used as the basis for further inference; and at the same time ordinary reasoning proceeds. Much remains to be done, but there is some hope that such a gradualist approach to inconsistency may be successful in various domains (Elgot-Drapkin et al., 1993).

5.3 ABSTRACT CONTENT

Let us return to Susie Q. If her friend Freddy usually goes to the gym at the same time as Susie, and if we tell her the gym is closed today, we can predict

⁹The issue is most salient when the incoming information is not immediately seen to be in conflict with existing beliefs, so that it can be accepted on a "generosity" default: information tends to be correct unless counter-evidence is at hand. But further thought may reveal a hidden conflict.

that she will try to tell Freddy. But what will she tell him? This we cannot predict in detail, only that she will somehow try to convey to him *that he should not try to use the gym today*. But she may do this in countless ways, including countless linguistic ways, e.g.:

- The gym is closed today
- Don't go to the gym today
- Do not try to use the gym today
- You can't use the gym today
- The gym is not open today
- Don't count on the gym today
- You'll have to go somewhere else to exercise today

What these have in common is her intention to warn him about the gym not being open, and their abstract shared content to that effect. It is that abstract content that, somehow, she wants to convey to him, and that we can predict she will so try. Moreover, she wants him to acquire that content as a belief so he can act on it too. It is not just that she wants him to avoid the gym; she wants him to have the actual belief about it so he can act upon it in as general a way as she. For instance, he can then tell other friends to avoid the gym even if he had no plans to go there himself.

How can we design a system to reason with abstract content? Or is it enough for it to reason with concrete syntactic representations (sentence-like entities) in a flexible way? And how flexible should that be? We would not be highly impressed with a robot that could tell Freddy "avoid the gym" but that would not refrain from telling him that even if another agent had just told him out loud seconds earlier "Don't go to the gym today." Our robot should understand that Freddy already has been informed of this and that "avoid the gym" would just be a useless repetition. That is, the ability to suitably relate sentences by similarity of content is part of the intelligent use of beliefs.

But we have no good theory of content at present. (Kripke, 1979) has in fact pointed out a specific puzzle about the content of beliefs. I will paraphrase one of his examples as follows: suppose Susie hears that the gym is closed today, but also hears her friend Freddy say "I'm going to play tennis at the club today", without her being aware that the club Freddy has in mind is in fact the gym where they both usually exercise. So she does not tell him the gym is closed. Now the puzzle is this: she seems to have come to hold two distinct beliefs, one with the content that a certain facility is open, and the other that it is not open; that is, she seems to have two contradictory beliefs about the same place: that it is open and that it is not open. If one adopts the "unique names hypothesis" (Reiter, 1980a) – that distinct names refer to distinct entities – then such a situation appears to be ruled out; however, there are various reasons not to make such an hypothesis: (i) it is highly implausible for a real agent, as this example shows; (ii) not only names but *descriptions*

are involved; and (iii) multiple agents cannot be assumed all to use the same descriptions – or names – for objects.

Note that Susie herself may be totally unaware of the content-contradiction lurking in her beliefs, even if she knows all logical consequences of her beliefs. The contradiction depends on semantics, or at least on an outside view of her syntax, and not simply that syntax itself.

It seems there is some confusion here in the notion of what it is for a belief to be “about” something: one belief is about the facility seen-as-gym and the other about the same facility seen-as-club. Yet no one has been able to explicate the idea of content, so important for commonsense reasoning and behavior, in a way that avoids the apparent contradiction. One possible solution-route is to accept the contradiction in content and show how this can be fit into a theory of the ascription of belief in a way that retains predictive power concerning behavior; here “ascription” of a belief E to an agent G is simply the assertion that G believes E. Thus one might ascribe contradictory beliefs to Susie: she believes of that facility (the one that happens to be both the club and the gym) that it is open and that it is closed.¹⁰

Another solution-route might be to unpack “aboutness” in such a way that teases apart the two senses (gym vs club) by which the facility is known. This is related to the distinction between *de re* and *de dicto* belief ascriptions. The former corresponds to semantics (what object is referred to), the latter to syntax: we say Susie believes *de re* that a particular facility itself is open/closed; and we say she believes *de dicto* that whatever is referred to by her syntax (e.g., “the gym”) is, say, closed. However, there is considerable doubt whether belief *de re* – independent of a syntactic characterization – is a meaningful notion at all; see (Devitt, 1984).

6 CONCLUSION

Beliefs are complex and poorly understood phenomena, yet they seem to be the best hope we have for understanding intelligent behavior. While it may be that an autonomous intelligence will be designed and built without our understanding how it works, it seems a good bet that a deep understanding of beliefs (what they are, how they arise, how they affect behavior) could only give us a leg up on the design problem.

The main research issues I see are:

1. characterize beliefs more carefully in terms of agent behavior
2. integrate this characterization into powerful modes of default reasoning and belief revision
3. keep the baby (effective commonsense reasoning) and throw out the bathwater (consistency).

¹⁰Kripke urges that this is a problem of how to *ascribe* beliefs, rather than about belief simpliciter. In either case it is a problem for the designer of AI systems: how are we to understand or analyze agent beliefs if we cannot define or ascribe them with clarity?

References

- Alchourron, C., Gardenfors, P., and Makinson, D. (1985). On the logic of theory change. *J. Symbolic Logic*, 50:510–530.
- Bacchus, F., Grove, A., Halpern, J., and Koller, D. (1993). Statistical foundations for default reasoning. In *IJCAI*, pages 563–569.
- Churchland, P. (1984). *Matter and Consciousness*. MIT Press, Cambridge, MA.
- des Rivières, J. and Levesque, H. (1986). The consistency of syntactical treatments of knowledge. In *Proceedings of the conference on Theoretical Aspects of Reasoning about Knowledge*, pages 115–130.
- Devitt, M. (1984). Thoughts and their ascription. In French, P. A., Uehling, T. A., and Wettstein, H. K., editors, *Midwest Studies in Philosophy, Volume IX: Causation and Causal Theories*. University of Minnesota Press, Minneapolis, MN.
- Elgot-Drapkin, J., Kraus, S., Miller, M., Nirke, M., and Perlis, D. (1993). Active logics: A unified formal approach to episodic reasoning. Technical Report UMIACS TR # 99-65, CS-TR # 4072, Univ of Maryland, UMIACS and CSD.
- Elgot-Drapkin, J. and Perlis, D. (1990). Reasoning situated in time I: Basic concepts. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(1):75–98.
- Fagin, R., Halpern, J., Moses, Y., and Vardi, M. Y. (1995). *Reasoning About Knowledge*. MIT Press.
- Gettier, E. (1963). Is justified true belief knowledge? *Analysis*, 23:121–123.
- Gilmore, P. (1974). The consistency of partial set theory without extensionality. In Jech, T., editor, *Axiomatic Set Theory*, pages 147–153. Amer.Math. Soc.
- Ginsberg, M., editor (1987). *Readings in Nonmonotonic Reasoning*. Morgan Kaufmann.
- Gurney, J., Perlis, D., and Purang, K. (1997). Interpreting presuppositions using active logic: From contexts to utterances. *Computational Intelligence*, 13(3):391–413.
- Hintikka, J. (1962). *Knowledge and Belief*. Cornell University Press, Ithaca, NY.
- Konolige, K. (1986). *A Deduction Model of Belief*. Pitman, London.
- Kripke, S. (1975). Outline of a theory of truth. *Journal of Philosophy*, 72:690–716.
- Kripke, S. A. (1979). A puzzle about belief. In Margalit, A., editor, *Meaning and Use: Papers Presented at the Second Jerusalem Philosophical Encounter*, pages 239–283. D. Reidel.
- Levesque, H. (1984). A logic of implicit and explicit belief. In *Proceedings of the National Conference on Artificial Intelligence*, pages 198–202, Austin, TX. American Association for Artificial Intelligence.
- McDermott, D. and Doyle, J. (1980). Non-monotonic logic I. *Artificial Intelligence*, 13(1,2):41–72.
- Miller, M. and Perlis, D. (1993). Presentations and this and that: logic in action. In *Proceedings of the 15th Annual Conference of the Cognitive Science Society*, pages 747–752, Boulder, Colorado.

- Montague, R. (1963). Syntactical treatments of modality, with corollaries on reflection principles and finite axiomatizability. In *Modal and Many-Valued Logics (Acta Philosophica Fennica, vol. 16)*. Academic Bookstore, Helsinki. Reprinted in R. Montague (1974). *Formal Philosophy*, New Haven, pp. 286–302.
- Morreau, M. and Kraus, S. (1998). Syntactical treatments of propositional attitudes. *Artificial Intelligence*, 106:161–177.
- Nilsson, N. J. (1983). Artificial intelligence prepares for 2001. *AI Magazine*, 4(4):7–14.
- Perlis, D. (1985). Languages with self reference I: Foundations. *Artificial Intelligence*, 25:301–322.
- Perlis, D. (1986). On the consistency of commonsense reasoning. *Computational Intelligence*, 2:180–190.
- Perlis, D. (1988). Languages with self reference II: Knowledge, belief, and modality. *Artificial Intelligence*, 34:179–212.
- Perlis, D. (1996). Nine sources of inconsistency in commonsense reasoning. In *1996 Workshop on Commonsense Reasoning*, Stanford.
- Perlis, D., Purang, K., and Andersen, C. (1998). Conversational adequacy: Mistakes are the essence. *International Journal of Human Computer Studies*, pages 553–575.
- Reiter, R. (1980a). Equality and domain closure in first-order databases. *Journal of the ACM*, 27:235–249.
- Reiter, R. (1980b). A logic for default reasoning. *Artificial Intelligence*, 13(1,2):81–132.
- Thomason, R. (1980). A note on syntactical treatments of modality. *Synthese*, 44:391–395.
- Traum, D. and Andersen, C. (1999). Representations of dialogue state for domain and task independent meta-dialogue. In *Proceedings of the IJCAI99 workshop: Knowledge And Reasoning in Practical Dialogue Systems*, pages 113–120.

Chapter 17

MODELING THE BELIEFS OF OTHER AGENTS

Achieving Mutuality

Richmond H. Thomason

AI Laboratory

University of Michigan

rich@thomason.org

Abstract This paper develops a formalism combining elements of epistemic and nonmonotonic logic, which is applied to the general topic of modeling the attitudes of other agents, and in particular to the problem of achieving mutuality.

Keywords: Epistemic logic, modal logic, reasoning about knowledge, reasoning about belief, nonmonotonic logic

1 INTRODUCTION AND BACKGROUND

The larger project of which this paper is a part is an attempt to develop a logical framework for *agent modeling*, the business of formulating and maintaining representations of the attitudes of other agents. This is something that normal, adult humans do very well. Take reasoning about beliefs. Under a wide variety of circumstances, we are able to reach rather detailed hypotheses about what the people with whom we interact believe, and to modify these hypotheses in the light of new evidence. But for some reason, it is especially difficult to obtain reflective insight into how we do this reasoning, to break it down into steps and to formalize it.

This paper is devoted to reasoning about belief and belief-like attitudes, and especially to the problem of how agents can achieve mutuality by this reasoning.¹

¹Readers who are not familiar with the term ‘mutuality’ should consult Section 2.1, below. There are terminological differences in the literature—some authors use ‘common’, as in ‘common ground’ and ‘common knowledge’, others use ‘mutual’, as in ‘mutual belief’. I

1.1 THE NEED FOR MUTUALITY

I will not devote much space in this paper to motivating the importance in various human endeavors of mutuality (of knowledge, or belief, or of other similar attitudes), or to arguing for the plausibility of the hypothesis that groups are often capable of achieving something like mutuality. The need for mutuality seems to turn up whenever conscious, human-level interactions are analyzed at a sufficiently deep level. It shows up in philosophical accounts of convention (Lewis, 1969), in accounts of conversation (Clark and Marshall, 1981), and in economic analyses of interactions (Aumann, 1976). Maybe it will suffice to give just one example of the importance of mutuality in thinking about such phenomena: you can't model a game like stud poker without accounting for the difference between the cards that are dealt face up and the ones that are dealt face down. The difference is simply that the values of the face-up cards are mutually known, and the values of the face-down cards are not.

1.2 AN OPPORTUNITY FOR LOGIC-BASED AI

I believe that agent modeling presents special opportunities for logic-based AI. That is, I believe that the techniques that have been developed in logic-based AI are well suited for formalizing much of the reasoning that is involved in agent modeling, and that the probabilistic techniques which work so well in modeling many natural phenomena are not likely to work well in many cases of agent modeling. The main purpose of this paper is to establish at least the positive point—I will try to show that a combination of modal logic and nonmonotonic logic provides a suitable foundation for agent modeling.

2 THE PROBLEM OF ACHIEVING MUTUALITY

2.1 WHAT IS MUTUALITY?

In this paper p , q , and r are used for propositions, and A , B and C for syntactic objects, usually sentences of a natural language or formulas of a formalized language. Propositions are to be understood as objects of propositional attitudes. The theory presented in this paper subscribes to a formalization of propositions that identifies them with sets of possible worlds. In the following, where a sentence or formula A is mentioned in connection with a proposition p , it is assumed that A expresses p .

Take a group G of agents. Suppose that this group has attitudes—say beliefs—and that among these are beliefs about the beliefs of other agents. In that case, we can talk about mutual belief for the group. A proposition p is *mutually believed* by the group in case every member of the group believes p , and believes that every member believes p , and believes that every member

prefer the latter term, because it nominalizes more happily: ‘mutuality’ sounds better than ‘commonality’.

believes that every member believes p , and so forth. So, for instance, if p is mutually believed by a group G including members a , b , and c , then a believes that c believes that b believes that a believes that p . Similarly, a sentence or formula is mutually believed if it expresses a proposition that is mutually believed.

Using notation from modal logic, I mark an attitude relating an agent a to a proposition p by $[a]A$. From here on, except in a few places where it is convenient to state things generally, I will confine myself to groups containing just two agents, a and b .

Some more notation: for $\alpha_i \in \{a, b\}$, let $[\alpha_1 \dots \alpha_n] = [\alpha_1] \dots [\alpha_n]$. Where α is a string over the alphabet $\{a, b\}$, α^n is the string consisting of n repetitions of α . In particular, \emptyset is the empty string.

We make the idea of an iterated attitude, and its connection to mutuality, precise in the following definitions.

- Definition 1 (Iteration depth.)**
1. $A[a]$ -iterates to depth 0 for all A .
 2. $A[a]$ -iterates to depth $1 + 2n$ in a model or example iff $[a(ba)^n]A$ is true in the model or example.
 3. $A[a]$ -iterates to depth $1 + 2n + 1$ in a model or example iff $[a(ba)^n b]A$ is true in the model or example.
 4. No $A[a]$ -iterates to depth ζ for any $\zeta \geq \omega$.

Definition 2 (Iteration complexity.) The $[a]$ -iteration complexity of A in a model or example is the smallest ordinal $\eta \leq \omega$ such that A does not iterate to depth $\eta + 1$ in the model or example.

$[b]$ iteration complexity is defined analogously to $[a]$ iteration complexity.

Mutuality is characterized globally for groups; for groups of more than one agent, it can't be defined in terms of the attitudes of any single agent. But reasoning is performed by individuals and is agent-bound. In this paper, where the focus is on reasoning, it is natural to concentrate on the iteration complexity for the attitudes of a single, fixed agent. I take this approach in the discussion below, and in the subsequent formal developments. Fortunately, mutuality can be characterized in terms of the attitudes of the separate agents in a group: p will be mutually believed if and only if the iteration complexity of A is infinite for every member of the group.

In an example where $[a]A$ is false, the $[a]$ iteration complexity of A is 0. If $[a]A$ is true but $[a][b]A$ is false, the $[a]$ iteration complexity of A is 1. If $[a]A$ and $[a][b]A$ are true but $[a][b][a]A$ is false, the $[a]$ iteration complexity of A is 2. If the $[a]$ and the $[b]$ iteration complexity of A are both ω , A is mutual for the group $\{a, b\}$ and the attitude $[]$. If the a -believes and the b -believes iteration complexity of A is greater than 0, but finite, we have a case in which both a and b believe p , and they may believe that each other believes p , and so forth, but at some finite point the iteration plays out. It is possible to construct plausible examples where the a -believes and the b -believes iteration complexities of A are finite but greater than one. But as the complexity increases, the examples become more contrived and more difficult

to think about intuitively. Some authors have noticed that human reasoners are not very good at reasoning about finite levels of iteration complexity greater than 2; people seem to be most comfortable with 0, 1, and ω . The formalization that I provide below goes a small way towards explaining this phenomenon.

2.2 IMMEDIACY THEORIES

I take it for granted that we want a formalized theory of the reasoning processes that yield mutual attitudes, and that a multimodal logic is the appropriate vehicle for formalizing the relevant agent attitudes.

The most detailed studies with which I am familiar of the reasoning leading to mutual attitudes² all suggest that an immediacy-based approach can account for the reasoning. The three accounts are similar, partly because the two later ones are influenced by Lewis. None of them is fully formalized; they all suggest that mutuality is somehow precipitated by the reflexive character of certain shared situations. There are differences in the three versions, which would probably lead to different formalizations; (Barwise, 1988) suggests different formalizations of (Lewis, 1969) and (Clark and Marshall, 1981). However, I don't believe that these differences are relevant to the points I wish to make.

I believe that intuitions about immediacy fail to provide an adequate formalization of the relevant reasoning. I'll use a simplified version of the immediacy theory to explain the point. Although it doesn't correspond exactly to any earlier presentation of the idea and won't do full justice to the views I mean to undermine, I don't think it will leave out anything essential or weaken the force of the arguments.

Let's say that a sentence or formula A guarantees B -mutuality for agent a and group $\{a, b\}$ in an example or model if

$$\text{MutAx} \quad [a][A \rightarrow [\text{MUT}]B]$$

holds in that example or model. A proposition p guarantees B -mutuality for agent a and group $\{a, b\}$ in an example or model if A guarantees B -mutuality, where, as usual, A expresses p .

The idea behind immediacy is that agents apparently often find themselves in circumstances where something like MutAx can be used to secure mutuality. The following sorts of examples are used to illustrate the point. In Example 1, (1a) represents A in MutAx and (1b) represents B .³ In Example 2, (2a) represents A in MutAx and (2b) represents B .⁴

- (1a) a and b are sitting across from one another at a small dining table, looking at a candle on the table and at each other looking at the candle.
- (1b) There is a candle on the table.
- (2a) b says to a , in a face-to-face conversation, "I will be here tomorrow at 9am." The day of the utterance is d , the place of utterance is l .

²By (Lewis, 1969), (Schiffer, 1972), and (Clark and Marshall, 1981).

³The example is from (Lewis, 1969, pp. 52-57).

⁴An example from (Schiffer, 1972, pp. 31-36).

(2b) b will be at l at 9am on $d + 1$.

In Example 1, what guarantees mutuality for (1b) is the proposition that a and b are face-to-face across a small table, and are both looking at the candle—or maybe a qualification of this proposition.⁵ In Example 2, what guarantees mutuality for (2b) is the proposition that, in a face-to-face conversation, a has just said to b “I will be here tomorrow at 9am.”

I don’t want to quarrel with the insight behind these examples: that circumstances such as these allow mutuality to be inferred. In fact, I want to say that in these circumstances, each agent can infer an infinite iteration complexity for the appropriate propositions. But I don’t want to say that an axiom like MutAx guarantees this inference. I’ll divide my explanation of why I think that immediacy theories provide an inadequate account of the reasoning into three topics: *incrementality*, *monolithicity*, and *fallibility*.

2.2.1 Incrementality issues. Finite iteration complexity can arise in naturally occurring situations. A classic series of examples of increasing finite complexity is presented in (Clark and Marshall, 1981); another is developed in Section 9, below, where for a certain A , $[a][b]A$ holds but $[a][b][a]A$ does not. By making inferences of mutuality take place in a single step whenever they do take place, immediacy approaches allow the reasoning that produces mutuality to fail in only one way—by blocking all iteration depths. A reasoning process that leads to an iteration complexity of, say, 2, apparently must involve entirely other forms of reasoning. An account of the reasoning that is more unified than this, I think, would be more plausible and more explanatory.

2.2.2 Monolithicity issues. MutAx implies $[a]A \rightarrow [a][b]A$ and $[a]A \rightarrow [a][b][a]A$. But this second formula is not plausible in many cases where mutuality is wanted. Suppose that Ann is an attorney defending a client, Bob, whose story she does not believe. In interviewing her client, she suspends her disbelief; the best she can do for Bob is to take his story at face value. Her interview with Bob makes use of many presuppositional features of conversation that are generally assumed to require mutuality—features like definite reference. She says, “Now, after you got up from your nap, did you make any phone calls?” But she doesn’t in fact believe that Bob took a nap while the robbery he is accused of took place. Bob doesn’t believe that she believes this. Nevertheless, the interview takes place without any presuppositional anomalies, without any of the abnormalities that accompany failures of mutuality.

Robert Stalnaker has discussed similar problems (Stalnaker, 1975), and has proposed a natural solution: invoke an attitude of “belief for the sake of conversation.” Ann and Bob are modeling not each other’s beliefs, but each other’s C-suppositions, suppositions for the sake of this particular conversation. For instance, $[a][b][a]A$ represents Ann’s C-supposition that Bob C-supposes that Ann C-supposes that p . Their mutuality is possible

⁵Schiffer produces a fairly elaborate qualification, (Schiffer, 1972, p. 35).

because the participants are constructing, for this conversation, a special purpose attitude that not only serves to keep track of the conversation but that maintains mutuality. Under some circumstances, this local mutuality may precipitate actual mutual belief, but these circumstances are decoupled from the rules that govern conversation.⁶

The achievement of mutuality in conversation, then, depends on the ability of the participants to construct at the beginning of the conversation an appropriate *ad hoc* attitude, which from one point of view models the content of the conversation, and from another point of view models for each participant the other participants' views of this content.

Now, the things that are supposed for the sake of conversation will include not only what has been contributed to the conversation by speech acts, but what the participants can reasonably expect to be mutual at the outset.⁷ Here, we find ourselves assuming that agents must be able to associate observable properties of other agents with an appropriate initial attitude which is assumed to be mutual. And if mutuality is taken to be a mark of successful conversation, then we are also supposing that agents often initialize C-supposition in much the same way.

Suppose, for instance, that Ann meets Bob at an AAAI conference, and begins a conversation. She supposes that Bob is a computer scientist. Assume that when Ann learned computer science, she kept track of the things she learned that she could expect any computer scientist to learn. It will be possible for her to do this if, as she learned things, she remembered not only the learned items but the type of circumstance in which they were learned.

In (Thomason, 1998), I propose to model this using a modality $[a, \text{CS}]$ to represent the things that Ann expects any computer scientist to believe.⁸ Now, Ann expects computer scientists to organize what they learned in much the same way. So she not only expects Bob to have learned, for instance, that finite state automata accept regular languages, but that any computer scientist can be expected to have learned this. That is, a modal model of Ann's mental contents will contain not only $[a, \text{CS}]A$, but $[a, \text{CS}][b, \text{CS}]A$.

Introducing into the makeup of single agents special-purpose modalities whose purpose is mutual modeling provides a more plausible way of producing iterated attitudes. Below, in Section 8, I show how this idea can lead to infinite iteration complexities, and so—at least, under favorable conditions—to full mutuality.

The technique of modeling a single agent's beliefs by a family of modal operators is not needed to account for many of the logical issues involved in mutuality. But it is indispensable in accounting in practice for a wide range of examples that involve mutuality.

⁶The work done by the distinction between C-supposition and belief is similar the work done by J.L. Austin's distinction between illocutionary and perlocutionary acts.

⁷See (Clark and Schober, 1989, pp. 257–258).

⁸More generally, if Ann can match Bob to features $M_1 \dots M_n$, she can use a modality $[M_1 \sqcap \dots \sqcap M_n]$ to initialize the conversation, where $wR_{M_1 \sqcap \dots \sqcap M_n} w'$ iff for all i , $1 \leq i \leq n$, $wR_i w'$.

2.2.3 Fallibility. The examples like (1) and (2) above that are usually given to motivate immediacy theories of mutuality are chosen to conceal the worst flaw of these theories—the defeasibility of the associated reasoning. If you and I are sitting at a small table, looking at a candle on the table and at each other looking at the candle, it is hard to see how we could fail to mutually believe there is a candle on the table. But this is due in part to the fact that we tend to imagine normal cases, especially in interpreting narratives.

The plausibility of the examples that are given to support immediacy theories depends to a large extent on the fact that we read them as narratives. In general, we expect authors to state explicitly anything that is abnormal in the circumstances they relate. So, in Schiffer's example, for instance, we assume that the candle is not a novelty item designed to look like a wine bottle. This assumption makes the case for MutAx more plausible.

In real life, we often have to deal with cases that are less straightforward. If a group of five quarters is lying on the table, I'm pretty safe in assuming that we mutually believe there is \$1.25 on the table, but I could well be wrong. If there are eleven quarters, the assumption that we mutually believe there is \$2.75 on the table is riskier, but I might well make it.

The initializing assumptions we make that are not based on immediately perceived mutual situations are even more patently defeasible. For instance, Example 2 fails if one participant takes 'here' to refer to a street corner, while the other takes it to refer to a city. Part of being a skilled conversationalist is to make well-coordinated assumptions, realizing at the same time that they may be incorrect, and having a notion of how to correct things if the assumptions should fail. Maybe Bob is a book exhibitor rather than a computer scientist. Once Ann finds this out, she will probably have an idea of where the conversation went wrong, and how to adjust it. Even a conversation that begins with the participants fully coordinated can lose mutuality, because of ambiguity and inattention. Skilled conversationalists are able to identify and repair such failures.⁹

Axioms like MutAx are inadequate in accounting for such phenomena. To qualify as an axiom, MutAx has to hold across all the examples in its intended domain of interpretation. In most cases when we want to imagine that interacting agents apply MutAx, the agents would be well aware that the axiom could be false, and so that it lacks the properties of an axiom. We could try to remedy this difficulty by using refined axioms of the form

$$\text{Qualified MutAx} \quad [a][[A \wedge B] \rightarrow [\text{MUT}]C],$$

where B is a conjunction of clauses which together eliminate the cases in which mutuality could fail.

But this merely relocates the problem. First, though it is often possible to find reasonable qualifying conditions, and to further improve these by further refinements, it seems impossible in realistic cases to bring the process of refinement to an end.¹⁰ Second, we are typically willing to use Qualified MutAx

⁹See (Mortensen, 1996).

¹⁰This is the qualification problem. See, for instance, (McCarthy, 1980).

without explicitly checking the qualifying conditions. These two circumstances are best dealt with by using a nonmonotonic logic.

The approach that I develop in this paper combines solutions to all three of these problems. I use a nonmonotonic logic, which secures ω -level iteration complexity in one step in the normal cases, but which in principle could fail at any finite iteration level. Within this logic, it is possible to develop a theory of exceptions to the normal case. Such a theory, I believe, is an essential part of any solution to the problem of inferring mutuality, since this reasoning is failure-prone, and agents need informed ways of recovering from failures. The logic represents individual beliefs as interrelated families of modalities; this mechanism provides a workable approach to the problem of initializing mutual attitudes.

3 SUBAGENT SIMULATION AS AN AGENT MODELING MECHANISM

From uses of multiple modalities in logical models of multi-agent systems (see (Fagin et al., 1995)) and contextual reasoning (see (Buvač and Mason, 1993)), we are familiar with the idea of modal logics in which indices are attached to the modalities, where these indices stand either for agents or microtheories. I propose to use this apparatus to model the modularization of single-agent belief that is required in the AAAI conference example of Section 2.2.2. Ann's beliefs in that example are now to be represented not by a single modality $[a]$, but by a family of modalities $[a, i]$, where $i \in I_a$. Here, I_a is a set of "subagents," or indices standing for special-purpose belief modules. In the example, Ann uses a modality $[a, cs]$ that singles out things that any computer scientist could be expected to have learned.

The general idea is similar to modal theories of context, such as that of (Buvač and Mason, 1993). So the general contours of the theory should be recognizable to those who are familiar with these theories. For instance, there will be "lifting rules" that organize the distribution of information among the subagents of a single agent. Although an agent a can obtain information from another agent b (for instance, by communication), this is not a matter of a 's internal epistemic organization, and we certainly do not want to relate indices $\langle a, i \rangle$ and $\langle b, i \rangle$ by monotonic lifting rules. But a 's beliefs about b 's beliefs do in general depend on beliefs of the subagents of a that imitate subagents of b ; so we will have lifting rules, rules that relate beliefs of some of a 's subagents to a 's beliefs about b 's beliefs.

Although the logic is similar to modal logics of context, there are extra complications due to the need to distinguish intra-agent from inter-agent modalities. We begin with the intra-agent logic.¹¹

¹¹Some of the next section corresponds to parts of (Thomason, 1998).

4 MODELING THE MULTIPLICITY OF SINGLE-AGENT BELIEFS

Some subagents can access other subagents. This is not a form of communication. It reflects the modular epistemic organization of a single agent. I will not go into details here, but I believe that this organization of the individual's epistemology is useful for the same reasons that make modularity useful in knowledge representation. There are, of course, many analogies between the organization of large-scale knowledge bases into microtheories, as discussed in (Guha, 1991) and the organization of individual attitudes that I am proposing here.

When a subagent i does not access j , I will assume that j is entirely opaque to i . We might model this by disallowing formulas like $[i][j]A$, but linguistic restrictions of this kind are in general less satisfactory than a semantic treatment. So I will assume that $[i][j]A$ is false if i can't access j .

These ideas lead to the following definition.

Definition 3 (Intra-Agent Modal Languages.) An intra-agent propositional language $\mathcal{L} = \langle \mathcal{I}, \preceq, \mathcal{P} \rangle$ is determined by the nonempty set \mathcal{I} of indices, a reflexive, transitive ordering \preceq over \mathcal{I} and a nonempty set \mathcal{P} of basic propositions.

\mathcal{I} is the set of subagents of the language, and \preceq determines accessibility for subagents. If $i \preceq j$ then i accesses j .

Definition 4 (Intra-Agent Modal Formulas.) The set $\text{FORMULAS}(\mathcal{P}, \mathcal{I})$ is the smallest set that (1) contains \mathcal{P} , (2) is closed under boolean connectives, and (3) is closed under i -necessitation, for $i \in \mathcal{I}$. I.e., for all $i \in \mathcal{I}$, if $A \in \text{FORMULAS}(\mathcal{P}, \mathcal{I})$, then $[i]A \in \text{FORMULAS}(\mathcal{P}, \mathcal{I})$.

Definition 5 (Intra-Agent Modal Frames.) An *intra-agent frame* $\mathcal{F} = \langle W, R \rangle$ for an intra-agent modal language $\mathcal{L} = \langle \mathcal{I}, \preceq, \mathcal{P} \rangle$ on a frame W consists of a nonempty set W of possible worlds and a function R providing a relation R_i over W for each $i \in \mathcal{I}$.

Definition 6 (Intra-Agent Modal Models.) A *model interpretation* M on an intra-agent frame $\mathcal{F} = \langle W, R \rangle$ for an intra-agent modal language $\mathcal{L} = \langle \mathcal{I}, \preceq, \mathcal{P} \rangle$ is an assignment of values $\llbracket P \rrbracket_M$ to the basic propositions P in \mathcal{P} ; $\llbracket P \rrbracket_M$ is a function from W to $\{T, F\}$. An *intra-agent modal model* \mathcal{M} of an intra-agent modal language $\mathcal{L} = \langle \mathcal{I}, \preceq, \mathcal{P} \rangle$ on a frame $\langle W, R \rangle$ consists of a triple $\langle M, w_0, i \rangle$, where M is a model interpretation on \mathcal{F} , $w_0 \in W$, and $i \in \mathcal{I}$. (w_0 is the initial world of the model, i is the designated subagent.)

The satisfaction relation $\mathcal{M} \models_{i,w} A$ is relativized to subagents as well as to worlds; formulas are true or false relative not only to a world, but to a subagent. $\mathcal{M} \models_{i,w} A$ means that M makes A true in w from the perspective of subagent i . The semantic effects of perspective are very limited; perspective influences only the truth values of modal formulas, and it affects these only in a limited way.

Definition 7 (Satisfaction in an Intra-Agent Modal System.) $\mathcal{M} \models_{i,w} P$ iff $\llbracket P \rrbracket_M(w) = T$. (Note that i is redundant in this case.) Satisfaction is standard for boolean connectives, and $\mathcal{M} \models_{i,w} \llbracket j \rbracket A$ iff $i \preceq j$ and for all w such that $w R_j w'$, $\mathcal{M} \models_{j,w'} A$.

Finally, $\mathcal{M} \models A$ iff $\mathcal{M} \models_{i,w_0} A$.

Depending on the application, we may wish to impose certain constraints on the relations R_i . Here, we are interested in the following conditions.

Transitivity. If $w R_i w'$ and $w' R_i w''$ then $w R_i w''$.

Euclideanness. If $w R_i w'$ and $w R_i w''$ then $w' R_i w''$.

Seriality. For all w , there is a w' such that $w R_i w'$.

Subagent Monotonicity. $R_i \subseteq R_j$ if $i \preceq j$.

Subagent Coherence. If $w R_i w'$ and $i \preceq j$ then $w' R_j w'$.

The combination of Transitivity, Euclideanness, and Seriality is commonly used in contemporary logical models of single-agent belief; see (Fagin et al., 1995). The remaining constraints are specific to intra-agent epistemic logic.

The resulting logic is a multimodal version of the non-normal logic **E2** that is formulated in (Lemmon, 1957) and proved complete in (Kripke, 1965).

I have a sound and complete axiomatization of the logic; but I will not discuss those details here.¹²

5 MODELING THE BELIEFS OF MANY AGENTS

We now want to imagine a community of agents. Each agent has modularized beliefs along the lines described above. But in addition, each has beliefs about its fellow agents; and these beliefs iterate freely. In fact, for multi-agent beliefs I want to adopt the familiar framework of (Fagin et al., 1995).

Intra-agent and multi-agent epistemic logic model fundamentally different aspects of reasoning about attitudes. In the latter case, agents form opinions about other agent's beliefs in much the same way that they form opinions about any other feature of the world. In the former case, when $i \preceq j$, then j represents a part of i 's opinion, and i directly accesses j in consulting its opinions.

I will now assume that we have indices for agents as well as for the associated subagents. Thus, we will have formulas like

$$\llbracket a, i \rrbracket [A \rightarrow \llbracket b, j \rrbracket [B \rightarrow \llbracket a, i \rrbracket C]],$$

where a and b are agent indices. Where A , B , and C express p , q , and r , respectively, this formula says that a 's i -module believes that if p then b 's j -module believes that if q then a 's i -module believes that r .

The notation may appear to assume that each subagent knows about how each other agent is decomposed into subagents, but this is required by the fact

¹²Some of the next section corresponds to parts of (Thomason, 1998).

that this decomposition is built into the language—and we do assume that a uniform language is available for subagents. Depending on the possibilities, any agent's subagents may be well informed or entirely ignorant about the beliefs of the subagents of other agents.

Definition 8 (Inter-Agent Modal Languages.) An inter-agent propositional language $\mathcal{L} = \langle \mathcal{P}, \mathcal{E}, \mathcal{A}, \mathcal{I}, \preceq \rangle$ is determined by a nonempty set \mathcal{P} of predicate letters; by a set \mathcal{E} of individual constants; by a nonempty set \mathcal{A} of agent indices; by a function \mathcal{I} on \mathcal{A} , where \mathcal{I}_a is a nonempty set of subagents (representing the subagents of a); and by a function \preceq which for each $a \in \mathcal{A}$ provides a reflexive, transitive ordering on \mathcal{I}_a . We require that $\mathcal{A} \subseteq \mathcal{E}$ —each agent index also serves as an individual constant.

I do not assume that if $a \neq b$, then \mathcal{I}_a and \mathcal{I}_b are disjoint; in fact, we often want to consider agents with the same general epistemic organization, and in this case, $\mathcal{I}_a = \mathcal{I}_b$ for all a and b .

In circumscriptive theories of belief, we will need abnormality predicates relating agents and propositions; for instance, we may want to characterize a proposition just asserted by a in a conversation as abnormal for a if it is not heard or not understood. So we will want formulas such as $Ab(a, P(b))$. It may be important to record whether agents believe that abnormalities hold, so we will need formulas such as $[a, i]Ab(c, P(b))$. Although I will not need them in this paper, I will not exclude formulas like $Ab_1(a, Ab_2(b, P))$. These ideas lead to the following formation rules.

A predicate letter P represents a relation over a finite number of arguments, each of which is either an individual or a proposition. The number of arguments may be zero; in that case, P is a propositional constant. I assume that there is at least one non-propositional predicate letter in \mathcal{P} , i.e., at least one predicate letter whose arguments are all individuals. To simplify things, I will also assume that arguments of individual type occur before arguments of propositional type.

Where $\mathcal{L} = \langle \mathcal{P}, \mathcal{E}, \mathcal{A}, \mathcal{I}, \preceq \rangle$, the set $\mathcal{B}_{\mathcal{L}}$ of *basic formulas* of \mathcal{L} consists of all formulas of the form P , where P is a zero-place predicate letter in \mathcal{P} , or of the form $P(t_1, \dots, t_n)$, where P is an n -place nonpropositional predicate letter in \mathcal{P} and $t_1, \dots, t_n \in \mathcal{E}$.

The set $\text{FORMULAS}(\mathcal{P}, \mathcal{I}, \mathcal{A})$ is the smallest set that:

- (1) extends $\mathcal{B}_{\mathcal{L}}$;
- (2) is closed under boolean connectives;
- (3) is closed under application of basic predicates to complex propositional arguments (i.e.,
 $P(t_1, \dots, t_n, A_1, \dots, A_m) \in \text{FORMULAS}(\mathcal{P}, \mathcal{I})$ if $t_1, \dots, t_n \in \mathcal{E}$,
 $A_1, \dots, A_m \in \text{FORMULAS}(\mathcal{P}, \mathcal{I}, \mathcal{A})$, and P takes n individual and
 m propositional arguments; and
- (4) is closed under i, a -necessitation for all $a \in \mathcal{A}$, $i \in \mathcal{I}_a$ (i.e., if
 $A \in \text{FORMULAS}(\mathcal{P}, \mathcal{I}, \mathcal{A})$, then $[a, i]A \in \text{FORMULAS}(\mathcal{P}, \mathcal{I})$, for all
 $a \in \mathcal{A}$ and $i \in \mathcal{I}_a$).

When we speak of a formula $[a, i]A$, we presuppose that $i \in \mathcal{I}_a$.

Definition 9 (Inter-Agent Modal Frames.) An *inter-agent frame* $\mathcal{F} = \langle W, D \rangle$ for $\mathcal{L} = \langle \mathcal{P}, \mathcal{E}, \mathcal{A}, \mathcal{I}, \preceq \rangle$ consists of nonempty sets W and D . W is the set of possible worlds of the frame and D is the domain of individuals. We will require that $\mathcal{A} \subseteq D$ —each agent index is also an individual. This means that we will treat agent indices as self-denoting individual constants.

We use relations on W as usual, to interpret modal operators, but now it makes better sense to include these relations in the models rather than in the frames.

Definition 10 (Inter-Agent Modal Models) A *model interpretation* M on an inter-agent frame $\mathcal{F} = \langle W, D \rangle$ for a language $\mathcal{L} = \langle \mathcal{P}, \mathcal{E}, \mathcal{A}, \mathcal{I}, \preceq \rangle$ is an assignment of appropriate values, determined by W and D , to constants of $\mathcal{L} = \langle \mathcal{P}, \mathcal{E}, \mathcal{A}, \mathcal{I}, \preceq \rangle$, where here we include modalities among the constants. In particular,

1. $\llbracket t \rrbracket_M \in D$ for individual constants t ; and if $t \in \mathcal{A}$, $\llbracket t \rrbracket_M = t$.
2. If P is a predicate letter taking n individual arguments and m propositional arguments, then $\llbracket P \rrbracket_M$ is assigned a function from W to subsets of $D^n \times \mathcal{P}(W)^m$, where $\mathcal{P}(W)$ is the power set of W . (Thus, for each world this function delivers an appropriate relation over individuals and propositions.)
3. $\llbracket [a, i] \rrbracket_M$ is a transitive, Euclidean, serial relation on W .

A *model* on an intra-agent frame $\mathcal{F} = W$ for a language $\mathcal{L} = \langle \mathcal{P}, \mathcal{E}, \mathcal{A}, \mathcal{I}, \preceq \rangle$ is a tuple $M = \langle M, w_0, a, i \rangle$, where M is a model interpretation, $w_0 \in W$, $a \in \mathcal{A}$, and $i \in \mathcal{I}_a$. (For many purposes, we can neglect the role that a and i play in satisfaction—often, then, we can simply think of a model as a pair (M, w_0) .) Where δ is a constant of \mathcal{L} , we let $\llbracket \delta \rrbracket_M = \llbracket \delta \rrbracket_{\mathcal{M}}$.

Definition 11 (Satisfaction.) The satisfaction relation $M \models_{a,i,w} A$ is relativized to subagents and to worlds, as before. (But the two indices a, i reflect the fact that a subagent is now a pair consisting of an agent and an index.)

For formulas A of the form $P(t_1, \dots, t_n)$, $M \models_{a,i,w} A$ iff $(\llbracket t_1 \rrbracket_M, \dots, \llbracket t_n \rrbracket_M) \in \llbracket P \rrbracket_M(w)$.

For any formula A , $\llbracket A \rrbracket_{M,a,i} = \{w \in W : M \models_{a,i,w} A\}$.

For formulas A of the form $P(t_1, \dots, t_n, A_1, \dots, A_m)$, $M \models_{a,i,w} A$ iff $(\llbracket t_1 \rrbracket_M, \dots, \llbracket t_n \rrbracket_M, \llbracket A_1 \rrbracket_{M,a,i}, \dots, \llbracket A_m \rrbracket_{M,a,i}) \in \llbracket P \rrbracket_M(w)$.

Satisfaction conditions are standard for boolean connectives, and finally

$M \models_{a,i,w} [b, j] A$ iff either (1) $a = b$, $i \preceq_a j$, and $M \models_{b,j,w'} A$ for all w' such that $w R_{a,j} w'$, or (2) $a \neq b$ and $M \models_{a,i,w'} A$ for all w' such that $w R_{b,j} w'$.

Where $M = \langle M, w_0, a, i \rangle$ is a model, $M \models A$ iff $M \models_{a,i,w_0} A$. Where T is a set of formulas, M simultaneously satisfies T , or is a model of T , iff $M \models A$ for all $A \in T$. And T implies A iff $M \models A$ for all models M that simultaneously satisfy T .

It is reasonable to require that for all $a \in \mathcal{A}$ there is a unique $i_a \in \mathcal{I}_a$ that is \preceq_a minimal in \mathcal{I}_a : for all $j \in \mathcal{I}_a$, $i \preceq_a j$; i_a represents the compiled beliefs of agent a .

As before, we are primarily interested in models that are transitive, Euclidean, and serial. Although I am also interested in nonmonotonic relaxations of subagent monotonicity, so far I have only looked at the case in which models are subagent monotonic and coherent.

Both the pure intra-agent logic and the subagentless multi-agent epistemic logic are special cases of inter-agent modal logic. We obtain the familiar multi-agent case by letting $\mathcal{I}_a = \{i_a\}$ for all $a \in \mathcal{A}$. We obtain the pure intra-agent case by letting $\mathcal{A} = \{a\}$.

For the remainder of this paper, I will only consider *agent-homogeneous* languages, where $\mathcal{I}_a = \mathcal{I}_b$ for all a and b . In this case, for each $i \in \mathcal{I}$ we can add a mutual belief operator $[\text{MUT}, i]$ to the logic. We can also simplify things by thinking of a language \mathcal{L} as a tuple $\langle \mathcal{P}, \mathcal{E}, \mathcal{A}, \mathcal{I}, \preceq \rangle$, where \mathcal{I} is a fixed set of indices. From now on, I will work with this simpler account of languages.

Definition 12 (Inter-Agent Modal Systems with Mutual Belief.) An inter-agent propositional language $\mathcal{L} = \langle \mathcal{P}, \mathcal{E}, \mathcal{A}, \mathcal{I}, \preceq, \text{MUT} \rangle$ with mutual belief is an agent-homogeneous inter-agent propositional language with a modal operator $[\text{MUT}, i]$ for each $i \in \mathcal{I}$. The satisfaction condition for $[\text{MUT}, j]$ is as follows:

$\mathcal{M} \models_{a,i,w} [\text{MUT}, j]A$ iff $i \preceq j$ and $\mathcal{M} \models_{a,i,w'} A$ for all w' such that wR_j^*w' , where R_j^* is the transitive closure of the set of relations $\{R_{b,j} : b \in \mathcal{A}\}$.

The resulting logic contains standard multi-agent modal logics for reasoning about mutual belief, such as the system KD45_n^C of (Fagin et al., 1995).

Definition 13 ($[\alpha, i]$) Where α is the string $\alpha_1 \dots \alpha_n$ over \mathcal{A} , we let $[\alpha, i] = [\alpha_1, i] \dots [\alpha_n, i]$.

Lemma 1 $\{[\alpha, i]A : \alpha \text{ a nonempty string over } \mathcal{A}\}$ implies $[\text{MUT}, i]A$.

Proof. Suppose that $\mathcal{M} = \langle M, w_0, a, j \rangle$ is a model for $\mathcal{L} = \langle \mathcal{P}, \mathcal{E}, \mathcal{A}, \mathcal{I}, \preceq, \text{MUT} \rangle$ that simultaneously satisfies $\{[\alpha, i]A : \alpha \text{ a string over } \mathcal{A}\}$, and let $w_0 R_i^* w$. Then for some $\alpha = \alpha_1 \dots \alpha_n$ and w_1, \dots, w_n , we have $w_0 R_{\alpha_1, i} w_1, \dots, w_{n-1} R_{\alpha_n, i} w_n$, where $w_n = w$. By hypothesis, $\mathcal{M} \models_{a,j,w_0} [\alpha, i]A$. So $\mathcal{M} \models_{a,j,w} A$. Now, we also have $j \preceq i$, since $\mathcal{M} \models_{a,j,w_0} [\alpha, i]A$. Therefore, $\mathcal{M} \models_{a,j,w} [\text{MUT}, i]A$.

Lemma 2 $\{[\alpha, j][\beta, i]A : \beta \text{ a nonempty string over } \mathcal{A}\}$ implies $[\alpha, j][\text{MUT}, i]A$.

Proof. This follows directly from Lemma 1, together with principles of modal logic.

Lemma 3 $\{[\alpha][\text{MUT}]A : a \in \mathcal{A}\}$ implies $[\text{MUT}]A$.

Proof. Clearly, $\{\llbracket a \rrbracket [\text{MUT}]A : a \in \mathcal{A}\}$ implies $\llbracket \alpha \rrbracket A$ for all nonnull strings α over \mathcal{A} , since any such string has the form $b\alpha$, where $b \in \mathcal{A}$. By Lemma 1, then, $\{\llbracket a \rrbracket [\text{MUT}]A : a \in \mathcal{A}\}$ implies $\llbracket \text{MUT} \rrbracket A$.

Together, these lemmas provide us with a methods for establishing mutuality and attitudes about mutuality. These methods will be used below in Section 8 to show how beliefs about mutuality (and, in favorable circumstances, mutuality itself) can be secured by default.

6 FORMULATING PROPOSITIONAL GENERALIZATIONS

In formalizing the problem of achieving mutual attitudes, we will need to express generalities about agents and propositions. A simple generalization of the sort I have in mind would be an axiom saying of a particular “bellwether” agent a that if this agent’s public module believes anything, then every agent’s public module believes this thing. It is most natural to use propositional quantifiers to state such generalizations:

$$(6.1) \forall P[\llbracket a, \text{PUB} \rrbracket P \rightarrow \forall x[\text{Agent}(x) \rightarrow \llbracket x, \text{PUB} \rrbracket P]]$$

This axiom uses a quantifier $\forall x$ over individuals and a quantifier $\forall P$ over propositions.

Unfortunately, propositional quantifiers complicate the logical setting. In cases where there is only one modal operator, the complexity of logics with propositional quantification differs wildly, depending on the type of the modality. The logic could be decidable, or it could be equivalent to second-order modal logic. (See (Fine, 1970) and (Kremer, 1997).) As far as I know, the case with multiple modalities has not been investigated to any extent, but I have an unpublished proof that, even with the modalities that are decidable in the monomodal case, multimodal logics with propositional quantifiers are equivalent to full second-order logic. I believe that this complexity is an inevitable consequence of a general approach to the phenomenon of knowledge transfer.¹³ Hopefully, however, tractable special cases will emerge.

Worse, propositional quantifiers create special difficulties in combining epistemic logic with circumscription, by making the technique of model separation introduced in Section 7 unusable. I have a way to overcome this problem using nonstandard models, but I can spare you the details of this by taking advantage of the fact that the only constraints needed to secure mutuality have the form

$$(6.2) \forall P_1 \dots \forall P_n A,$$

where A has no propositional quantifiers. This means that we can use axiom schemata in place of axioms that involve propositional quantification. The use of axiom schemata means that the theories we will consider will not be finitely axiomatizable, but that is a pretty small price to pay in this context.

By confining ourselves to cases where there are finitely many agents, we can (temporarily, at least) dispense with quantification over individuals. Our

¹³Similar problems arise, for instance, in the general logic of contextual reasoning.

investigation of the logic of mutuality, then, will take place in the context of inter-agent modal propositional logics with mutual belief.

7 CIRCUMSCRIPTIVE REASONING ABOUT BELIEFS

Several frameworks have been proposed for formalizing nonmonotonic reasoning about beliefs: autoepistemic logic, (Morgenstern, 1990); circumscription in a higher-order modal logic, (Thomason, 1990); default logic (or something like it), (Parikh, 1991); only-knowing, (Halpern and Lakemeyer, 1996); and preferential models, (Wainer, 1993) (Monteiro and Wainer, 1996).

I will adopt a circumscriptive approach, partly because it is useful and straightforward as a development tool, and partly because circumscription appears to provide more powerful and flexible mechanisms for describing complex configurations of abnormalities. Fortunately, the interrelations between the various approaches to nonmonotonic logic are by now pretty thoroughly worked out, and in the simple cases at least it is possible to go fairly easily from one to the other. Hopefully this will carry over to epistemic applications.

The early circumscriptive accounts of nonmonotonic reasoning appeal to a completion operation on finitely axiomatized theories, which takes the original theory into one in which certain terms are minimized. However, later formulations dispense with the need for a finitely axiomatized theory by defining circumscriptive consequence in model theoretic terms. Here, I will use the latter approach.

Circumscriptive modal logics are a neglected topic. In thinking through this application of circumscription, I encountered technical difficulties, due to the interaction of the possible worlds interpretation of propositional attitudes with circumscription. The purpose of the next two paragraphs is to explain these difficulties and to motivate their formal solution.

In circumscribing a theory, we minimize the extensions of some constants in a space of models in which the extensions of other constants are allowed to vary. In standard circumscriptive theories, the constants are all either first-order predicates or functors; but there are no formal difficulties in applying circumscriptive techniques to constants of other logical types, and in particular to modal operators, which for our purposes can be thought of as predicates of propositions. In fact, since we are interested in minimizing differences between the beliefs of agents, the only constants that are minimized or varied in simple cases will be abnormality predicates (which mark discorrelations among certain beliefs) and modalities. (In the presence of a theory of abnormalities, it may be necessary to consider more complex patterns of variation.) In possible worlds semantics, beliefs are not represented directly as predicates of propositions but are characterized indirectly in terms of relations over possible worlds. This lack of uniformity creates some difficulties in motivating an appropriate account of circumscription.

Suppose that we are trying to minimize certain abnormalities for an agent a , in order to make a 's picture of b 's beliefs correspond, as far as possible,

to a 's own beliefs. Intuitively, the things we want to vary here are a 's beliefs about b 's beliefs. But in all our models, the only parameter we can vary is the relation $R_{a,i}$ that serves to interpret the modal operator $[a,i]$ in which we are interested. Suppose we are working with a base world w_0 . The closest we can come to varying a 's beliefs about b 's beliefs would be to vary how b 's beliefs work in worlds epistemically accessible for a from w_0 . That is, we vary the worlds b, i -related to w , for worlds w such that $w_0 R_{a,i} w$. The problem with this, however, is that there is no guarantee that such changes will be confined to a 's beliefs about b . We can easily have both $w_0 R_{a,i} w$ and $w_0 R_{b,i} w$. In this case, if we change $W_{b,i,w}$ we are also changing b 's beliefs. But intuitively, when we vary a 's beliefs about b , we want to hold b 's actual beliefs constant. So we need to exclude cases in which for some w , $w_0 R_{a,i} w$ and $w_0 R_{b,i} w$.

More generally, we need to look at chains of related worlds originating at w_0 . That is, we need to ensure that $W_{w_0,a,i}^*$ and $W_{w_0,b,j}^*$ are disjoint, where $W_{w_0,a,i}^*$ and $W_{w_0,b,j}^*$ are defined below in Definition 15.

Definition 14 ($W_{w,\alpha,i}$ (where $\alpha \in \{a, b\}$).) $W_{w,\alpha,i} = \{w' : w R_{\alpha,i} w'\}$.

Definition 15 ($W_{w,\alpha,i}^*$) $W_{w,\alpha,i}^*$ IS the smallest subset of W such that (1) $W_{w,\alpha,i} \subseteq W_{w,\alpha,i}^*$ and (2) if $w_1 \in W_{w,\alpha,i}^*$ and $w_1 R_{\beta,j} w_2$ for any agent-subagent pair (β,j) , where $\beta \in \mathcal{A}, \beta \neq \alpha$, then $w_2 \in W_{w,\alpha,i}^*$.

We arrive at the following definition of a *separated* model by applying the disjointness requirement to all differing agent-subagent pairs; we also require that $w_0 \notin W_{w_0,a,i}^*$ for all agent-subagent pairs (a,i) .

Definition 16 (Separated model.) Let $\mathcal{M} = \langle M, w_0, i, a \rangle$ be a model on a frame $\mathcal{F} = \langle W, S \rangle$ for a language $\mathcal{L} = \langle \mathcal{P}, \mathcal{E}, \mathcal{A}, \mathcal{I}, \preceq \rangle$. \mathcal{M} is *separated* iff (1) for all agent-subagent pairs $(b,j), (b',j') \in \mathcal{A} \times \mathcal{I}$ such that $(b,j) \neq (b',j')$, $W_{w_0,b,j}^* \cap W_{w_0,b',j'}^* = \emptyset$ and (2) $w_0 \notin W_{w_0,b,j}^*$ for all agent-subagent pairs (b,j) .

It is easy to show that restricting our attention to separated models will lose no generality; for every model, there is a separated model that satisfies exactly the same formulas. The construction that proves this fact involves “cloning” worlds, replacing each $w \in W_{w_0,b,j}^*$ with a copy $\langle w, b, j \rangle$ indexed to (b,j) .

Theorem 1 For every model $\mathcal{M} = \langle M, w_0, a, i \rangle$ of an inter-agent propositional language \mathcal{L} with mutual belief on a frame $\mathcal{F} = \langle W, D \rangle$, there is a separated model $\mathcal{M}' = \langle M', w_0, a, i \rangle$ of \mathcal{L} on $\mathcal{F}' = \langle W', D \rangle$ such that for all formulas A of \mathcal{L} , $\mathcal{M} \models A$ iff $\mathcal{M}' \models A$.

As I noted above in Section 6, this result fails, for standard models at least, if propositional quantifiers are added.

We are now in a position to define the apparatus that, using circumscription, will provide a nonmonotonic consequence relation. Suppose that we have a group \mathcal{G} of agents, and we are interested in the beliefs of the members of some subgroup \mathcal{G}' of \mathcal{G} about the beliefs of the members of \mathcal{G} . To locate the appropriate beliefs, we fix some subagent index i ; in general, this will be an index that is taken to be public for \mathcal{G} by the members of \mathcal{G} . Finally, suppose that we are working with a set \mathbf{Ab} of abnormality predicates which we wish to

minimize, with respect to the beliefs of the i -subagents of members of \mathcal{G}' about the members of \mathcal{G} .

We begin with a separated model \mathcal{M} . First, we identify a class of models (not necessarily separated) obtained from \mathcal{M} by allowing abnormalities and the beliefs of the i -subagents of \mathcal{G}' about the beliefs of other members of \mathcal{G} to vary freely. Within this class, we prefer models with fewer abnormalities. We then define the formulas circumspectively implied by a theory T by restricting our attention to minimal models of T , relative to this preferential ordering. All this is made precise in the following definitions.

Definition 17 ($\mathcal{M}_1 \cong_{\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i} \mathcal{M}_2$) Let $\mathcal{G}' \subseteq \mathcal{G} \subseteq \mathcal{A}$. Let $\mathcal{M}_1 = \langle M_1, w_0, a, i \rangle$ and $\mathcal{M}_2 = \langle M_2, w_0, a, i \rangle$ be separated models on the same frame such that M_1 and M_2 are alike except on a set \mathbf{Ab} of predicate constants and on the set of modalities $\{\llbracket b, j \rrbracket : b \in \mathcal{G}'\}$. Let $R_{b,k}^1 = \llbracket \llbracket b, k \rrbracket \rrbracket_{\mathcal{M}_1}$, $R_{b,k}^2 = \llbracket \llbracket b, k \rrbracket \rrbracket_{\mathcal{M}_2}$. Then $\mathcal{M}_1 \cong_{\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i} \mathcal{M}_2$ iff the following holds: $w R_{b,k}^1 w'$ iff $w R_{b,k}^2 w'$ unless for some $a \in \mathcal{G}'$, $w R_{a,j}^1 w$, $j = k$, $b \neq a$, and $b \in \mathcal{G}$.

Definition 18 ($\mathcal{M}_1 \leq_{\mathbf{Ab}} \mathcal{M}_2$) Let $\mathcal{M}_1 = \langle M_1, w_0, a, i \rangle$ and $\mathcal{M}_2 = \langle M_2, w_0, a, i \rangle$ be models on the same frame. Let \mathbf{Ab} be a set of predicate constants. Then $\mathcal{M}_1 \leq_{\mathbf{Ab}} \mathcal{M}_2$ iff for all $P \in \mathbf{Ab}$, $\llbracket P \rrbracket_{\mathcal{M}_1}(w_0) \subseteq \llbracket P \rrbracket_{\mathcal{M}_2}(w_0)$.

Definition 19 (Ab, G, G', i-minimality for T.) Let T be a set of formulas of $\mathcal{L} = \langle \mathcal{B}, \mathcal{I}, \mathcal{A}, \preceq, \text{MUT} \rangle$, and $\mathcal{M} = \langle M, w_0, a, i \rangle$ be a model of T . \mathcal{M} is $\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i$ -minimal for T iff (1) for some separated model $\mathcal{M}_1 = \langle M_1, w_0, a, i \rangle$, $\mathcal{M} \cong_{\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i} \mathcal{M}_1$ and (2) for all models $\mathcal{M}' = \langle M', w_0, a, i \rangle$ of T such that $\mathcal{M}' \cong_{\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i} \mathcal{M}_1$, $\mathcal{M} \leq_{\mathbf{Ab}} \mathcal{M}'$.

Definition 20 ($\llbracket \sim_{\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i} \cdot \rrbracket$) Let $\mathcal{G}' \subseteq \mathcal{G} \subseteq \mathcal{A}$. Let T be a set of formulas of $\mathcal{L} = \langle \mathcal{B}, \mathcal{I}, \mathcal{A}, \preceq, \text{MUT} \rangle$. Then $T \llbracket \sim_{\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i} A \rrbracket$ iff $\mathcal{M} \models A$ for all models \mathcal{M} that are $\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i$ -minimal for T .

8 ACHIEVING MUTUALITY THROUGH NONMONOTONIC REASONING

8.1 SIMPLIFICATIONS

We begin by simplifying things. First, let's assume that there are only two agents. Second, the language is not only homogeneous, but in fact each agent has only one minimal subagent i_0 (which is also public). So there are only two agent modalities: $\llbracket a, i_0 \rrbracket = \llbracket a, \text{PUB} \rrbracket$ and $\llbracket b, i_0 \rrbracket = \llbracket b, \text{PUB} \rrbracket$. And we confine ourselves to frames with $D = \mathcal{A}$, i.e. the only individuals are agents.

We will only be interested in two abnormality predicates: the predicates Ab_1^a and Ab_1^b given in Section 8.2, below. And we will only want to circumscribe in three ways: (1) minimizing Ab_1^a while allowing a 's beliefs about b to vary, (2) minimizing Ab_1^b while allowing b 's beliefs about a to vary, and (3) minimizing Ab_1^a and Ab_1^b while allowing a 's beliefs about b and b 's beliefs about a to vary.

In case (1), we are circumscribing only with respect to a 's beliefs about b ; in case (2), we are circumscribing only with respect to b 's beliefs about a ; in case (3), we are circumscribing with respect to a 's and b 's beliefs about each other.

This enables us to simplify our notation for circumscription. The following definitions provide for the three types of circumscription in which we are interested.

Definition 21 ($\mathcal{M}_1 \cong_a \mathcal{M}_2$, $\mathcal{M}_1 \leq_a \mathcal{M}_2$, a -minimality for T , \Vdash_a) Let $\mathbf{Ab} = \{Ab_1^a, Ab_2^a\}$, $\mathcal{G}' = \{a\}$, $\mathcal{G} = \{a, b\}$, and $i = i_0$. Then:

- (1) $\mathcal{M}_1 \cong_a \mathcal{M}_2$ iff $\mathcal{M}_1 \cong_{\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i} \mathcal{M}_2$;
- (2) $\mathcal{M}_1 \leq_a \mathcal{M}_2$ iff $\mathcal{M}_1 \leq_{\mathbf{Ab}} \mathcal{M}_2$;
- (3) \mathcal{M} is a -minimal for T iff \mathcal{M} is $\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i$ -minimal for T ;
- (4) $T \Vdash_a A$ iff $T \Vdash_{\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i} A$.

Definition 22 ($\mathcal{M}_1 \cong_b \mathcal{M}_2$, $\mathcal{M}_1 \leq_b \mathcal{M}_2$, \leq_b -minimality for T , \Vdash_b) Let $\mathbf{Ab} = \{Ab_1^b, Ab_2^b\}$, $\mathcal{G}' = \{b\}$, $\mathcal{G} = \{a, b\}$, and $i = i_0$. Then:

- (1) $\mathcal{M}_1 \cong_b \mathcal{M}_2$ iff $\mathcal{M}_1 \cong_{\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i} \mathcal{M}_2$;
- (2) $\mathcal{M}_1 \leq_b \mathcal{M}_2$ iff $\mathcal{M}_1 \leq_{\mathbf{Ab}} \mathcal{M}_2$;
- (3) \mathcal{M} is b -minimal for T iff \mathcal{M} is $\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i$ -minimal for T ;
- (4) $T \Vdash_b A$ iff $T \Vdash_{\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i} A$.

Definition 23 ($\mathcal{M}_1 \cong_{a,b} \mathcal{M}_2$, $\mathcal{M}_1 \leq_{a,b} \mathcal{M}_2$, $\leq_{a,b}$ -minimality for T , $\Vdash_{a,b}$)

Let $\mathbf{Ab} = \{Ab_1^a, Ab_2^a, Ab_1^b, Ab_2^b\}$, $\mathcal{G}' = \mathcal{G} = \{a, b\}$, and $i = i_0$. Then:

- (1) $\mathcal{M}_1 \cong_{a,b} \mathcal{M}_2$ iff $\mathcal{M}_1 \cong_{\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i} \mathcal{M}_2$;
- (2) $\mathcal{M}_1 \leq_{a,b} \mathcal{M}_2$ iff $\mathcal{M}_1 \leq_{\mathbf{Ab}} \mathcal{M}_2$;
- (3) \mathcal{M} is a, b -minimal for T iff \mathcal{M} is $\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i$ -minimal for T ;
- (4) $T \Vdash_{a,b} A$ iff $T \Vdash_{\mathbf{Ab}, \mathcal{G}, \mathcal{G}', i} A$.

We can also simplify models. In this case, where there is only one subagent per agent, satisfaction in a model does not depend on a choice of any particular subagent. So instead of treating models as structures $\langle M, w_0, a, i \rangle$, we can simplify to $\langle M, w_0 \rangle$.

Together, these simplifications make it much easier to present the basic results; and I do not believe that they lose any significant generality.

8.2 EPISTEMIC TRANSFER AXIOM SCHEMATA

Since each agent has only one subagent, we can simplify the notation for modalities: for instance, we let $[a] = [a, \text{PUB}]$.

The following epistemic transfer axiom schemata provide an incremental approach to the reasoning that underlies mutuality.

- (Tr₁^a) $[[a]A \wedge \neg Ab_1^a(A)] \rightarrow [a][b]A$
- (Tr₂^a) $[[a][b]A \wedge \neg Ab_2^a(A)] \rightarrow [a]A$

$$\begin{aligned} (\text{Tr}_1^a) \quad & [\![b]\!]A \wedge \neg Ab_1^a(A) \rightarrow [\![b]\!][\![a]\!]A \\ (\text{Tr}_2^a) \quad & [\![b]\!][\![a]\!]A \wedge \neg Ab_2^a(A) \rightarrow [\![b]\!]A \end{aligned}$$

According to Axiom Schema (Tr_1^a) , a normally believes that b believes whatever a believes; Axiom Schema (Tr_1^b) says the corresponding thing about b . The converse axiom schemata (Tr_2^a) and (Tr_2^b) are not needed to show that mutuality (or belief in mutuality) can be obtained in simple cases, but can be useful in formalizing more complex cases. (For an example, see the discussion of Case 3 in Section 9, below.)

The interpretation of the transfer schemata is potentially confusing, and needs to be clarified. I do not require that these schemata are believed; it is compatible with the schemata, for instance, that for some formulas A , $\langle a \rangle [\![a]\!]A \wedge \neg Ab_1^a(A) \wedge \neg [\![a]\!][\![b]\!]A$ is true. The transfer schemata require that an abnormality condition must fail for their conclusions to follow; they do not require that the condition should be *believed* to fail. All this makes sense if we think of the transfer schemata not as internal principles that the agents themselves reason with, but as external constraints imposed by an agent designer on the epistemic makeup of agents. The case in which agents are themselves able to reason about abnormalities and transfer constraints is certainly worth considering, but it is more complicated, and the transfer schemata I consider here do not provide a logical model of this reflective case.

I now show that if the extension of $Ab_1^a(A)$ is empty for all A , Axiom Schema (Tr_1^a) implies that whatever a believes, a also believes to be mutual. (And similarly, of course, for Axiom Schema (Tr_1^b) and b .) I will use ' Tr_1^a ', etc. to denote not only the schemata given above, but the corresponding sets of formulas.

8.3 A RESULT ABOUT MUTUAL BELIEF

Let $\text{Normal}_{a,1} = \{\neg Ab_1^a(A) : A \text{ a formula}\}$. Consider the following rule concerning mutual belief.

(NMB) From $\text{Tr}_1^a \cup \text{Normal}_{a,1}$ infer $[\![a]\!]A \rightarrow [\![a]\!][\text{MUT}]A$

Lemma 4 The rule (NMB) is valid.

Proof. Suppose that \mathcal{M} satisfies Tr_1^a , $\text{Normal}_{a,1}$, and $[\![a]\!]A$.

We show by induction on length of α that for all strings α over $\{a, b\}$, $\mathcal{M} \models [\![a\alpha]\!]A$. By hypothesis, $\mathcal{M} \models [\![a\alpha]\!]A$ when $\alpha = \phi$. Suppose that $\mathcal{M} \models [\![a\alpha]\!]A$. Note that $\text{Tr}_1^a \cup \text{Normal}_{a,1}$ implies $[\![a]\!]B \rightarrow [\![ab]\!]B$, for all formulas B . So, in particular, $\text{Tr}_1^a \cup \text{Normal}_{a,1}$ implies $[\![a\alpha]\!]A \rightarrow [\![ab\alpha]\!]A$. So $\mathcal{M} \models [\![ab\alpha]\!]A$. Also, by the modal logic of $[\![a]\!]$, $\mathcal{M} \models [\![aaa]\!]A$. This completes the induction. It now follows from Lemma 2 that $\mathcal{M} \models [\![a]\!][\text{MUT}]p$.

We now show how Lemma 4 allows mutuality to be secured by default.

8.4 MUTUALITY BY DEFAULT

The theorems in this section show that the transfer schemata ensure that, under fairly general conditions, agents will by default believe that their own

beliefs are mutual. This does not, of course, imply that the beliefs of agents will in fact, even by default, be mutual. And this is not something we should expect to prove. Suppose, for example, that a and b , while standing side by side, read a poster saying that a lecture will be given at 9, believing what they read. Also suppose that the transfer axiom schemata hold for these agents. But a is a morning person, and believes that the lecture will be given at 9am, while b is a night person, and believes the lecture will be given at 9pm. Then (assuming that neither agent is aware of a relevant abnormality, and in particular has not noticed the ambiguity in the poster) each agent will believe that their beliefs are mutual, but this belief will be incorrect. The agents will be *uncoordinated*, in the sense that their mutually modeling public modules will in fact differ.

The transfer axiom schemata allow cases of this kind to occur without any concomitant abnormalities. That is, although epistemic transfer creates defaults about agent's beliefs about what each other believe, it creates no defaults about the coordination of agents. To put it another way, the transfer axiom schemata only apply to what agents believe about one another. They do not apply to what a third party should believe about the agents' beliefs about the world, so they will not enable us to infer epistemic coordination of a group by default. There are, of course, circumstances under which a third party could have reason to suppose that a group of agents is coordinated, but I will not attempt to formalize these here.

The following result establishes moderately general conditions under which agents satisfying the local epistemic transfer axiom schemata will believe (publicly) that their public beliefs are mutual.¹⁴

Lemma 5 Let $T = T_1 \cup T_2 \cup T_3 \cup T_4 \cup T_5 \cup T_6 \cup T_7$, where:

1. T_1 consists of all instances of the epistemic transfer axiom schemata for agent a : $T_1 = \text{Tr}_1^a \cup \text{Tr}_2^a$.
2. T_2 consists of boolean formulas (i.e., of formulas that contain no modal operators) that contain no occurrences of Ab_1^a or of Ab_2^a .
3. T_3 consists of formulas having the form $[a]A$, where A is boolean.
4. T_4 consists of formulas having the form $\langle a \rangle A$, where A is boolean.
5. T_5 consists of formulas having the form $[b]A$, where A is boolean.
6. T_6 consists of formulas having the form $\langle b \rangle A$, where A is boolean.
7. T_7 consists of formulas having the form $[a]\text{MUT}A$, where A is boolean.

Then $T \Vdash_a \neg Ab_1^a(A)$ for all formulas A .

¹⁴The conditions are actually not general enough to cover many realistic cases. In particular, the restriction on T_2 to formulas containing no occurrences of Ab_1^a or of Ab_2^a makes the result inapplicable to theories that contain plausible axioms about abnormalities. But the technique used to prove the result seems to generalize, and I believe it can be used to cover many realistic cases involving an abnormality theory. I do not yet have a general result covering a suitably large class of these cases.

Proof. We show that for all separated models $\mathcal{M} = \langle M, w_0 \rangle$ of T on frame $\mathcal{F} = \langle W, D \rangle$ there is a model $\mathcal{M}' = \langle M', w_0 \rangle$ on $\mathcal{F} = \langle W, D \rangle$ such that \mathcal{M} simultaneously satisfies T , $\mathcal{M}' \cong_a \mathcal{M}$ and $\llbracket Ab_1^a \rrbracket_{\mathcal{M}'}(w_0) = \emptyset$.

Let $R_a = \llbracket [a, \text{PUB}] \rrbracket_{\mathcal{M}}$ and $R_b = \llbracket [b, \text{PUB}] \rrbracket_{\mathcal{M}}$. Similarly, $R'_a = \llbracket [a, \text{PUB}] \rrbracket_{\mathcal{M}'}$ and $R'_b = \llbracket [b, \text{PUB}] \rrbracket_{\mathcal{M}'}$. And let $W_{w,a} = W_{a,i_a} = \{w' : wR_{a,i_a}w'\}$, $W_{w,b} = W_{b,i_b} = \{w' : wR_{b,i_b}w'\}$. Define \mathcal{M}' by (i) letting $w_1 R_b w_2$ iff $w_1 R_a w_2$ for $w_1 \in W_{w_0}$, (ii) letting $\llbracket Ab_1^a \rrbracket_{\mathcal{M}'}(w_0) = \llbracket Ab_2^a \rrbracket_{\mathcal{M}'}(w_0) = \emptyset$, and letting \mathcal{M}' agree with \mathcal{M} elsewhere. (What we are doing here is replacing a 's beliefs about b with a 's beliefs about a 's own beliefs.)

Remark (i). For all boolean formulas B not containing occurrences of Ab_1^a or Ab_2^a , $\mathcal{M} \models_{w_0} B$ if and only if $\mathcal{M}' \models_{w_0} B$.

Remark (ii). For all boolean formulas B and for all $w \in W$, if $w \neq w_0$ then $\mathcal{M} \models_w B$ if and only if $\mathcal{M}' \models_w B$.

It is straightforward to prove these by induction on the complexity of B . The proofs make essential use of the assumption that \mathcal{M} is separated.

We now establish the following claims about \mathcal{M}' :

- (1.1) \mathcal{M}' simultaneously satisfies Tr_1^a .
- (1.2) \mathcal{M}' simultaneously satisfies Tr_2^a .
- (2) \mathcal{M}' simultaneously satisfies T_2 .
- (3) \mathcal{M}' simultaneously satisfies T_3 .
- (4) \mathcal{M}' simultaneously satisfies T_4 .
- (5) \mathcal{M}' simultaneously satisfies T_5 .
- (6) \mathcal{M}' simultaneously satisfies T_6 .
- (7) \mathcal{M}' simultaneously satisfies T_7 .

Proof of (1.1). Let $A \in \text{Tr}_1^a$. Then for some B , A is $[\alpha]B \wedge \neg Ab_1^a(B) \rightarrow [\alpha][\beta]B$. Suppose that $\mathcal{M}' \models_{w_0} [\alpha]B$. Let $w_0 R'_a w_1$ and $w_1 R'_b w_2$. Then $w_0 R_a w_1$ and $w_1 R_a w_2$. So $w_0 R_a w_2$, and therefore $w_0 R'_a w_2$, so $\mathcal{M}' \models_{w_2} B$. Therefore, $\mathcal{M}' \models_{w_0} [\alpha]B \rightarrow [\alpha][\beta]B$.

Proof of (1.2) Let $A \in \text{Tr}_2^a$. Then for some B , A is $[\alpha][\beta]B \wedge \neg Ab_1^a(B) \rightarrow [\alpha]B$. Suppose that $\mathcal{M}' \models_{w_0} [\alpha][\beta]B$. Let $w_0 R'_a w_1$. Then $w_0 R_a w_1$, and (by Euclideanness), $w_1 R_a w_1$. Therefore, $w_1 R'_b w_1$, so $\mathcal{M}' \models_{w_1} B$. Therefore, $\mathcal{M}' \models_{w_0} [\alpha]B$.

Proof of (2). It follows directly from Remark (i) that \mathcal{M}' simultaneously satisfies T_2 .

Proof of (3). Suppose $A \in T_3$; then A is $[\alpha]B$. Let $w_0 R'_a w_1$; then $w_0 R_a w_1$. Now, $\mathcal{M} \models_{w_0} A$, so $\mathcal{M} \models_{w_1} B$. By Remark (ii), $\mathcal{M}' \models_{w_1} B$. Therefore, $\mathcal{M}' \models_{w_0} [\alpha]B$, i.e., $\mathcal{M}' \models_{w_0} A$.

Proof of (4). Suppose $A \in T_4$; then A is $\langle \alpha \rangle B$. Now, $\mathcal{M} \models_{w_0} A$, so for some w_1 , $w_0 R_a w_1$, $\mathcal{M} \models_{w_1} B$. By Remark (ii), $\mathcal{M}' \models_{w_1} B$. Therefore, $\mathcal{M}' \models_{w_0} \langle \alpha \rangle B$, i.e., $\mathcal{M}' \models_{w_0} A$.

Proof of (5). Suppose $A \in T_5$; then A is $[\beta]B$. Now, $\mathcal{M} \models_{w_0} A$, so $\mathcal{M} \models_{w_0} [\beta]B$. Let $w_0 R'_b w_1$. Then $w_0 R_b w_1$, so $\mathcal{M} \models_{w_1} B$, and by Remark (ii) we have $\mathcal{M}' \models_{w_1} B$. Therefore, $\mathcal{M}' \models_{w_0} [\beta]B$, i.e., $\mathcal{M}' \models_{w_0} [\beta]A$.

Proof of (6). Suppose $A \in T_6$; then A is $\langle b \rangle B$. We have $\mathcal{M} \models \langle b \rangle B$, so for some w_1 such that $w_0 R_b w_1$, $\mathcal{M} \models_{w_1} B$. Then $w_0 R'_b w_1$ and by Remark (ii), $\mathcal{M}' \models_{w_1} B$.

Proof of (7). Let $R_*(w)$ be the transitive closure of $\{w\}$ under R_a and R_b , and $R'_*(w)$ be the transitive closure of $\{w\}$ under R'_a and R'_b . Suppose that $A \in T_7$; then A is $[a][\text{MUT}]B$. Now, $\mathcal{M} \models_{w_0} A$, so for all w_1 such that $w_0 R_a w_1$, we have $\mathcal{M} \models_{w_2} B$ for all w_2 such that $w_1 R_* w_2$. Let $w_0 R'_a w'_1$ and let $w'_1 R'_* w'_2$. Then $w_0 R'_a w'_1$ and $w'_1 R_* w'_2$, so $\mathcal{M} \models_{w'_2} B$. By Remark (ii), $\mathcal{M}' \models_{w'_2} B$. Therefore, $\mathcal{M} \models_{w_0} [a][\text{MUT}]B$.

By construction, $\llbracket Ab_1^a \rrbracket = \llbracket Ab_2^a \rrbracket = \emptyset$. In view of (1)-(7), \mathcal{M}' is a model of T . Therefore, \mathcal{M}' is a minimal model of T .

Theorem 2 Let T be as in Lemma 5. Then $T \Vdash_a [a]A \rightarrow [a][\text{MUT}]A$, for all formulas A .

Proof. In view of Lemma 5, (1) $T \Vdash_a A$, for all formulas $A \in \text{Tr}_1^a$ and (2) $T \Vdash_a A$, for all formulas $A \in \text{Normal}_{a,1}$. Then by Lemma 4, $T \Vdash_a [a]A \rightarrow [a][\text{MUT}]A$.

Theorem 3 Let T be as in Lemma 5, except that T_1 consists of all instances of the epistemic transfer axiom schemata for agent b , and T_2 consists of boolean formulas that contain no occurrences of Ab_1^b or Ab_2^b . Then $T \Vdash_b [b]A \rightarrow [b][\text{MUT}]A$, for all formulas A .

Proof. Just like that of Theorem 2.

Theorem 4 Let T be as in Lemma 5, except that T_1 consists of all instances of the epistemic transfer axiom schemata for agents a and b , and T_2 consists of boolean formulas that contain no occurrences of Ab_1^a , Ab_2^a , Ab_1^b or Ab_2^b . Then $T \Vdash_{a,b} ([a]A \wedge [b]A) \rightarrow [\text{MUT}]A$, for all formulas A .

Proof. Combine the constructions used in the proofs of Theorem 2 and Theorem 3, and use Lemma 3 to establish the result.

9 AN EXAMPLE

Belief transfer is fallible, and is recognized as such in everyday cases of reasoning about belief. So it is important to provide a means of formalizing the circumstances under which the reasoning that governs belief transfer will be blocked.

The example I'll develop in this section resembles the one at the beginning of (Clark and Marshall, 1981) which shows that the iteration complexity for agent knowledge can reach fairly high finite levels. That example, though, deals simply with agent beliefs. As I explained in Section 2.2.2, I feel that agent beliefs are the wrong attitudes to use when mutuality is at stake. Instead, we need "public" attitudes that are invoked specifically to model other agents.

As usual, there are two agents, a and b in the following example. We distinguish between their private beliefs and the beliefs that they expect to be public in a conversation they are having along a potentially faulty communication channel. Each agent has two subagents, ROOT and PUB. The former represents the sum of the agent's beliefs and the latter represents the belief module that is devoted to tracking the conversation. We have $\text{ROOT} \preceq \text{PUB}$, but $\text{PUB} \not\preceq \text{ROOT}$.

The following rudimentary theory of email communication consists of three parts: (A) protocols for updating the contents of $[a, \text{PUB}]$, (B) a theory of exceptions to the protocols in (1), and (C) the transfer axiom schemata (Tr_1^a) and (Tr_2^a) . To keep things simple, the formalization ignores temporal considerations. The following axiom schemata use quantification over individuals (but not over propositions).

(A) Protocols for updating $[a, \text{PUB}]$

$$(A.1) \forall m [[\text{Send}(a, b, m) \wedge \text{Incontents}(m, A)] \rightarrow [a, \text{PUB}]A]$$

If a sends a message to b that says p (where, as before, A expresses the proposition p), then a adds p to $[a, \text{PUB}]$.

$$(A.2) \forall m_1 \forall m_2 [[\text{Send}(a, b, m_1) \wedge \text{Incontents}(m_1, A) \wedge \text{Read}(a, m_2) \\ \wedge \text{sender}(m_2) = b \wedge \text{Ack}(m_2, m_1)] \rightarrow [a, \text{PUB}][b, \text{PUB}]A]$$

If a sends a message to b that says p and reads an acknowledgement of that message from b then a adds $[b, \text{PUB}]p$ to $[a, \text{PUB}]$.

Notes on the formalization. Neither (A.1) nor (A.2) is a default. The nonmonotonic aspects of the reasoning about belief are handled by the epistemic transfer rules. This division of labor seems simpler on the whole; it has the defect that the modality $[a, \text{PUB}]$ no longer represents a 's view of b 's beliefs. Since other modalities, such as $[a, \text{PUB}][b, \text{PUB}]$ and $[\text{MUT}, \text{PUB}]$ are available, this is not much of a disadvantage. In order to prove that mutuality is secured by default, we may wish to restrict the values of ' A ' in these schemata to boolean formulas that do not contain abnormality predicates.

(B) The abnormality theory.

$$(B.1) \forall m_1 \forall m_2 [[\text{Send}(a, b, m_1) \wedge \text{Incontents}(m_1, A) \wedge \text{Read}(a, m_2) \\ \wedge \text{Not-Delivered}(m_2, m_1)] \rightarrow Ab_1^a(A)]$$

If a sends a message to b saying that p and reads a message saying the message was not delivered then the conditions for inferring $[b, \text{PUB}]A$ are blocked.

$$(B.2) Ab_1^a(A) \rightarrow Ab_1^a([a, \text{PUB}]A)$$

If a proposition is abnormal, so is the proposition that a publicly believes it. (This is needed for technical reasons; without it, $[a, \text{PUB}][\text{MUT}]MB$ could be inferred in Case 2, below.) The next schema is similar.

$$(B.3) Ab_2^a(A) \rightarrow Ab_2^a([a, \text{PUB}]A)$$

(C) The transfer axiom schemata.

$$(\text{Tr}_1^a) [\neg Ab_1^a(A) \wedge [a, \text{PUB}]A] \rightarrow [a, \text{PUB}][b, \text{PUB}]A$$

$$(\text{Tr}_2) \quad [\neg Ab_2^S(A) \wedge [a, \text{PUB}] [b, \text{PUB}] A] \rightarrow [a, \text{PUB}] A$$

I will present four cases, of increasing complexity.

Case 1. The story: Ann and Bob correspond regularly and normally by email. Ann sends Bob the following message, M_1 . Nothing unusual happens.

To: Bob <robert@xyz.org>
 From: Ann <ann@abc.org>
 Subject: Movies at the Roxie

Bob,
 Monkey Business is showing tonight at the Roxie.
 Ann

The reasoning: Since her communications with Bob are normally successful, Ann assumes that this one is successful, and in fact it has much the same status for her that face-to-face conversation does. Ann maintains a subagent to keep track of beliefs that are *prima facie* shared with Bob. On sending M_1 , she adds the contents of the message to the beliefs of this subagent, i.e., MB is added to $[a, \text{PUB}]$.

Here, we want $[a, \text{PUB}] [\text{MUT}] MB$ to be a circumspective consequence of the theory—we want the theory to imply that Ann (publicly) believes that the content of the message is (publicly) mutually believed. Theorem 2 does not suffice to prove this, since now a part of the theory deals with abnormality. A construction similar to the one used in the proof of Lemma 5 provides the desired result. The presence of an abnormality theory provides one further complication—Definition 21 needs to be amended so that the predicates on which abnormalities depend, *Send*, *Incontents*, *Read*, *Not-Delivered*, and *Ack*, are varied in circumscribing for Ann’s beliefs about Bob’s beliefs. Allowing these predicates to vary arbitrarily is somewhat implausible, and it may well be necessary to appeal to a more powerful circumspective technique, such as pointwise circumscription (see (Lifschitz, 1986)). The possible need for such techniques is one side effect of the complexity of these examples.

The formalization:

Initial Conditions: $\text{Send}(a, b, M_1, e_1)$, $\text{Incontents}(M_1, MB)$

Monotonic consequence: $[a, \text{PUB}] MB$

Circumspective consequence: $[a, \text{PUB}] [\text{MUT}] MB$.

Case 2. The story: Ann sends the following message, M_2 , to Bob.

To: Bob <bob@xyz.org>
 From: Ann <ann@abc.org>
 Subject: Movies at the Roxie

Bob,
 Monkey Business is showing tonight at the Roxie.
 Ann

Immediately afterwards, she receives the following message, M_3 . She says to herself “Oops, I misaddressed the message.”

To: ann@abc.org
 From: mailer-daemon@xyz.org
 Subject: Undeliverable Mail

The following errors occurred when trying to deliver the attached mail:

bob: User unknown

The reasoning: As in Case 1, Ann adds the contents of M_1 to the beliefs of the subagent representing *prima facie* beliefs shared with Bob. However, the receipt of the mailer daemon's message precipitates an anomaly, which in turn blocks any ascription of this belief to Bob.

The formalization:

Initial Conditions:

$\text{Send}(a, b, M_2, e_1)$, $\text{Incontents}(M_2, MB)$,
 $\text{Read}(a, M_3)$, $\text{Not-Delivered}(M_3, M_2)$

Consequences: $[a, \text{PUB}]MB$ is a consequence, but $[a, \text{PUB}][b, \text{PUB}]MB$ is not.

Case 3. The story: Ann sends the misaddressed message M_2 to Bob. She receives error message M_3 from the mailer daemon. Shortly after that, she receives the following message, M_4 , from Bob.

To: ann@abc.org
 From: Bob <robert@xyz.org>
 Subject: re: Movies at the Roxie

Ann,

We just rigged the mailer here to send me blind copies of messages to bob@xyz.org, so actually I got your message about Monkey Business.

Bob

She realizes that despite the mailer daemon message, Bob has received a copy of M_2 .

The reasoning:

As in Cases 1 and 2, Ann adds MB to the beliefs of the subagent representing *prima facie* beliefs shared with Bob. As in Case 2, receipt of the mailer daemon's message precipitates an abnormality of the form $Ab_1^a(MB)$, which blocks the usual chain of inference to $[b, \text{PUB}]MB$. But Bob's acknowledgement provides direct evidence for $[b, \text{PUB}]MB$; the protocol axiom (A.2), permits $[a, \text{PUB}][b, \text{PUB}]MB$ to be inferred without using (Tr_1^a) .

Using techniques from the proof Lemma 4, I believe that we can show that under these circumstances, $[a, \text{PUB}][\text{MUT}, \text{PUB}]MB$ can be inferred, provided that we also have $\neg Ab_1^a(MB)$ and $\neg Ab([\alpha]MB)$, for any string α containing at least one occurrence of b . To obtain Ann's belief of the message's mutuality

as a circumscriptive consequence, we would need to show that in any minimal model of the theory, these abnormalities are indeed empty. I believe this can be done, but I have not checked all the details.

Case 4. The story: Ann has just returned from a vacation. Forgetting to turn off her vacation daemon, she sends the misaddressed message M_2 to Bob. She receives error message M_3 from the mailer daemon. Shortly after that, she receives the following message, M_4 , from Bob.

To: ann@abc.org
 From: Bob <robert@xyz.org>
 Subject: re: Movies at the Roxie

Ann,

We just rigged the mailer here to send me blind copies of messages to bob@xyz.org. so actually I got your message about Monkey Business.

Bob

She realizes that Bob has received an automatic reply to M_4 from her vacation daemon saying that she is on vacation, but will answer the message as soon as she gets back.

The reasoning: This case would require the addition of an abnormality theory for Ab_1^a ; I have not provided such a theory. The desired conclusions here would include:

1. Ann (publically) believes p .
2. Ann (publically) believes that Bob (publically) believes p .

But they would not include

3. Ann (publically) believes that Bob (publically) believes that Ann (publically) believes p .

The reasoning here is about as complicated as common sense reasoning about attitudes can get. The cases in this example are meant to illustrate a sequence of increasingly complex formalization problems; this last one is best thought of as a challenge. I confess that I have not yet thought through the formalization issues for this case.

10 CONCLUSION

Previous attempts to explain interagent reasoning about attitudes do not provide plausible formalizations of the reasoning that underlies mutuality in cases that seem to require it, or provide logical resources for formalizing cases where mutuality is blocked. Unless I have missed something, the literature contains no flexible formal reasoning mechanisms for obtaining mutuality.

Many authors have suggested that mutuality somehow arises out of certain shared situations. This suggestion is flawed, since shared situations do not in general lead to mutuality—for instance, I will not treat information that I obtain from a situation I share with you as mutual if I observe that you do

not observe me sharing the situation. If we believe that mutuality is required for some purposes, then we have to exhibit a reasoning mechanism that allows agents to obtain it from information that we can plausibly expect agents to have, and that also allows us to block the reasoning in cases where mutuality should not be forthcoming.

The only way to demonstrate the viability of a theory of these mechanisms is to demonstrate their utility in formalizing a wide variety of fairly complex cases. I do not claim to have done that here, and in fact the cases considered in Section 9 put a certain degree of pressure on the circumscriptive approach that is developed in this paper. Obviously, more work needs to be done on the logical foundations, to develop an approach to the nonmonotonic reasoning that is not intolerably complex and is flexible enough to handle cases like those of Section 9. I myself am happy to be eclectic about the choice of nonmonotonic formalisms, and I believe that alternatives to the circumscriptive approach need to be explored.

I hope that at least I have made a plausible case for the promise of the general approach that is developed in this paper, and in particular that I have convinced you that immediacy theories of how mutuality is secured do not do justice to the relevant reasoning.

Of course, it is highly desirable not only to deploy nonmonotonic formalisms in applications such as this, but to show how the reasoning can be efficiently implemented in special cases. I have not addressed that question in this paper, but I hope to do this in subsequent work.

References

- Aumann, R. J. (1976). Agreeing to disagree. *Annals of Statistics*, 4(6):1236–1239.
- Barwise, K. J. (1988). Three views of common knowledge. In Vardi, M. Y., editor, *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 365–379, Los Altos, California. Morgan Kaufmann.
- Buvač, S. and Mason, I. (1993). Propositional logic of context. In Fikes, R. and Lehnert, W., editors, *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 412–419, Menlo Park, California. American Association for Artificial Intelligence, AAAI Press.
- Clark, H. (1992). *Arenas of Language Use*. University of Chicago Press, Chicago.
- Clark, H. H. and Marshall, C. R. (1981). Definite reference and mutual knowledge. In Joshi, A., Webber, B., and Sag, I., editors, *Linguistics Structure and Discourse Setting*, pages 10–63. Cambridge University Press, Cambridge, England.
- Clark, H. H. and Schober, M. (1989). Understanding by addressees and overhearers. *Cognitive Psychology*, 24:259–294. Republished in Clark, 1992.
- Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). *Reasoning about Knowledge*. The MIT Press, Cambridge, Massachusetts.
- Fine, K. (1970). Propositional quantifiers in modal logic. *Theoria*, 36:336–346.

- Guha, R. V. (1991). Contexts: a formalization and some applications. Technical Report STAN-CS-91-1399, Stanford Computer Science Department, Stanford, California.
- Halpern, J. Y. and Lakemeyer, G. (1996). Multi-agent only knowing. In Shoham, Y., editor, *Theoretical Aspects of Rationality and Knowledge: Proceedings of the Sixth Conference (TARK 1996)*, pages 251–265. Morgan Kaufmann, San Francisco.
- Kremer, P. (1997). On the complexity of propositional quantification in intuitionistic logic. *The Journal of Symbolic Logic*, 62(2):529–544.
- Kripke, S. (1965). A semantical analysis of modal logic II: Non-normal propositional calculi. In Henkin, L. and Tarski, A., editors, *The Theory of Models*, pages 206–220. North-Holland, Amsterdam.
- Lemmon, E. (1957). New foundations for Lewis modal systems. *Journal of Symbolic Logic*, 22(2):176–186.
- Lewis, D. K. (1969). *Convention: A Philosophical Study*. Harvard University Press, Cambridge, Massachusetts.
- Lifschitz, V. (1986). Pointwise circumscription. In Kehler, T. and Rosenschein, S., editors, *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 406–410, Los Altos, California. American Association for Artificial Intelligence, Morgan Kaufmann.
- McCarthy, J. (1980). Circumscription: A form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39.
- Monteiro, A. M. and Wainer, J. (1996). Preferential multi-agent nonmonotonic logics. In Aiello, L. C., Doyle, J., and Shapiro, S., editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 446–452. Morgan Kaufmann, San Francisco, California.
- Morgenstern, L. (1990). A theory of multiple agent nonmonotonic reasoning. In Dietterich, T. and Swartout, W., editors, *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 538–544, Menlo Park, CA. American Association for Artificial Intelligence, AAAI Press.
- Mortensen, C. D. (1996). *Miscommunication*. Sage Publications, Thousand Oaks, California.
- Parikh, R. (1991). Monotonic and nonmonotonic logics of knowledge. *Fundamenta Informaticae*, 15(3–4):255–274.
- Schiffer, S. (1972). *Meaning*. Oxford University Press, Oxford.
- Stalnaker, R. C. (1975). Pragmatic presuppositions. In Munitz, M. K. and Unger, P., editors, *Semantics and Philosophy*. Academic Press, New York.
- Thomason, R. H. (1990). Propagating epistemic coordination through mutual defaults I. In Parikh, R., editor, *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the Third Conference (TARK 1990)*, pages 29–39, Los Altos, California. Morgan Kaufmann.
- Thomason, R. H. (1998). Intra-agent modality and nonmonotonic epistemic logic. In Gilboa, I., editor, *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the Seventh Conference (TARK 1998)*, pages 57–69, San Francisco, California. Morgan Kaufmann.
- Wainer, J. (1993). Epistemic extension of propositional preference logics. In Bajcsy, R., editor, *Proceedings of the Thirteenth International Joint*

Conference on Artificial Intelligence, pages 382–387, San Mateo, California.
Morgan Kaufmann.

XII

LOGIC AND LANGUAGE

Chapter 18

THE SITUATIONS WE TALK ABOUT

Lenhart K. Schubert

University of Rochester

Rochester, New York 14627-0226

schubert@cs.rochester.edu

Abstract I argue in favor of associating situations (events, episodes, eventualities, etc.) with arbitrarily complex sentences, not just atomic predicates, in NL interpretation. In that respect, a Situation Semantics approach to incorporating situations into semantic representations is preferable to a Davidsonian one. However, I will further argue that beyond the notion of truth or falsity of a sentence *in* a situation, as in Situation Semantics, we also need the notion of a sentence *characterizing* a situation, in order to deal adequately with *causal* relations mentioned or implied in NL texts. I propose a way of doing this that essentially reduces complex situations to joins of basic, Davidsonian ones, along with basic situations corresponding to negated predication. The resulting situational logic, called FOL^{**}, captures many of the essential features of both Davidsonian and Situation Semantics approaches to representing the content of sentences describing situations. The proposed semantics supports common intuitions about truth-in-situations, about the existence of situations characterized by sentences, and about persistence of information from parts of situations to the whole. I allow for *temporal* parts of situations as well as concurrent parts, and distinguish persistence properties of telic and atelic sentences. The development of FOL^{**} is part of a continuing effort to fully formalize Episodic Logic, an implemented knowledge representation designed to support language understanding.

Keywords: Events, situations, eventualities, episodes, situation description, situation characterization, FOL^{**}, episodic logic, causal relations, event anaphora, Davidsonian events, negative situations, complex situations, situation semantics, event semilattice, persistence.

1 INTRODUCTION: COMPETING VIEWS ON THE RELATION BETWEEN SENTENCES AND SITUATIONS

It is routinely observed in NLP that sentences seem to "evoke" situations (where I use this term comprehensively to cover events, episodes, eventualities, processes, etc.). Much like discourse entities evoked by explicit noun phrases, these evoked situations can be referred to anaphorically, as for instance in (1) and (3) below, and can be modified in various ways, for instance by supplying their duration and location, as in (2) and (4).

1. Molly barked. *This* woke up John.
2. Molly barked for 20 minutes in the yard last night.
3. It's chilly outside. *This* is making the last of the leaves drop from the trees.
4. It has been chilly outside for several weeks.

Thus we need to incorporate situations explicitly into the logical forms of NL sentences. One of the better-known early approaches was that of Hans Reichenbach, 1947. For example, he offered the logical form (LF) in (5b) for sentence (5a):

- 5 a. George VI was crowned.
- b. $\exists e. [\text{was-crowned}(G)]^*e$ $e = \text{crowning (coronation) of } G$.

Note the use of the operator ' $[]^*$ ' for associating situations¹ with sentences. These sentences were potentially complex, and in that respect Reichenbach anticipated Situation Semantics. Another approach, which proved more influential because of its unproblematic semantics, was that of Donald Davidson, 1967b, as illustrated here.

- 6 a. George VI was crowned in Westminster Abbey
- b. $\exists e. \text{was-crowned}(G,e) \wedge \text{in}(e,W)$

Here events are simply introduced as additional arguments of event predicates. As is evident in (6b), Davidson also regarded the meaning of adverbial adjuncts as expressible by conjoined predication about the location, time, manner, etc., of the event introduced by the verb.

More recent approaches show the influences of both of these early proposals. For example, Barwise and Perry, in (Barwise and Perry, 1983) and subsequent writings, suggest meaning representations like the following (where all 3 examples correspond to (6a) and illustrate slightly different notations).

- 7 a. in e : at W : was-crowned, G ; yes
- b. $e \models << \text{was-crowned}, G, 1 >> \wedge << \text{loc}, e, W, 1 >>$
- c. $e \models \text{was-crowned}(G) \wedge \text{loc}(e,W)$

The ' \models ' relation indicates that a situation e *supports* certain facts, and as in the case of Reichenbach's ' $[]^*$ ' operator, these facts can be complex. However, while Reichenbach seems to have had in mind that e in $[\phi]^*e$ stands just for a ϕ -event, no more, no less, Situation Semantics views situations as potentially supporting many other facts besides those explicitly specified. For example, the

¹More accurately, Reichenbach had in mind events and facts, whose differences he blurred.

situation e in (7a) could perfectly well support additional facts (or “infons”) such as that there was a bat in the belfry, even if this is not considered part of George VI’s coronation. This point is particularly clear in versions of Situation Semantics that employ a more standard syntax and semantics, such as Muskens’ very tidy and general formulation (Muskens, 1995). Like Davidson, he treats situations as predicate arguments, but this syntactic similarity is deceptive. For Davidson and his adherents, $\text{was-crowned}(G, e)$ means that e is the coronation of G ; but for Muskens, it means that e is a *partial world* that happens to support the truth of G ’s being crowned, but may also support arbitrarily many other, possibly quite unrelated, facts.²

As an example of a neo-Davidsonian approach, we might mention that of Parsons, 1990. The distinctive feature of this approach is the “factoring” of n -ary event predication into a unary predication for the event type and a set of conjoined binary predication specifying the *thematic roles* of the event. For instance, (6a) might be rendered as

$$8. \exists e. \exists t. \text{crowning}(e) \wedge \text{obj}(e, G) \wedge \text{occur}(e, t) \wedge \text{past}(t) \wedge \text{loc}(e, W)$$

As in Davidson’s original approach (as well as in Reichenbach’s), e is intended to be a crowning and nothing more. Though various things can be said about it, such as that it involves George VI as its object, it is not a “partial world” supporting miscellaneous positive and negative facts.

Hwang and Schubert (e.g., Schubert and Hwang, 1989; Hwang and Schubert, 1993a; Hwang and Schubert, 1993b; Schubert and Hwang, 2000) have long maintained that logical forms for general NLU must be able to capture both sorts of relationship between sentences and situations – sentences as partial descriptions of situations, and as “characterizing” descriptions of situations. I will reiterate and strengthen the arguments for this view in the next section. Here I briefly introduce the Episodic Logic (EL) notation used in our previous work, in particular the ‘**’ and ‘*’ operators for connecting sentences with situations (episodes, etc.). (These operators are the only features of EL I will retain for the rest of the paper.) The LF for (6a) would look roughly as follows. (Square brackets indicate sentential infix syntax, with the sentence subject in first place, the predicate second, and additional arguments after the predicate.)

$$9. (\exists e: [e \text{ before Now7}] (\exists x [[G (\text{pasv crown}) x]**e] \wedge [e \text{ loc-in } W]))$$

In quantified sentences of form $(Qx: \phi \psi)$, the ϕ -formula is interpreted as a restriction on the domain of quantification, and the ψ -formula is the main, or “matrix” clause. In (9), the quantifier Q is \exists , and the restriction and matrix

²This is not to say that for Davidson, an event characterized in one way could not also be characterized in many other ways (for instance more or less abstractly, and more or less elaborately); e.g., note his discussion of ‘crossing the Channel’ and ‘swimming the Channel’, or the various ways of describing a shooting (Davidson, 1967b). In his discussion of causes (Davidson, 1967a), Davidson does move in the direction of admitting a description of a part of an event (in particular, part of a causal chain) as a description of the event as a whole (in particular, a complete causal chain). He is motivated by the observation that one can refer to different parts of a causal chain as the cause of another event. Whatever the merits of this position may be, it is still very different from saying that we can join arbitrary events to a given event without invalidating the original characterization. Furthermore AI researchers who have followed Davidson’s strategy of event-introduction have generally taken a predication $P(e, \dots)$ to mean that $P(\dots)$ describes e as a whole, not just some part of e .

clauses can be viewed as conjoined. (The restriction [e before Now7] captures the past tense information, i.e., that the event e of George VI being crowned is before the time of narration, Now7.) What is noteworthy here is first, that such LFs are easily computed from surface form, keeping the syntax-semantics interface as simple as possible; and second, that a '**' operator is used to connect the main clause with the episode it characterizes. Its intended meaning is quite similar to that of Reichenbach's '[]*' operator: given $[[G \text{ (pasv crown)} x]**e]$, e must be the event of George VI being crowned, i.e., his coronation. But EL also has another operator, '*', which is more akin to the support relation, ' \models ', of Situation Semantics. While the direct LFs of English sentences generally involve only the '**' operator, inference often leads to occurrences of the '*' operator. For instance, a consequence of (9) via meaning postulates might be

$$10. (\exists y: [y \text{ crown}] [[x \text{ place-upon} (\text{head-of } G) y]*e])$$

In other words, one of the facts supported by the coronation event is that someone placed a crown on George VI's head.³ Furthermore there may be many contingent details of this particular coronation all of which are supported by e . (Among these details may be aspects of *temporal segments* of the coronation.)

In the next section, I will make the argument for a tighter coupling between sentences and events than is provided by '*' (or the *supports* relation ' \models ' of Situation Semantics). A sufficiently tight coupling is provided by '**', or by a Davidsonian approach. However, in Section 3 I will show that ordinary language makes reference to events more complex than those that can be represented in a Davidsonian framework. This justifies the use of the '**' and '*' operators, or something like them. In Section 4, I will suggest a semantic basis for these operators, in effect showing that both can be explicated in terms of Davidsonian events (along with "negative situations") and their joins in a join semilattice. I will go on to show in Section 5 that this semantic basis supports many of the intuitively wanted properties of situations and their descriptions, including the existence of parts of situations characterized by ϕ whenever ϕ partially describes those situations; the "transparency" and truth-like character of the partial description (or support) operator '*'; and the persistence of information through the part-of ordering of situations, with different kinds of persistence corresponding to telicity or atelicity of the descriptions employed.

2 WHY WE NEED '**'

Let me begin by suggesting a few ways of paraphrasing the meaning of '**' and '*', as a way of fixing intuitions about them:

- [G be-crowned]** e e is an episode of George VI being crowned;
- '*George VI is crowned*' characterizes e
- e as a whole is of type '*George VI is crowned*';

³We could eliminate the free occurrences of e and x in (10) in favor of Skolem constants replacing e and x in (9); or else we could simply adjoin (10) to (9), relying on the dynamic binding mechanism in EL to "carry forward" values of e and x that can truthfully bind them in (9) (in effect extending the scopes of the quantifiers in (9) to include (10)) (Hwang and Schubert, 1993b).

	e is an episode that consists just of George VI being crowned
[G be-crowned]*e	e is an episode <i>in which</i> George VI is crowned; 'George VI is crowned' holds (occurs) in e
	e is (partially) described by George VI being crowned;
	e is an episode that consists in part of George VI being crowned

Now, it is commonplace in texts (particularly narratives) to find a description of some event or situation e , followed immediately by a description of another event e' that is said or implied to be caused by the first. (The sentence pairs in (1) and (3) illustrate this pattern.) So in the LFs of such causal descriptions, we will need some way of expressing that an event e of type ϕ occurred, and that this caused the subsequent event e' . From a Situation Semantics perspective (and using the *-notation), it would seem natural to represent the occurrence of an event e of type ϕ by writing $[\phi*e]$.

However, I maintain that

" e' has a cause e of type ϕ "

cannot be formalized as

" e' has a cause e satisfying $[\phi*e]$ ".

Rather, the formalization needs to use '*' (or some other event characterizing representation, such as a Davidsonian or Reichenbachian one) in place of '*'. To recognize the incorrectness of the above formalization, we observe that it can lead to incorrect conclusions in particular instances.

For example, suppose that Mary's waking up was caused by John's singing. Further suppose that John was taking a shower at the time, so it is also correct to say that Mary's waking up was caused by John's simultaneous singing and showering. I.e., though the showering may be causally irrelevant to Mary's waking up, nonetheless the larger, singing-plus-showering event is sufficient to account for Mary's waking up. But John's simultaneous singing and showering is *partially* described by '*John showers*'. In other words, there is an event e (viz., John's simultaneous singing and showering) satisfying

$[[\text{John showers}]*e]$.

But if it is correct to identify "has a cause e of type ϕ " with "has a cause e satisfying $[\phi*e]$ ", then we have reached the conclusion that Mary's waking up was caused by John's showering - surely not a valid conclusion under the conditions we have assumed.⁴

This example instantiates a general pattern. Seeing this pattern at a more abstract level should further clarify the nature of the difficulty with using '*' in causal descriptions:

⁴Of course it may be the case that if John had not taken a shower he would not have sung and so would not have awakened Mary; in that sense his taking a shower may be a contributing cause in Mary's awakening. But a contributing cause is in general something less than a cause, and in any case he might have sung even if he had not taken a shower. Thus the conclusion that John's taking a shower caused Mary to wake up is not deductively valid.

- (i) We are given events A and E such that
[A cause-of E].
- (ii) We consider some arbitrary situation or event B concurrent with A , with some partial description
[$\phi * B$].
- (iii) Let C be the combination of events A and B . (E.g., the combination of John's singing and showering, above. Instead of his taking a shower, we could have picked any other concurrent situation or event, not necessarily one involving John. While the combination C may not be a "natural" one, it still seems reasonable to assume it exists, in the logical sense.) Then ϕ is also a partial description of C , i.e.,
[$\phi * C$].
- (iv) In addition, since C includes A as a part, and A caused E , we also have
[C cause-of E].

The reader may not agree that "enlarging" a cause with an arbitrary concurrent situation or event necessarily preserves its status as a cause. But it is sufficient, for the purposes of the generalized refutation I am outlining, that there be some instances where this is reasonable (as in the singing-and-showering example above). I will pursue this point further below.

- (v) Hence, if "having a cause C such that [$\phi * C$]" can be understood as "having a cause C of type ϕ ", then we have shown that

E has a cause of type ϕ .

But this is in general absurd, since ϕ is a partial description of an arbitrary situation or event that happens to be concurrent with E 's actual cause, and need not have any causal relevance to E at all.

For those who are uncomfortable with the idea that events viewed as causes can be augmented with miscellaneous concurrent events without losing their causal status, we can frame a similar argument beginning with conjunctive causes, as in the following examples.

11. The ferry was overloaded and unstable, and this caused it to capsize.
12. John was driving and using his cellular phone, and this caused an accident.

It does not follow from (11) that it was the situation of the ferry being overloaded (or that of the ferry being unstable) that caused it to capsize (though these narrower causal claims might also be true). Nor does it follow from (12) that John's driving caused the accident in question, or that his using the cellular phone did. Yet, much as before, logical forms for (11) and (12) based on a partial-description connection '*' between sentences and the situations they evoke would lead to precisely those consequences. The only assumption this involves is that a situation partially described by '*The ferry was overloaded and*

unstable' is also partially described by '*The ferry was overloaded*', and by '*The ferry was unstable*', and similarly for (12).

So '*' (or ' \models ' in Situation Semantics) is simply too "unselective" to be able to characterize causes. The truth of $\phi * e$ fails to ensure that e consists *just* of an episode of type ϕ . Instead, it may have parts or aspects of many different types, so in saying that an event e satisfying $\phi * e$ is the cause of e' , we are failing to say that it is the " ϕ -part" of e that is causally effective. We need a tighter coupling between sentences and events, and that is what the '**' operator provides.

Now in fact a Davidsonian representation also provides a sufficiently tight coupling between sentences and events for characterizing causes. For example, to express that John's singing woke up Mary, we might write (neglecting tense)

$\exists e. \exists e'. \text{sing}(e, \text{John}) \wedge \text{wake-up}(e', \text{Mary}) \wedge \text{cause}(e, e')$,

where we understand e to be (just) an event of John singing, and e' (just) an event of Mary waking up. In contrast with the case of '*'- (or ' \models '-) based causal descriptions, there is no allowance for e to be something more than an event of John singing, such as an event of John singing and taking a shower. So with respect to causal description, a Davidsonian approach is more nearly adequate than one based on a partial-description relation between sentences and events, as in Situation Semantics. But the limitation of the Davidsonian approach is its inability to describe complex events, as I now argue.

3 SITUATIONS DESCRIBED BY NEGATIVE, CONJOINED AND QUANTIFIED SENTENCES

The observations that led us to posit events in logical forms apply equally to complex descriptions as to simple ones: we can refer anaphorically to them, ascribe properties such as locations and durations to them, and adduce them as causes. Some examples of conjunctively described situations, and reference to such situations as causes, were already seen in the preceding section. The following are some further examples of anaphoric reference to complex situations, including negative, conjunctive and quantified situations. (I will use this loose terminology to refer to complex situations characterized respectively by negative, conjoined, or quantified sentences.)

13. There has been no rain for several weeks.

This has ruined the crops.

14. The evening was warm and windless.

This brought out the mosquitoes.

15. Kevin is home alone.

That's a risky situation.

16. Each graduate went to the podium to receive his or her diploma.

This took half an hour.

Note the adverbial specifying the duration of the no-rain situation in (13), and the anaphoric reference to that situation. (14) and (15) (like the earlier (11) and (12)) describe and subsequently refer to conjunctive situations. Note that '*home alone*' in (15) is equivalent to '*at home while no-one else is at home*' and in that sense specifies a concurrent positive and negative situation. In

(16) we have a quantified situation, and anaphoric reference to that situation in the description of its duration. The duration could alternatively have been specified by an initial adverbial such as '*Over the next half hour*'.

(13) and (14) can also be seen as supplying causal relations involving complex causes, since the verbs '*ruined*' and '*brought out*' take an agent or a cause as subject. The following are two examples where a causal relation is explicitly asserted for a negative and a quantified cause.

17. The parachute did not deploy on schedule.

This caused the loss of the Mars lander.

18. Each superpower menaced the other with its nuclear arsenal.

This caused an escalating arms race.

Additional clear evidence for the reality and linguistic significance of complex situations is that we can (and do) explicitly refer to them as situations/ circumstances/ conditions/ states of affairs, etc. (15) is one example, and the following are two more.

19. It has not rained for several weeks.

This situation/circumstance/condition/state of affairs has caused the crops to fail.

20. *The situation of her children no longer being at home is disconcerting for Mary.*

The notion of "negative" situations (such as in (13), (17), (19) and (20)) deserves further comment. While one can imagine gathering together basic Davidsonian events or situations to form complex situations corresponding to conjunctive or quantified sentences, it is hard to imagine any Davidsonian approach to negative situations. For example, suppose that in (19) we attempt to represent the situation at issue as an episode *e* (at some locale) such that no rain falls *during e*. But this would not capture the meaning of the causal claim in (19). To say that certain conditions hold *during e* scarcely constrains *e* at all, merely requiring that it occupy a time and place *during* which the stated conditions obtain. But since many other conditions may obtain at that time and place as well, *e* could consist of much more than the drought implied by (19), and so cannot adequately serve as the cause anaphorically referred to in (19).

A possible response open to a Davidsonian is to say that negative phrasings like those in (17), (19), (20) imply the existence of corresponding "positive" eventualities, and these are the actual referents of the subsequent referring expressions.⁵ For example, (17) implies that the parachute *failed* to deploy, and the deictic pronoun *might* might refer to this failure; similarly the reference in (19) might be to the implied *dry spell* or *drought* or *lack of rain*; and the reference in (20) might be to the implied *absence* of Mary's children from home. However, such an (essentially pragmatic) account seems hard to defend, since we cannot freely refer deictically to implied entities. For example, the following

⁵In the context of Davidsonian event theory, the term 'eventuality' is often understood as an umbrella term covering events, processes, situations, etc., rather than in its ordinary sense of *contingency*, i.e., a *possible* event, outcome, or condition.

(a)-examples are infelicitous, in contrast with the (b)-examples (where referents are explicitly introduced):

- 21 a. He pricked the overinflated balloon with a pin.
??*This* was noisy.
- b. The overinflated balloon burst. *This* was noisy.
- 22 a. She ate two of the three cookies.
??She left *that cookie* for her brother.
- b. She ate all but one of the cookies.
She left *that cookie* for her brother.
- 23 a. He dreamed all night. ??*This sleep* was not refreshing.
- b. He slept fitfully all night. *This sleep* was not refreshing.
- 24 a. The sky was cloudless all day at the beach.
?*This* gave Mary a sunburn.
- b. The sun shone all day at the beach.
This gave Mary a sunburn.

(24a) is perhaps acceptable, but only if we are prepared to view the cloudlessness itself as the (indirect) cause of the sunburn. In other words, it is still the case that the deictic pronoun can refer only to the explicitly mentioned condition (cloudlessness).

A related point is that a Davidsonian approach makes it difficult to give expression to intuitions about the lexical meanings of "absence terms" like the ones employed for the implied positive eventualities in (17), (19) and (20), such as '*fail (to)*', '*dry*', '*empty*', '*lack*', or '*absent*'. For example, it seems natural to say that a situation *e* of *x* being absent from *y* is precisely one of *x not being present* at *y*. But we cannot express this as

- 25. $\forall e, x, y. \text{absent}(e, x, y) \leftrightarrow \neg \text{present}(e, x, y)$,

since the right-hand side merely says that *e* is not a situation of *x* being present at *y*; this surely doesn't require it to be a situation of *x* being absent from *y*. (For instance, it could be a situation of *x* liking *y* – which in a Davidsonian framework would presumably be distinguished from a situation of *x* being present at *y*, even if both situations occur simultaneously.) But if we omit the "if" direction of the conditional, then we have succeeded in expressing only half of the stated intuition.

Finally, it is sometimes claimed that apparent references to negative or quantified situations are actually references to facts or propositions. I would maintain instead that both simple and complex sentences can evoke situations as well as facts or propositions. It may sometimes be ambiguous whether a reference is to a situation or a fact, but there are predicates that distinguish the two sorts of entities, and can thus be used to test for the two sorts of reference. In particular, situations can persist/ endure/ last/ go on, whereas facts cannot (or if they do, they "persist" forever). On the other hand, facts can be asserted/ denied/ communicated, whereas situations cannot. The following examples illustrate the two kinds of reference, based on the distinguishing predicates, for negative and quantified sentences.

- 26 a. No rain fell.
- b. This situation endured/ lasted/ persisted/ went on for months.
- c. *This fact lasted for months.

- d. This fact is asserted in today's paper.
 - e. *This situation is asserted in today's paper.
- 27 a. Each superpower menaced the other with its nuclear arsenal
- b. This situation persisted for 4 decades.
 - c. *This fact persisted for 4 decades.
 - d. This fact is asserted in every history book.
 - e. *This situation is asserted in every history book.⁶

Facts are presumably related closely to true propositions, but I will not attempt develop a theory of facts or propositions here; instead I will focus exclusively on events or situations. However, it is worth mentioning in passing that EL permits both an "event view" and a "propositional view" of causation. They are connected as follows, based on a modal operator *because*:

$$\text{cause}(e,e') \wedge (\phi^{**}e) \wedge (\psi^{**}e') \leftrightarrow \text{because}(\phi^{**}e, \psi^{**}e').$$

An example of the corresponding wording in English might be '*John's singing caused Mary to wake up*', versus '*Mary woke up because John sang*'.

I now show how we can make formal sense of complex situations, where we allow both characterizations (using '**') and partial descriptions (using *) of such situations. This leads to a logic that naturally accommodates both Davidsonian events (at the level of atomic fluent predication) and situations of the sort posited in Situation Semantics. Despite the lack of propositions or properties in this logic (at least as first-class entities), the '*' operator turns out to have much in common with the ' \models ' relation of Situation Semantics.

4 FORMALIZING '*' AND '**'

Rather than employing a comprehensive representation for NL logical forms such as EL, I will restrict myself to a minimal extension of first-order logic (FOL) allowing for the '**' and '*' operators. The syntax will be conventional (without the square-bracketed infix predication of EL). I will call this extension FOL**.

4.1 SYNTAX OF FOL**

Terms in FOL** are individual constants and variables as well as functional terms formed with the 2-place function \sqcup (denoting the situation join). A subsort of the individual constants and variables are the *situational* constants and variables. A functional term contains only situational constants, variables and terms and is likewise a situational term.

Predicate constants have a specified arity and may have the initial argument position designated as situational. (Positions not designated as situational may be filled with any sort of term, general or situational.) Predicates with an initial situational argument are called *fluent* predicates. Note that situations in FOL** are thought of in general as extended in time, and having limited

⁶I am not claiming, of course, that any predicate applicable to situations is inapplicable to facts, or vice versa. For example, one can discuss, recount, ponder, remember, etc., both facts and situations. But the predicates in (26-27) do distinguish facts from situations rather reliably.

"information content"; in these respects they are more akin to the situations of Situation Semantics than those of McCarthy and Hayes, 1969. Still, the latter can be thought of as a special case of the former, and this motivates the "fluent" terminology. Non-fluent predicates are also termed "atemporal", though it should be noted that such predicates can have situational arguments. For instance, we would probably treat *cause*(e_1, e_2), expressing that situation e_1 causes (caused) situation e_2 , as atemporal. Also, the situational/temporal part-of relations \sqsubseteq , \sqsubset and \sqtriangleleft introduced below require situational arguments but are not considered fluents. Technically this means that they will not be used to form "situation abstracts" (see below).

Fluent predicates are further subdivided into *telic* and *atelic* predicates. Telic predicates correspond intuitively to events or occurrences, i.e., the types of situations that have a "built-in" culmination or point of completion; examples are yawning, greeting, dressing, building (something) and so on. Atelic predicates, by contrast, describe "open-ended" situations with no intrinsic point of completion, such as sleeping, walking, being awake, being at a location, being a nonsmoker, etc.⁷ Technically, the distinction we are after is that telic predication are "outward persistent", i.e., whatever happens in a temporal segment of a larger episode also happens in the larger episode; whereas atelic predication are "inward persistent" (or homogeneous), i.e., if such a predication characterizes an episode as a whole it also characterizes its temporal segments. As an example, if Mary greeted the audience in the opening of her talk, then she greeted the audience in her talk. This testifies to the telicity of "greet". On the other hand, if Mary was at the microphone for her talk, then she was at the microphone for the various parts of the talk. This testifies to the atelicity of "be (located) at".⁸

An important observation, which will affect some details of our formal semantics, is that negated fluent predication, whether telic or atelic, behave

⁷It would be more in keeping with traditional usage to classify only sleeping and walking as atelic among these examples, perhaps calling the rest "states". However, lumping these together under a single term serves my purposes here.

⁸Since the subject of aspectual verb classes is a subtle one, a couple of comments are in order. First, atelic predicates are often used in a way that implies less than uniform inward persistence. For instance, we might say that Mary was at the microphone in her talk even if she spoke a few words away from the microphone. We might even mean merely that she was at the microphone for some portion of her talk. I take the view that such uses involve tacit down-toning or bounding modifiers of the predicate, such as "mostly" or "for some time". The latter type-shifts the atelic predicate to a telic one (Hwang and Schubert, 1994). Secondly, it is well-known that in English the aspectual class of a predication depends not only on the predicate but also on the arguments, tense, adverbial adjuncts, and other factors. For instance, '*John yawned*' would normally be understood as specifying just one, terminating yawn, but '*John yawned for an hour*' or '*People yawn when they are tired*' appear to involve indeterminate numbers of yawns. Here again my view is that the corresponding LFs involve operators that interact with telicity. In particular, in '*yawn for an hour*' the duration adverbial forces a repetitive, essentially atelic reading of the verb, and this needs to be made logically explicit with an operator meaning 'repeatedly' (Hwang and Schubert, 1994). The second sentence is generic, and its proper LF can be assumed to be a tripartite quantificational structure quantifying over (actual and certain possible) episodes of people being tired (Carlson and Pelletier, 1995). These devices, and others needed for a more adequate treatment of aspectual class, are not available in FOL**.

like atelics. For example, if Mary did not take a sip of water in her talk, then she did not take a sip of water in any part of her talk. So this negated telic shows the same inward persistence that is the defining characteristic of atelics. As well, if Mary was not at the microphone in her talk, then she was not at the microphone in any part of her talk. Again, we have inward persistence. This will motivate making the *anti-extensions* of fluent predicates inward-persistent in the semantics. However, there remains one striking difference between negated telic and atelic predication that we must take into account: if we negate again, we recover a telic and thus outward persistent predication in the first case, and an atelic and thus inward persistent predication in the second. For example, if the claim that Mary didn't take a sip of water halfway through her talk is mistaken, then the claim that she didn't take a sip of water in her talk is also mistaken. I won't attempt to illustrate doubly negated atelics since they are all too susceptible to aspectual coercion; but we surely do not wish to treat double negation as *logically* distinct from the positive form.

Formulas in FOL^{**} are formed as in FOL by applying predicate constants to a suitable number of terms (of the right sort, if the initial argument position is designated as situational), by equating two terms, or by applying connectives \neg , \wedge , \vee , \rightarrow , \leftrightarrow , or quantifiers \forall , \exists to formulas. In addition, where τ, τ_1, τ_2 are situation terms,

$$\tau_1 \sqsubseteq \tau_2, \quad \tau_1 \prec \tau_2, \quad \tau_1 \trianglelefteq \tau_2,$$

are formulas. (Strict versions \sqsubset , \prec , and \triangleleft may also be used, with the obvious definitions in terms of the non-strict versions and inequality.) Intuitively, the three relations mean respectively that situation (or event) τ_1 is part of situation (or event) τ_2 , τ_1 is a concurrent part of τ_2 , and τ_1 is a temporal segment of τ_2 (i.e., consists of that part of τ_2 which occupies a particular subset of the times covered by τ_2).

The final class of formulas involves the notion of a *situation abstract*. A situation abstract is defined exactly like a formula, except that in place of atomic predication, we may use *proper atomic* situation abstracts. The latter are obtained from *fluent* predication by omitting the first argument. For instance, if *loves(E, John, Mary)* is a fluent predication, then

$$\text{loves(John, Mary)}$$

is a proper atomic situation abstract, and

$$\exists x. \text{person}(x) \wedge \text{loves}(x, \text{Mary}), \quad \text{and}$$

$$\exists x. \text{person}(x) \wedge \text{loves}(x, \text{Mary})$$

are non-atomic situation abstracts. Here '*person*' is also treated as a fluent. Alternatively it might be treated as a unary atemporal predicate; in that case '*person(x)*' in the second formula would not be a proper atomic situation abstract. Note that by definition all formulas are also situation abstracts (though of a "degenerate" sort, if they contain no proper atomic abstracts). A situation abstract is thought of as abstracting over situations satisfying the constraints it imposes with respect to the elided situational arguments. We now add the following types of formulas to our logic, where ϕ is a situation abstract and τ is a situation term:

$$(\phi^{**\tau}), \quad (\phi^*\tau).$$

As expected from the earlier discussion, these are read respectively as ' ϕ characterizes τ ' and ' ϕ (partially) describes τ (or holds or occurs in τ)'. Note that such formulas cannot be converted to proper situation abstracts, in the sense that the '*' and '**' operators will "bind" any fluent arguments abstracted in their scope. So in general, we define a proper situation abstract as a situation abstract that contains a proper atomic situation abstract not lying within the scope of a '*' or '**' operator.

4.2 MODELS FOR FOL**

The following formal definition will be followed by explanatory remarks. A *model* for FOL** is a 5-tuple $\mathcal{M} = (\mathcal{D}, (\text{Sit}, \sqsubseteq), (\mathcal{T}, <), \text{time}, \mathcal{I})$,⁹ where

- \mathcal{D} is a nonempty set (of individuals, including situations);
- $(\text{Sit}, \sqsubseteq)$ is a complete join semilattice with join \sqcup and set join \sqcup , where $\text{Sit} \subset \mathcal{D}$; $\sqcup\{s, s'\} = s \sqcup s'$ for all $s, s' \in \text{Sit}$; $\sqcup\emptyset = \varepsilon$; and $s \sqcup \varepsilon = s$ for all $s \in \text{Sit}$;
- $(\mathcal{T}, <)$ is a strict linear order (whose elements are regarded as time points);
- $\text{time}: \text{Sit} \rightarrow \text{Pow}(\mathcal{T})$ is a (total) function such that (a) if $s = \sqcup S$ then $\text{time}(s) = \bigcup\{\text{time}(s') \mid s' \in S\}$; (b) $\text{time}(s) = \emptyset$ iff $s = \varepsilon$; further conditions on the relation between *time* and situations are stated below.

We define the derived relations \preceq ("concurrent part of") and \trianglelefteq ("segment of") before proceeding. For $s, s' \in \text{Sit}$,

$$s \preceq s' \quad \text{iff } s \sqsubseteq s' \text{ and } \text{time}(s) = \text{time}(s'); \text{ and}$$

$$\begin{aligned} s \trianglelefteq s' \quad \text{iff and there exists } s'' \in \text{Sit} \text{ such that } & \text{time}(s'') \cap \text{time}(s) \\ & = \emptyset \text{ and } s \sqcup s'' = s'; \end{aligned}$$

A further condition on *time* in relation to situations that turns out to be crucial for establishing inward persistence of atelic descriptions is that temporal segmentation of situations is downward-inherited to concurrent parts, in the following sense. Let S be a set of concurrent parts of s (i.e., for each $s' \in S$, $s' \preceq s$) such that $\sqcup S = s$, and let $s_1 \trianglelefteq s$. Then there is a function $f: S \rightarrow \text{Sit}$ such that $\sqcup\{f(s') \mid s' \in S\} = s_1$ and for all $s' \in S$, $f(s') \trianglelefteq s'$ and $f(s') \preceq s_1$ (i.e., $\text{time}(f(s')) = \text{time}(s_1)$).

- \mathcal{I} (the interpretation) is a pair of functions $(\mathcal{I}^+, \mathcal{I}^-)$ on the individual and predicate constants, where if c is an individual constant, $\mathcal{I}^+(c) = \mathcal{I}^-(c) \in \mathcal{D}$ (and we can also write $\mathcal{I}(c)$ or $c^\mathcal{I}$ for $\mathcal{I}^+(c)$ or $\mathcal{I}^-(c)$), and if P is an n -place predicate constant, then $\mathcal{I}^+(P) \subseteq \mathcal{D}^n$ and $\mathcal{I}^-(P) \subseteq \mathcal{D}^n$. In particular, if P is a fluent predicate then $\mathcal{I}^+(P), \mathcal{I}^-(P) \subseteq (\text{Sit} - \{\varepsilon\}) \times \mathcal{D}^{n-1}$. In addition we build inward persistence into the anti-extensions of all predicates and the extensions of atelic ones: whenever

⁹Strictly, we should speak of models for a specific FOL** language based on a particular vocabulary, but this small inaccuracy should cause no confusion

$(s, d_1, \dots, d_{n-1}) \in \mathcal{I}^-(P)$ and $s' \sqsubseteq s$, we also have $(s', d_1, \dots, d_{n-1}) \in \mathcal{I}^-(P)$; and if P is atelic then whenever $(s, d_1, \dots, d_{n-1}) \in \mathcal{I}^+(P)$ and $s' \sqsubseteq s$, we also have $(s', d_1, \dots, d_{n-1}) \in \mathcal{I}^+(P)$. Finally, we stipulate that fluent extensions and anti-extensions do not “collide”; this means that if $(s, d_1, \dots, d_n) \in \mathcal{I}^+(P)$ and $(s', d_1, \dots, d_n) \in \mathcal{I}^-(P)$ then

- (i) $\text{time}(s) \not\subseteq \text{time}(s')$; and
- (ii) if P is atelic then $\text{time}(s) \cap \text{time}(s') = \emptyset$.

A caveat concerning the notation is that the symbols \sqsubseteq , \preceq , \sqsupseteq and \sqsubset are being overloaded (as is routinely done for equality) to do double duty as object-language and metalinguistic symbols.

Note the assumption that we can arbitrarily join situations into larger situations, whether or not they are concurrent. Note also that the concurrent part-of ordering \preceq and the temporal segment-of ordering \sqsubseteq specialize the general part-of ordering \sqsubseteq among situations. These orderings are relevant to the various kinds of persistence we are interested in (upward, outward, and inward). Also, an easily derived consequence from stipulation (a) about *time* is that

if $s \sqsubseteq s'$ then $\text{time}(s) \subseteq \text{time}(s')$;

i.e., the subset structure of the times at which situations occur (or hold) reflects the part-of structure of those situations. (However, note that we can have $s \sqsubset s'$, yet $\text{time}(s) = \text{time}(s')$ – in fact, this is the case whenever $s \prec s'$.) Though times play a crucial role in the semantics of FOL^{**}, for instance in assuring the persistence properties established in Section 5.5, we make no special provision for talking about times in the object language. (This might be done in later extensions of FOL^{**}.)

The most unusual feature of models as defined above is the use of predicate anti-extensions as well as extensions. This is a feature one would normally find only in partial logics such as that in (Muskeens, 1995), but FOL^{**} is not partial. Truth and falsity of ordinary formulas (not involving ‘*’ or ‘**’) will be based on predicate extensions alone. However, anti-extensions are crucial for specifying the semantics of negation in the context of ‘*’ and ‘**’. In particular, as will be seen, a negated atomic fluent abstract such as $\neg \text{at-home}(\text{Mary})$ characterizes a situation s just in case the anti-extension of the predicate contains the tuple consisting of s followed by the values of the specified arguments. In the example, the negative abstract characterizes s just in case $(s, m) \in \text{at-home}^{\mathcal{T}^-}$, where m is the denotation of ‘Mary’ in the model. Similarly $\neg \text{at-home}(\text{Mary})$ (partially) describes s just in case for some $s' \preceq s$, $(s', m) \in \text{at-home}^{\mathcal{T}^-}$.

Thus, to say that s is a situation of Mary not being at home is in general a much stronger statement (with a semantics determined by anti-extensions) than merely saying that s is not a situation of Mary being at home (whose semantics is determined by extensions). This is what enables us to overcome the problem noted for Davidsonian event semantics, that of not being able to represent negative eventualities on a par with positive ones. For assigning truth values to formulas with outer operator ‘*’ or ‘**’, the semantics of extensions and anti-extensions gets us off the ground, and we can then generalize to arbitrarily complex event characterizations or descriptions (though as will be

seen, we need the mediation of a “situational support” function that provides all sets of situations that can support the truth of a situation characterization or description). The resemblance of FOL^{**} models to those of partial truth theories is explained by the fact that characterization or description of a situation (especially the latter) is essentially a relation of partial truth in a situation; i.e., it is entirely possible that neither a given situation abstract, nor its negation, characterizes or describes a given situation. However, the “collision avoidance” clause in the definition of extensions and anti-extensions ensures that a situation abstract and its negation cannot both characterize (or describe) a situation. This requirement is not automatically satisfied just by making extensions and anti-extensions disjoint, since the persistence of information in the situational part-of orderings can propagate characterization and partial description relations upward, outward, and inward in those orderings.

Finally note that inward and outward persistence are not treated in the semantics as exact duals. Through the definition of an interpretation, we have ensured that atelic as well as negative predication already exhibit inward persistence (homogeneity) in their extensions. By contrast, outward persistence does not show up directly in extensions, but rather shows up only in the context of the ‘*’ and ‘**’ operators, i.e., in the context of situation descriptions and characterizations. This is why we needed to employ the notion of “collision”, which anticipates the persistence properties of descriptions relative to situations.

4.3 TRUTH IN FOL^{**}

Term denotations and truth values of formulas are defined relative to a model $\mathcal{M} = (\mathcal{D}, (\text{Sit}, \sqsubseteq), (\mathcal{T}, <), \text{time}, \mathcal{I})$ and a variable assignment \mathcal{U} . Truth for sentences (closed formulas) is then defined by quantifying universally (or existentially) over assignments. Situation variables receive values in Sit under an assignment, while other individual variables receive values in \mathcal{D} . As usual, the denotation $c^{\mathcal{M}, \mathcal{U}}$ of a constant c is $c^{\mathcal{T}}$, the denotation $x^{\mathcal{M}, \mathcal{U}}$ of a variable x is $x^{\mathcal{U}}$, and the denotation $(\sqcup(r_1, r_2))^{\mathcal{M}, \mathcal{U}}$ of a term $\sqcup(r_1, r_2)$ (where r_1, r_2 are terms) is $r_1^{\mathcal{M}, \mathcal{U}} \sqcup r_2^{\mathcal{M}, \mathcal{U}}$.

We first consider truth conditions for formulas other than those with top-level operator ‘*’ or ‘**’. In the following P is an n -place predicate, r_1, \dots, r_n are terms, x is a variable, and ϕ and ψ are formulas.

T-pred: $\models_{\mathcal{M}, \mathcal{U}} P(r_1, \dots, r_n)$ iff $(r_1^{\mathcal{M}, \mathcal{U}}, \dots, r_n^{\mathcal{M}, \mathcal{U}}) \in P^{\mathcal{I}^+}$;

T=: $\models_{\mathcal{M}, \mathcal{U}} (r_1 = r_2)$ iff $r_1^{\mathcal{M}, \mathcal{U}} = r_2^{\mathcal{M}, \mathcal{U}}$;

T \sqsubseteq : $\models_{\mathcal{M}, \mathcal{U}} (r_1 \sqsubseteq r_2)$ iff $r_1^{\mathcal{M}, \mathcal{U}} \sqsubseteq r_2^{\mathcal{M}, \mathcal{U}}$; and similarly for **T \leq** , **T \trianglelefteq** ;

T \neg : $\models_{\mathcal{M}, \mathcal{U}} \neg\phi$ iff not $\models_{\mathcal{M}, \mathcal{U}} \phi$;

T \wedge : $\models_{\mathcal{M}, \mathcal{U}} (\phi \wedge \psi)$ iff $\models_{\mathcal{M}, \mathcal{U}} \phi$ and $\models_{\mathcal{M}, \mathcal{U}} \psi$;

Similarly for **T \vee** , **T \rightarrow** , **T \leftrightarrow** ;

T \forall : $\models_{\mathcal{M}, \mathcal{U}} (\forall x)\phi$ iff for all $d \in \mathcal{D}$, $\models_{\mathcal{M}, \mathcal{U}_{d/x}} \phi$; and similarly for **T \exists** .

The truth conditions for '*' and '**' are specified in terms of a function $Support_{\mathcal{M}, \mathcal{U}}^+(\phi, s)$ and its dual $Support_{\mathcal{M}, \mathcal{U}}^-(\phi, s)$ defined inductively below. Intuitively, $Support_{\mathcal{M}, \mathcal{U}}^+(\phi, s)$ supplies all possible sets of situations that "support" a situation abstract ϕ in situation s (relative to model \mathcal{M} and assignment \mathcal{U}). Roughly speaking, a set of situations supports ϕ in situation s if they are all part of s , and together "verify" a set of basic facts sufficient to support the truth of ϕ in s . The part-of relation between an element of such a support set and s depends on the telicity of the basic fact the element supports. The existence of a support set for ϕ in s verifies the (partial) description relation between ϕ and s . When the join of elements of a support set equals s , this (in addition) verifies the characterization relation between ϕ and s .

For example, consider $\phi = \exists x. person(x) \wedge greet(x, Mary)$, where both conjuncts are proper situation abstracts (i.e., both 'person' and 'greet' have an implicit situation argument). A support set for this abstract might be $\{s_1, s_2\}$, where for some individual d , (s_1, d) is in the extension of 'person' and (s_2, d, m) is in the extension of 'greet' (where m is the denotation of 'Mary'); and in addition, $s_1 \preceq s$ and $s_2 \sqsubseteq s$. So in that case, if E denotes s , then $(\phi * E)$ is true in the model under consideration. Furthermore, if $s_1 \sqcup s_2 = s$ (i.e., $\{s_1, s_2\}$ is precisely sufficient to support ϕ in s), then $(\phi**E)$ is true in the model. Note that we require a ' \preceq ' (concurrent part-of) relation between s_1 and s , since 'person' is atelic, so that its truth in s requires its truth *everywhere* in s ; whereas we merely require a ' \sqsubseteq ' (part-of) relation between s_2 and s , since 'greet' is telic so that its truth anywhere in s assures its truth in s as a whole.

In the following definition of $Support_{\mathcal{M}, \mathcal{U}}^+(\dots)$ and $Support_{\mathcal{M}, \mathcal{U}}^-(\dots)$, ϕ and ψ are situation abstracts, $s \in Sit$, x is a variable, r_1, \dots, r_n are terms, and P is an $(n+1)$ -place fluent predicate (with an initial situation argument followed by n additional arguments). S is a variable over sets of situations that is understood to be universally quantified.

If ϕ is a formula (i.e., not a proper situation abstract), then

$$\begin{aligned} Support_{\mathcal{M}, \mathcal{U}}^+(\phi, s) &= \{\emptyset\} \quad \text{if } \models_{\mathcal{M}, \mathcal{U}} \phi, \\ &= \emptyset \quad \text{otherwise}; \end{aligned}$$

$S \in Support_{\mathcal{M}, \mathcal{U}}^+(P(r_1, \dots, r_n), s)$ iff $S = \{s'\}$ for some $s' \in Sit$ such that s' rel s and $(s', r_1^{\mathcal{M}, \mathcal{U}}, \dots, r_n^{\mathcal{M}, \mathcal{U}}) \in P^{\mathcal{T}^+}$, where rel is \sqsubseteq if P is telic and \preceq if P is atelic;

$S \in Support_{\mathcal{M}, \mathcal{U}}^+(\neg\phi, s)$ iff $S \in Support_{\mathcal{M}, \mathcal{U}}^-(\phi, s)$;

$S \in Support_{\mathcal{M}, \mathcal{U}}^+((\phi \wedge \psi), s)$ iff $S = S' \cup S''$ for some $S' \in Support_{\mathcal{M}, \mathcal{U}}^+(\phi, s)$ and some $S'' \in Support_{\mathcal{M}, \mathcal{U}}^+(\psi, s)$;

$S \in Support_{\mathcal{M}, \mathcal{U}}^+((\phi \vee \psi), s)$ iff $S \in Support_{\mathcal{M}, \mathcal{U}}^+(\phi, s)$ or $S \in Support_{\mathcal{M}, \mathcal{U}}^+(\psi, s)$;

(corresponding conditions for \rightarrow and \leftrightarrow are easily filled in);

$S \in Support_{\mathcal{M}, \mathcal{U}}^+((\forall x)\phi, s)$ iff there exists a function $f : \mathcal{D} \rightarrow Pow(\mathcal{D})$ such that $S = \bigcup range(f)$ and for each $d \in \mathcal{D}$, $f(d) \in Support_{\mathcal{M}, \mathcal{U}_{d/x}}^+(\phi, s)$;

$S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^+(\exists x)\phi, s$ iff for some $d \in \mathcal{D}$, $S \in \text{Support}_{\mathcal{M}, \mathcal{U}_{d/x}}^+(\phi, s)$.

The definition of $\text{Support}_{\mathcal{M}, \mathcal{U}}^-(\dots)$ is very nearly the dual of that of $\text{Support}_{\mathcal{M}, \mathcal{U}}^+(\dots)$, obtained from the clauses for $\text{Support}_{\mathcal{M}, \mathcal{U}}^+(\dots)$ by interchanging \models and $\not\models$, $+$ and $-$, \wedge and \vee , and \exists and \forall . The only amendment is that *rel* in the second clause (for atomic predication) is simply \leq , since anti-extensions behave like atelic extensions:

$S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^-(P(r_1, \dots, r_n), s)$ iff $S = \{s'\}$ for some $s' \in \text{Sit}$ such that $s' \leq s$ and $(s', r_1^{\mathcal{M}, \mathcal{U}}, \dots, r_n^{\mathcal{M}, \mathcal{U}}) \in P^T^-$,

We can now complete the truth definition (e is a situation term):

T*: $\models_{\mathcal{M}, \mathcal{U}} (\phi * e)$ iff $\text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi, e^{\mathcal{M}, \mathcal{U}}) \neq \emptyset$;

T**: $\models_{\mathcal{M}, \mathcal{U}} (\phi ** e)$ iff there is an $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi, e^{\mathcal{M}, \mathcal{U}})$ such that $e^{\mathcal{M}, \mathcal{U}} = \bigcup S$.

Note that the definition of $\text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi, s)$ in the case where ϕ is a formula (and thus not a proper situation abstract) is designed not to “interfere” with the correct selection of support sets for proper situation abstracts. For instance, in the case of a conjunction $(\phi \wedge \psi)$ of a formula ϕ and a proper situation abstract ψ , the support sets will simply be the ones for ψ , as long as ϕ is true. This is because the only support set for ϕ in that case is the empty set (whose union with any support set for ψ is of course that same set). If ϕ is false, then it has no support sets ($\text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi, s) = \emptyset$), so in that case the conjunction has no support sets either, and $((\phi \wedge \psi) * e)$ will be false regardless of e . This is as it should be, given the truth-like character of the $*$ -relation. An easily verified consequence of these stipulations is that a true sentence is considered to (partially) describe any situation, and characterize none, save ε . A false sentence neither describes nor characterizes any situation.

5 PROPERTIES OF FOL**

We now look at some of the consequences of the preceding semantics for FOL**. First we note a few of the laws that reflect the model-theoretic assumptions made about the situation orderings. Then we state some of the most important relationships between ‘*’ and ‘**’, laws governing complex formulas within the scopes of ‘*’ and ‘**’, persistence laws for ‘*’ (when we keep the situation abstract argument fixed while replacing the situation argument by one denoting a larger or smaller situation), and finally a set of sound inference rules. The logic seems to deal in a satisfactory way with simple and complex situations. At the level of atomic fluent sentences, the sentence-event connection provided by ‘**’ is precisely Davidson’s. But ‘**’ also provides a coupling between negated atomic sentences and situations, and indeed between arbitrarily complex sentences and situations; and this coupling is tight enough to deal with causal descriptions. Furthermore, the weakened version of the coupling, ‘*’, behaves logically very much like the support relation, ‘ \models ’, of Situation Semantics.

5.1 LAWS CONCERNING \sqsubseteq , \preceq , AND \trianglelefteq

Validity of a formula ϕ , $\models \phi$, is defined as usual as truth in all models. The following are some obviously valid formulas, given the truth conditions for \sqsubseteq , \preceq , and \trianglelefteq and the assumptions about the orderings they represent. Multiple quantifiers of the same type are usually collapsed into a single quantifier with multiple variables.

- O1. $\models \forall e.(e \sqsubseteq e)$
- O2. $\models \forall e, e', e''.(e \sqsubseteq e') \wedge (e' \sqsubseteq e'') \rightarrow (e \sqsubseteq e'')$
- O3. $\models \forall e, e'.(e \sqsubseteq e') \wedge (e' \sqsubseteq e) \rightarrow (e = e')$
- O4. $\models \forall e, e'.(e \sqsubseteq \sqcup(e, e'))$
- O5. $\models \forall e, e', e''.(e' \sqsubseteq e) \wedge (e'' \sqsubseteq e) \rightarrow (\sqcup(e', e'') \sqsubseteq e)$
- O6. $\models \forall e, e'.(e \preceq e') \rightarrow (e \sqsubseteq e')$
- O7. $\models \forall e, e'.(e \trianglelefteq e') \rightarrow (e \sqsubseteq e')$

Idempotency and commutativity of \sqcup follow from O4 and O5, and associativity from O2–O5. One law that is not completely obvious is

- O8. $\models \forall e, e'.(e \trianglelefteq e') \wedge (e \preceq e') \rightarrow (e = e')$

This is readily proved from the definitions of (metalinguistic) \trianglelefteq and \preceq , along with the fact that if $s \neq \varepsilon$, then $\text{time}(s) \neq \emptyset$ (in view of the assumptions about ε and time).

An intuitively plausible formula whose validity we might want to ensure in a future strengthening of the model theory is

$$\forall e, e'.(e \sqsubseteq e') \rightarrow \exists e''.(e \preceq e'') \wedge (e'' \trianglelefteq e').$$

A final point worth noting is that since we have assumed a minimal element ε in our complete semilattice of situations, we actually have a complete lattice, and so could introduce a meet operator.¹⁰

5.2 RELATIONSHIPS BETWEEN '*' AND '**'

Three basic laws connecting '*' and '**' are the following, where ϕ is a situation abstract.

- $$\begin{aligned} (** \rightarrow *) & \models \forall e.(\phi**e) \rightarrow (\phi*e) \\ (* \rightarrow **) & \models \forall e.(\phi*e) \rightarrow \exists e'.(e' \sqsubseteq e) \wedge (\phi**e') \\ (* \rightarrow **)_{atel} & \models \forall e.(\phi_{atel}*e) \leftrightarrow \exists e'.(e' \preceq e) \wedge (\phi_{atel}**e') \end{aligned}$$

The proof of the first law is immediate from the truth conditions for '*' and '**'. The second law is more subtle, stating that a (partial) description of a

¹⁰This well-known fact rests on the ability to take the set join of elements that are common parts of given elements, to obtain their meet – which will exist since at least ε is part of it.

situation always characterizes some part of it. For the proof we make use of the following

Lemma 1. (a) If $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi, s)$ then $\bigsqcup S \sqsubseteq s$ and for every $s' \in \text{Sit}$ such that $\bigsqcup S \sqsubseteq s' \sqsubseteq s$, $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi, s')$. (b) As in (a), with ' $^+$ ' replaced by ' $^-$ '.

Lemma 1 can be proved straightforwardly (if somewhat tediously) by simultaneous induction on the complexity of ϕ in (a) and (b), with reference to the definition of support sets. In the inductive argument for negated abstracts, the proof of (a) depends on the induction hypothesis for (b) and vice versa. (Several other lemmas to follow use similar proofs.) The proof of $(* \rightarrow **)$ then essentially starts with the supposition that $\models_{\mathcal{M}, \mathcal{U}} (\phi * e)$ where $e^{\mathcal{U}} = s$ for some arbitrary $s \in \text{Sit}$. Then by the lemma there is a positive support set S for (ϕ, s) whose join is part of s , and S is also a positive support set for $(\phi, \bigsqcup S)$. But then $\bigsqcup S$ supplies the value of e' that verifies the consequent in $(* \rightarrow **)$ (relative to the \mathcal{M}, \mathcal{U} under consideration).

In the third law the '*atcl*' subscript indicates an *atelic situation abstract*. For such an abstract, every proper atomic situation abstract occurring in it outside the scope of all ' $*$ ' and ' $**$ ' operators either has an atelic predicate or has a telic predicate and lies in a negative environment. A negative environment is one embedded in an odd number of negations, where antecedents of conditional formulas or abstracts count as carrying a negation (and \leftrightarrow is rewritten in terms of \rightarrow). For later reference, we similarly define a *telic situation abstract* as one in which every proper atomic situation abstract occurring outside the scope of all ' $*$ ' and ' $**$ ' operators has a telic predicate and lies in a positive environment. Note that not all abstracts are telic or atelic. For instance, *tired(John) → yawn(John)* is neither telic nor atelic, since it involves an atelic predicate in a negative environment and a telic predicate in a positive environment. The proof of $(* \rightarrow **)_{\text{atcl}}$ depends on the following lemma.

Lemma 2. (a) If ϕ is atelic and $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi, s)$, then for all $s' \in S$, $s' \preceq s$. (b) If the negation of ϕ is atelic (i.e., every proper atomic situation abstract occurring in ϕ outside the scope of all ' $*$ ' and ' $**$ ' operators has an atelic predicate or it has a telic predicate in a positive environment), and $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^-(\phi, s)$, then for all $s' \in S$, $s' \preceq s$.

The proof is by simultaneous induction for (a) and (b), paying attention to downward inheritance of atelicity to parts of formulas. As basis we consider positive and negated atelic atomic situation abstracts. This involves looking at the positive support definition for positive and negated atelics and negated telics, and the negative support definition for positive and negated atelics and for positive telics. In all cases the \preceq relation applies. For conjunction, disjunction, and quantification telicity is inherited by the constituent abstracts. For negation we again "switch" between the positive and negative support cases (a) and (b).

The proof for $(\rightarrow)**_{atel}$ is then quite obvious in both directions, since all relevant support sets consist of situations concurrent with the denotation of e , by Lemma 2.

5.3 LAWS CONCERNING COMPLEX FORMULAS WITHIN THE SCOPE OF '**'

The behavior of complex formulas within the scope of '**' reflects the fact that truth for this characterization operator is based on joins of situations in a support set. I believe that the laws that follow are in keeping with intuitions about what it means to characterize a situation or event. For example, If we characterize a situation with a binary conjunction, then that situation is presumably made up of two parts (which might be concurrent aspects or temporal segments), each characterized by one of the two sentences. That is exactly what we find in $(\wedge)**$ below, among other reasonable properties. The first group of laws concerns the movement of various operators out of or into the scope of '**'. P is an $(n+1)$ -place fluent predicate.

$$(\text{PRED})^{**}: \models \forall x_1, \dots, x_n. \forall e. (P(x_1, \dots, x_n)**e) \leftrightarrow P(e, x_1, \dots, x_n)$$

$$(\neg)**: \models \forall e. ((\neg\phi)**e) \rightarrow (\neg\phi**e)$$

$$(\wedge)**: \models \forall e. ((\phi \wedge \psi)**e) \rightarrow \exists e', e''. (\sqcup(e', e'') = e) \wedge (\phi**e') \wedge (\psi**e'')$$

$$(\wedge)_{tel}^{**}: \models \forall e. ((\phi_{tel} \wedge \psi_{tel})**e) \leftrightarrow \exists e', e''. (\sqcup(e', e'') = e) \wedge (\phi_{tel}**e') \wedge (\psi_{tel}**e'')$$

$$(\wedge)_{atel}^{**}: \models \forall e. ((\phi_{atel} \wedge \psi_{atel})**e) \leftrightarrow \exists e', e''. (\sqcup(e', e'') = e) \wedge (e' \preceq e) \wedge (e'' \preceq e) \wedge (\phi_{atel}**e') \wedge (\psi_{atel}**e'')$$

$$(\vee)**: \models \forall e. ((\phi \vee \psi)**e) \leftrightarrow (\phi**e) \vee (\psi**e)$$

$$(\forall)**: \models \forall e. ((\forall x \phi)**e) \rightarrow \forall x. \exists e', e''. (\sqcup(e', e'') = e) \wedge (\phi**e') \wedge ((\forall y. (y \neq x) \rightarrow \phi_{y/x})*e'')$$

$$(\exists)**: \models \forall e. ((\exists x \phi)**e) \leftrightarrow \exists x. (\phi**e)$$

$$(=)**: \models \forall e, x, y. (\phi**e) \wedge x = y \rightarrow (\phi_{y/x}**e)$$

(PRED)** shows that characterization at the atomic level is equivalent to a Davidsonian predication. Its validity is obvious from the definition of truth and support for '**'. $(\neg)**$ is an easy consequence of the following lemma.

Lemma 3. For every situation abstract ϕ and situation s , either $\text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi, s) = \emptyset$ or $\text{Support}_{\mathcal{M}, \mathcal{U}}^-(\phi, s) = \emptyset$, regardless of \mathcal{M}, \mathcal{U} .

An inductive basis for proving the lemma is given by the non-collision of predicate extensions and anti-extensions, and the fact that for s' to be in a negative support set relative to s , it must be concurrent with s , and so no tuple in the extension of the predicate can have as its first element a situation that is part of s . The induction step is straightforward.

$(\wedge)^{**}$ can be proved from the definition of support for conjunctions (and of course truth of ** -formulas). A counterexample to the converse direction is obtained by choosing ϕ to be P , an abstract of a unary fluent predicate, and ψ to be $\neg P$, with the denotations of e' , e'' temporally non-overlapping and with a join equal to the value of e under some variable assignment. Then with a suitable interpretation of P , the consequent in $(\wedge)^{**}$ is verified but the antecedent is false, because its truth requires the supporting situation for $\neg P$ to be concurrent with the denotation of e . However, $(\wedge)_{tel}^{**}$ asserts a biconditional for the case where both ϕ and ψ are telic, and $(\wedge)_{ateli}^{**}$ is an analogous law for atelic ϕ and ψ , though it requires the extra condition on the right that e and e' be concurrent with e . The two variants of $(\wedge)^{**}$ require lemmas, viz., Lemma 2 and the following related one.

Lemma 4. (a) If ϕ is telic, $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi, s)$, and $s \sqsubseteq s'$ then $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi, s')$. (b) If the negation of ϕ is telic (i.e., every proper atomic situation abstract occurring in ϕ outside the scope of all ' $*$ ' and ' ** ' operators has a telic predicate and lies in a negative environment), and $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^-(\phi, s)$, and $s \sqsubseteq s'$ then $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^-(\phi, s')$.

The proof (as in Lemma 2) is by simultaneous induction for (a) and (b), again with attention to downward inheritance of telicity to parts of formulas. As basis we consider positive and negated telic atomic situation abstracts. For conjunction, disjunction, and quantification telicity is inherited by the constituent abstracts. For negation we then "switch" between cases (a) and (b).

The proofs of $(\wedge)_{tel}^{**}$ and $(\wedge)_{ateli}^{**}$ are then simple. The former uses Lemma 4 for the backward implication, and the latter Lemma 2. $(\vee)^{**}$ is obvious from the definitions. $(\forall)^{**}$ states that a universal characterization of a situation implies that for each element of the domain there is a subsituation supporting the characterization of that element, and another subsituation supporting the characterization of the remaining elements. Since we don't have sets and set joins in the object language, we can't readily break down the universal characterization more uniformly. The proof idea is much as for conjunction. In assuming the truth of the antecedent of $(\forall)^{**}$ for a particular value of e , we are assuming the existence of a function f as specified in the definition of $\text{Support}_{\mathcal{M}, \mathcal{U}}^+((\forall x)\phi, s)$. For any particular value d of x in the consequent, this function supplies the value of both e' (as $\bigsqcup f(d)$) and e'' (as another set join, over the union of sets $f(d')$ for all $d' \neq d$); a variant f' of f which is the same as f except that $f'(d) = \emptyset$ serves to verify the truth of the last conjunct in the consequent of $(\forall)^{**}$. We could again set up special cases for telic and atelic ϕ , but we omit this. $(\exists)^{**}$, a law for existential characterization, indicates the "transparency" of ' ** ' to existential quantification. The final law $(=)^{**}$ is particularly noteworthy, showing the transparency of ' ** ' with respect to substitution of coreferential terms. This can be shown straightforwardly by induction on the complexity of ϕ , with use of the fact that term denotations are invariant under substitution of coreferential terms (whose proof is as in FOL, since complex FOL ** terms, though restricted here to being situational, are

interpreted as in FOL). Essentially the induction establishes the rather obvious fact that the definition of (positive and negative) support sets is insensitive to substitution of coreferential terms in any of the situation abstracts mentioned in the definition.

A second group of laws concerns transformations *within* the scope of '**'. Negation distribution within the scope of '**' turns out to be valid, while the behavior of conjunction and disjunction is non-truth-functional in a way that may at first seem surprising, but which I believe reflects desirable properties of the logic.

- $(\neg\neg)**: \models \forall e.((\neg\neg\phi)**e) \leftrightarrow (\phi**e)$
- $(\neg\wedge)**: \models \forall e.((\neg(\phi \wedge \psi))**e) \leftrightarrow ((\neg\phi \vee \neg\psi)**e)$
- $(\neg\vee)**: \models \forall e.((\neg(\phi \vee \psi))**e) \leftrightarrow ((\neg\phi \wedge \neg\psi)**e)$
- $(\neg\forall)**: \models \forall e.((\neg(\forall x \phi))**e) \leftrightarrow ((\exists x \neg\phi)**e)$
- $(\neg\exists)**: \models \forall e.((\neg(\exists x \phi))**e) \leftrightarrow ((\forall x \neg\phi)**e)$
- $(\phi \wedge \phi)**: \models \forall e.(\phi**e) \rightarrow ((\phi \wedge \phi)**e) \text{ but not the converse}$
- $(\phi \vee \phi)**: \models \forall e.(\phi**e) \leftrightarrow ((\phi \vee \phi)**e)$
- $(\vee\wedge)**: \models \forall e.((\phi \vee (\chi \wedge \psi))**e) \rightarrow (((\phi \vee \chi) \wedge (\phi \vee \psi))**e)$
but not the converse
- $(\wedge\vee)**: \models \forall e.((\phi \wedge (\chi \vee \psi))**e) \rightarrow (((\phi \wedge \chi) \vee (\phi \wedge \psi))**e)$

$(\neg\neg)**$ is obvious from the identity $\text{Support}_{\mathcal{M}, \mathcal{U}}^+(\neg\phi, s) = \text{Support}_{\mathcal{M}, \mathcal{U}}^-(\phi, s)$ and its dual. $(\neg\wedge)**$ is also easily proved from these identities and the membership conditions for $\text{Support}_{\mathcal{M}, \mathcal{U}}^-(\phi \wedge \psi, s)$ and $\text{Support}_{\mathcal{M}, \mathcal{U}}^+(\neg\phi \vee \neg\psi, s)$; similarly for $(\neg\vee)**$, $(\neg\forall)**$, and $(\neg\exists)**$. The failure of the converse of $(\phi \wedge \phi)**$ may seem surprising – one might expect the embedded conjunction $\phi \wedge \phi$ to be equivalent to ϕ . However, it needs to be kept in mind that ϕ is (in general) a situation abstract, not a formula, and as such makes tacit reference to a situation – and distinct occurrences of ϕ may refer to distinct situations. Consider, for example, a conjunction such as *sneeze(John) \wedge sneeze(John)*. Though this could be a repeated description of the same sneezing event, it could also perfectly well be a description of two distinct such events; the logic as such should not commit us to either one of these possibilities, and so it would be undesirable for the converse of $(\phi \wedge \phi)**$ to hold. The point is perhaps even clearer from an example such as '*Beethoven composed a violin concerto and (he composed) a piano concerto*'; this entails '*Beethoven composed a concerto and (he composed) a concerto*'. We would not want the latter conjunction to be, of necessity, the description of a single composing event.¹¹ Note that by contrast,

¹¹ It turns out that with an additional model-theoretic assumption, the converse of $(\phi \wedge \phi)**$, with ϕ restricted to atelic formulas that can be written as conjunctions of atoms or negated atoms, with no existential quantification, is valid. The additional assumption is the following

ϕ and $\phi \vee \psi$ are indistinguishable in the scope of '**', as seen in $(\phi \vee \psi)**$ (again a simple consequence of the definition of *Support*[†]). A closely related contrast is seen in $(V\wedge)**$ and $(\wedge V)**$: distribution of 'V' over ' \wedge ' in the scope of '**' is not "reversible", unlike distribution of ' \wedge ' over 'V'.

Thus **-contexts are quite subtle, in some respects (e.g., $(V)**$, $(\exists)**$ and $(=)**$) behaving "transparently", and in others (e.g., the failure of the converses of $(\phi \wedge \psi)**$ and $(V\wedge)**$) showing an even stronger sensitivity to logical form than classical modal operators. To the extent that '**' can be viewed as a reconstruction of Reichenbach's '[]*' operator (setting aside Reichenbach's conflation of facts and events), the semantics provided here refutes Davidson's main objection to Reichenbach's proposal (Davidson, 1967b). Essentially, Davidson argued that []*-contexts must allow substitution of coreferential singular terms, hence should also allow substitution of logically equivalent sentences, and this leads to indistinguishability of all events. But as seen from the properties of '**', a Reichenbach-like operator may perfectly well allow substitution of coreferential singular terms without also allowing substitution of logically equivalent sentences (more precisely, situation abstracts). At the same time, such a theory is compatible with Davidson's conception of events at the level of atomic predication, as seen from $(PRED)**$.

5.4 COMPLEX FORMULAS WITHIN THE SCOPE OF '*': TRUTH-LIKENESS AND TRANSPARENCY

We now consider complex formulas within the scope of '*'. These show the much simpler, more transparent behavior of '*' compared to '**', as one would expect from a partial truth-like operator.

- $(\neg)* \models \forall e.((\neg\phi)*e) \rightarrow \neg(\phi*e)$
- $(\wedge)* \models \forall e.((\phi \wedge \psi)*e) \leftrightarrow (\phi*e) \wedge (\psi*e)$
- $(V)* \models \forall e.((\phi \vee \psi)*e) \leftrightarrow (\phi*e) \vee (\psi*e)$
- $(\forall)* \models \forall e.((\forall x \phi)*e) \leftrightarrow \forall x.(\phi*e)$
- $(\exists)* \models \forall e.((\exists x \phi)*e) \leftrightarrow \exists x.(\phi*e)$

In addition we could give analogous formulas for \rightarrow and \leftrightarrow . Also, many standard manipulations are possible *within* the scope of '*', such as the following.

$$(\neg\wedge)* \models \forall e.((\neg(\phi \wedge \psi))*e) \leftrightarrow ((\neg\phi \vee \neg\psi)*e)$$

rather plausible situation individuation assumption: If two n -tuples in the extension of a fluent predicate differ only in their (initial) situational element, then the times of those situations are distinct. This applies to both positive and negative extensions. For example, two events of John sneezing, or two episodes of his not being at home, cannot be concurrent yet distinct.

$$(\neg\vee)^* \models \forall e.((\neg(\phi \vee \psi))*e) \leftrightarrow ((\neg\phi \wedge \neg\psi)*e)$$

In fact, bidirectional versions of all the transformations in the scope of ‘**’, listed earlier, hold for ‘*’, and these (along with others) can be captured by a general “embedded inference rule” (below). $(\neg)^*$, like $(\neg)**$, follows from Lemma 3. The laws for \wedge , \vee , \forall and \exists , follow directly from the corresponding support set definitions. The embedded rules (and others like them) are proved by showing that the existence of a support set for one side of the equivalence ensures that one exists for the other side as well. Note that once we have distributed negation in this way, we can apply $(\vee)^*$ or $(\wedge)^*$; this proves useful below.

5.5 PERSISTENCE OF INFORMATION RELATIVE TO \sqsubseteq , \preceq , \trianglelefteq

Among the most important properties of a situation logic are its persistence properties – as we go from “smaller” to “larger” situations, information is preserved (and expands). The point is that events or situations are limited parts or aspects of the world, and we manipulate these aspects cognitively with great ease, expanding or narrowing our purview at will. In FOL**, we capture not only the “upward” growth of information, but also its “outward” growth to temporally larger episodes, and its “inward preservation” for certain kinds of information, as illustrated earlier. For example, we noted that a greeting uttered at the beginning of a talk is also a greeting uttered in the talk, which may be temporally much larger. In fact, the temporal persistence of information seems to play an important role in NL, finding expression in time adverbials, for example. And it is also important to be aware of inward persistence (homogeneity) of many kinds of information, for instance recognizing that if the library is closed tomorrow, then it will be closed no matter what time I show up tomorrow. The following are the most important persistence properties in FOL**.

$$(\mathbf{UP}) \models \forall e, e'.(e \preceq e') \wedge (\phi * e) \rightarrow (\phi * e')$$

$$(\mathbf{OUT}) \models \forall e, e'.(e \sqsubseteq e') \wedge (\phi * e) \rightarrow \neg((\neg\phi) * e') \quad \text{if } \phi \text{ is telic} \\ \text{or strictly atelic (i.e., both } \phi \text{ and } \neg\phi \text{ are atelic)}$$

$$(\mathbf{OUT}_{tel}) \models \forall e, e'.(e \sqsubseteq e') \wedge (\phi_{tel} * e) \rightarrow (\phi_{tel} * e')$$

$$(\mathbf{IN}_{atel}) \models \forall e, e'.(e \trianglelefteq e') \wedge (\phi_{atel} ** e') \rightarrow (\phi_{atel} ** e)$$

(UP) is the most general persistence property, stating that all types of information persist through the concurrent part-of ordering. One possible proof is by induction and by use of the previous laws for complex formulas in *-environments, with a separate induction for negated abstracts, aided by the “embedded” inference rules mentioned above (and again below). A much more direct proof is enabled by the following lemma.

Lemma 5.

- (a) If $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi, s)$, and $s \preceq s'$, then $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi, s')$.

(b) As in (a), with ‘+’ replaced by ‘−’.

The proof is again a straightforward simultaneous induction for (a) and (b), using the definition of support sets and the basic properties of the situation orderings.

The (**OUT**) law can also be proved inductively, with a basis that depends on the collision avoidance condition on predicate extensions and anti-extensions in the definition of models. As in the case of (**UP**), the proof can be carried through using the previously established laws for *-environments, or using appropriate semantic lemmas; but we omit further details. The need for the qualification in the statement of (**OUT**) can be appreciated by considering an example of a negated telic in the context of ‘*’, such as $((\neg lightning)*e)$ (i.e., there’s no lightning in episode e). Such a negated telic is atelic but not strictly atelic. This is perfectly compatible with $((\neg\neg lightning)*e')$, i.e., $((lightning)*e')$ (there’s lightning in e') for some larger e' that has e as a temporal segment (which implies that $e \sqsubseteq e'$). (**OUT_{tel}**) is an immediate consequence of Lemma 3(a).

(**IN_{ateli}**) is a consequence of the following semantic lemma.

Lemma 6. (a) If $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi_{ateli}, s)$, $s = \bigsqcup S$, and $s' \sqsubseteq s$, then there exists an $S' \in \text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi_{ateli}, s')$ such that $s' = \bigsqcup S'$. (b) If the negation of ϕ is atelic, $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^-(\phi_{ateli}, s)$, $s = \bigsqcup S$, and $s' \sqsubseteq s$, then there exists an $S' \in \text{Support}_{\mathcal{M}, \mathcal{U}}^-(\phi_{ateli}, s')$ such that $s' = \bigsqcup S'$.

The proof as usual is by simultaneous induction. The basis makes use of the assumption in the definition of a model that atelic predications and negated predications that are true for a situation are also true for its temporal segments. The (a)-part of the inductive argument for conjunctive atelic situation abstracts makes use of the inheritance of atelicity to conjuncts, of Lemmas 2(a), 1(a) and 5(a) (in that order), of the downward inheritance of temporal segmentation assumed in the definition of models, and of the commutativity and associativity of situation joins. The (a)-part of the inductive argument for universally quantified abstracts is analogous to that for conjunctive abstracts, and uses the same lemmas. The remaining (a)-cases are straightforward, and the arguments for the (b)-parts are the duals of those for the (a)-parts (with use of Lemmas 2(b), 1(b), and 5(b)).

5.6 RULES OF INFERENCE

The usual rules of FOL are sound in FOL**. For instance, substitution of equals and \forall -instantiation can be shown to be sound, by a generalization of the usual inductive argument to allow for *- and **-contexts. I will term the rules in the following group “chaining” rules, since they do not introduce new material, except that \forall -introduction introduces a disjunct and \forall -instantiation can introduce new terms.

■ Chaining rules

- Commutativity and associativity rules for \wedge, \vee

- \wedge -, \rightarrow -, and \leftrightarrow -introduction and elimination
- \vee -introduction
- DeMorgan's rules
- Quantifier negation rules
- Substitution of equals
- \forall -distribution, instantiation, and generalization
- \exists -generalization
- Modus ponens

We might also use

■ Rules that make assumptions

- *Reductio ad absurdum*
- Assumption of antecedent
- Assumption of negation of disjunct
- etc.

and of course there are other possibilities (sequent rules, semantic tableaux, etc.) In any case, the point is just to draw some distinctions, not to make a comprehensive list. The following two rules are particular to FOL^{**} and are sound:

■ Embedded weak derivation

$$\frac{\phi * \tau, \phi_c \vdash \psi_c}{(\psi * \tau)} \quad (\text{EMB})$$

■ Situation consistency

$$\frac{\phi_{c+} \vdash \square}{\neg(\phi * \tau)} \quad (\text{CONSIS})$$

In the **EMB** rule, $\phi_c \vdash \psi_c$, read as " ψ_c is *weakly derivable* from ϕ_c ", means that ψ_c is derivable from ϕ_c using *chaining rules* only, without recourse to assumption making or logical axioms. The notation ϕ_c indicates uniform conversion of a situational abstract to a formula by filling its implicit argument slots with c . More exactly, ϕ_c denotes a formula obtained from a situation abstract ϕ by adding the initial situational argument c , where c is a situational constant not appearing in any premises, to all proper atomic situation abstracts in ϕ occurring outside the scope of all '*' or '**' operators in ϕ .

The following are some simple schematic examples of inferences allowed by the **EMB** rule. The first two in effect recast the previous formulas $(\neg \wedge)^*$ and $(\neg \vee)^*$ as rules.

$$\frac{((\neg(\phi \wedge \psi)) * \tau)}{((\neg\phi \vee \neg\psi) * \tau)}, \quad \frac{((\neg(\forall x \phi)) * \tau)}{((\exists x \neg\phi) * \tau)}, \quad \frac{((\phi \wedge (\phi \rightarrow \psi)) * \tau)}{(\psi * \tau)}$$

Establishing soundness of **EMB** is basically a matter of showing for each of the chaining rules that the existence of support sets for the premises (cast as situation abstracts, and relativized to some situation) entails existence of support sets for the conclusion (similarly abstracted and relativized to the same situation). For many of the chaining rules this follows directly from the inductive clauses for support sets and the duality between positive and negative support sets. In the case of *modus ponens*, we use the fact that if an abstract is positively supported relative to a situation, then it cannot be negatively supported relative to that situation (see Lemma 3).

The **CONSIS** rule basically says that a situation cannot support inconsistent information. (Thus our notion of situations here is a realistic one – they cannot support inconsistent information any more than ordinary individuals can “support” inconsistent properties.) ϕ_{c+} in (**CONSIS**) roughly speaking is ϕ with all its tacit situation arguments (in proper atomic abstracts occurring outside the scopes of all ‘*’ and ‘**’ operators) filled in with distinct new constants, the first such constant being c . (A definition using a systematic way of filling in the new constants is given below, as needed in Lemma 7.)

The reason for the soundness of the rule is that as long as “ \vdash ” is based on sound rules of inference, inconsistent premises have no models, and a corresponding situation abstract has no support sets. To make this argument go through, we use a lemma related to the contrapositive of the soundness claim.

Lemma 7. Given an individual constant c not occurring in a situation abstract ϕ , (a) if $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^+(\phi, s)$ then $\models_{\mathcal{M}', \mathcal{U}} \phi_{c+}$ for some \mathcal{M}' that differs from \mathcal{M} only in the interpretations of constants occurring in ϕ_{c+} but not ϕ ; and (b) if $S \in \text{Support}_{\mathcal{M}, \mathcal{U}}^-(\phi, s)$ then $\models_{\mathcal{M}', \mathcal{U}} \neg\phi_{c+}$ for some \mathcal{M}' that differs from \mathcal{M} only in the interpretations of constants occurring in ϕ_{c+} but not ϕ .

A more precise definition of ϕ_{c+} that facilitates the proof is as follows. Assume that we have some fixed enumeration c_0, c_1, \dots of individual situation constants (we refer to i as the *rank* of c_i). Then for a given situation abstract ϕ and a situation constant c that does not occur in ϕ , ϕ_{c+} is the formula obtained from ϕ by inserting initial argument c into the first (leftmost) proper atomic situation abstract occurring in ϕ outside the scopes of all ‘*’ and ‘**’ operators; and then inserting the lowest-ranked situation constant that outranks c and does not occur in ϕ into the next (second from the left) proper atomic situation abstract occurring in ϕ outside the scopes of all ‘*’ and ‘**’ operators; and so on, until no further eligible atomic abstracts remain. The inductive proof of the lemma is straightforward, but it is worth mentioning that in the basis for part (b), Lemma 3 can be used; and in the induction step for conjunctions and disjunctions it is useful to note that $(\phi_{c+} \wedge \psi_{c+})$ is the same as $(\phi \wedge \psi)_{c+}$ if c' is chosen to be the lowest-ranked situation constant that is identical to c or outranks c and does not occur in ϕ_{c+} , and similarly for other truth-functional

compounds. Models for ϕ_{c+} and $\psi_{c'+}$ can then be combined in the manner required for the induction step.

The proof of the soundness of **CONSIS** is then by induction on the number of applications of **CONSIS** in the derivation $\phi_{c+} \vdash \square$, with the basis (0 applications) provided by Lemma 7 and the soundness of the remaining rules of inference.

6 RELATED WORK

An earlier discussion of the relation between Davidsonian events and the situations of Situation Semantics is that of Robert Moore, 1989. In my view, Moore's essay is best understood as an argument for distinguishing facts from Davidsonian events, based on the contrast between fact modification in sentences like '*Strangely, John sang*', and event modification in sentences like '*John sang strangely*'. As such, the argument seems reasonable; however, unlike Moore, I would not want to take the further step of identifying facts with (actual) situations, for reasons that include the contrasts (26-27) mentioned earlier.

Jerry Hobbs, 1985, advocates a Davidsonian approach in which any NL-derived predication, such as *loves(John, Mary)*, is viewed as a shorthand for a predication with an added eventuality argument, conjoined with an existence assertion for that eventuality; e.g., *loves'(E, John, Mary) \wedge Exist(E)*. One might make an analogy between the prime ' used by Hobbs and the '**' operator of FOL**, except that ' is not viewed as an operator, but just as part of the name of a predicate related to the unprimed one. Thus there is no general mechanism for introducing eventualities corresponding to complex sentences. In fact, Hobbs *et al.* (in Hobbs, 1985; Hobbs *et al.*, 1986 and other later work) attempt to avoid complex sentences (other than conjoined and existentially quantified ones) in LF by viewing all forms of modification (adjective-noun, noun-noun, adverbials, etc.) as adding conjunctive information, interpreting quantified talk as talk about "typical elements" of sets, and treating negation as a relation *not*(e_1, e_2) between eventualities (that may or may not *Exist*, i.e., be actual).

However, Hobbs' eventualities appear not only in positions where one would expect events (for instance in causal relations, or as arguments of locative predications), but also in positions where one would expect propositions (for instance as objects of belief). I don't think such a conflation of events and propositions is tenable. Just as events and situations need to be distinguished from facts, they also need to be distinguished from propositions (whatever we take the connection between facts and propositions to be). For example, one can state, entertain, and prove propositions but not events; while one can set in motion, observe, and participate in events but not propositions. Propositions, unlike events, can be true or false, while events, unlike propositions, can begin, occur and end. Such incongruities, it seems to me, would ultimately lead to unwanted consequences and contradiction for language-derived information in Hobbs' approach.

In fact, Hobbs' treatment of negation in terms of a relation between eventualities suggests that his eventualities cannot be events at all, in any natural sense. Consider, for instance, a certain event e of *John sprinting*. On any natural conception of such a physical event (even if it is only a "possible" event), e could be characterized in innumerable ways; for instance, it could be characterized more generically as an event of *John running*, or an event of *John moving*. (This is certainly the case in neo-Davidsonian theories that organize event types hierarchically.) But then what is a situation n constrained by Hobbs' relation $\text{not}(n, e)$? Is it a situation of *John not sprinting*, or one of *John not running*, or one of *John not moving*? The answer can't be "all of these", since a situation of *John not sprinting* might well be one where he is running (at a more leisurely pace), and a situation of *John not running* might well be one where he is moving (e.g., walking). In other words, there is no way to recover, from a *particular event* characterized by a sentence ϕ , the situations characterized by $\neg\phi$. Events just do not carry any particular description "on their sleeves", and so they do not reveal what situations fall under the negated descriptions. This is in contrast with the case of propositions, where the concept of a proposition that is the denial (or complement) of another makes perfect sense.

Robert Wilensky, 1991, acknowledges the need for quantified and negative situations, but finds fault with the EL approach (as first presented in Schubert and Hwang, 1989) and offers an alternative. Concerning the EL representation of sentences such as '*Everyone looked at Mary*', which consists of a universal in the scope of '**', Wilensky says that "we cannot explicitly represent the fact that Mary was the patient of any number of looking actions" or relate these individual actions to the overall event. In fact, however, the following holds,

$$(\forall x \phi)**\eta \rightarrow \forall x \exists e \sqsubseteq \eta. \phi**e,$$

which expresses precisely what Wilensky is looking for. This is a consequence of $(\forall)**$ and has also been valid in every past version of EL, because the semantics of '**' has always been defined so that the truth of $(\forall x \phi)**\eta$ guarantees the truth of $(\forall x \phi)$ in the situation denoted by η , and the semantics of universal quantification has always been defined so that the truth of $(\forall x \phi)$ in a situation guarantees the existence of a subsituation characterized by ϕ , for each possible value of x (e.g., see Hwang and Schubert, 1993b).¹² Wilensky's own proposed LF for quantified sentences essentially corresponds to the right-hand side of the above conditional (*modulo* his use of neo-Davidsonian thematic relations), with an existential quantifier for η and an added predication stating that η is of type *Complex-event*. But this suffers from the same "loose coupling" problem that motivated the introduction of '**': it merely says that η has certain subevents, without saying that these subevents comprise *all* of η . Thus the argument of Section 2 against loose coupling of events to sentences in causal talk applies to Wilensky's representation.

Wilensky also agrees that "non-events seem to be ... as causally culpable as 'real' events". But again he proposes his own representation of the

¹²However, our writings on EL have not made this particular consequence explicit.

connection between a negative predication $\neg P(x_1, \dots, x_n)$ and the "non-event" it introduces, namely (suppressing thematic roles)

$$\exists n, e. \text{Non-event}(n) \wedge \text{negated-event}(e, n) \wedge P(e, x_1, \dots, x_n).$$

Here *negated-event*(e, n) presumably says much the same as Hobbs' *not*(n, e), i.e., n is the sort of condition that holds if e doesn't occur. However, the rest of the conjunction asserts that e (the positive event) occurs after all, since Wilensky (unlike Hobbs, judging from the examples he offers) adheres to the conventional view that existential quantification of an event of a specified type with specified participants entails a commitment to the actual occurrence of the event. This problem does not arise in EL or FOL**, despite the existential import of positive event predictions. Moreover, even if we take Wilensky's events to be merely possible, not necessarily actual, there remains the problem pointed out for Hobbs' proposal, that instances of event types do not reveal the corresponding negative event types.

Finally, Wilensky faults EL for not breaking predications derived from NL sentences into thematic relations (termed *aspectuals* by him). However, it seems to me that the arguments for introducing '*' and '**' into a logical-form language for NL are entirely independent of whether we subscribe to thematic relations or not. For instance, instead of coding the LF of '*Mary yawned*' as

$$\exists e (\text{yawn}(\text{Mary})**e),$$

we could formulate it thematically as

$$\exists e (\text{yawn} \wedge \text{experiencer}(\text{Mary}))**e.$$

(Since *yawn* in this version has just a situation argument, the corresponding situation abstract has no arguments.) Wilensky considers such examples, but places the thematic relations outside the scope of '**'. He correctly observes that this leads to an impoverished characterization of the event; but such externalization of thematic relations would be at odds with our strategy of associating events with sentences (or at least with English clauses) as a whole.

7 CONCLUDING REMARKS

I have argued that LFs for natural language require a way of representing complex situations evoked by complex sentences (as in Situation Semantics), but at the same time these situations must be as tightly linked to sentences as Davidsonian (or Reichenbachian) events, in order to represent causation correctly. I then proposed an extension of FOL called FOL**, containing a situation description operator '*' similar to the "supports" relation of Situation Semantics, as well as a situation characterization operator '**' that couples events to sentences as tightly as Davidson's approach, but without restriction to atomic predication. FOL** has a well-defined semantics founded on the idea that sets of basic parts of a given situation support a situation abstract relative to that given situation. This idea provides a linkage between Davidsonian event semantics and Situation Semantics, capturing many of the essential features of both.

The semantics of FOL** provides a formal justification for numerous intuitively natural laws and inferences concerning the connections among situations and between sentences and situations. Since FOL** can also capture,

in a more satisfactory way than previous approaches, the causal relationships implicit in ordinary language (especially narratives), it should prove useful as a core component of a logical-form language for NLU. Theoretical extensions currently under consideration include strengthening of the temporal syntax and semantics and further work on proof theory, particularly establishment of completeness.

In a sense, FOL^{**} is already a "field-tested" representation, since something quite similar to it is a part of the EL knowledge representation. Over the past 10 years, EL has been used as a logical-form language in various startup applications, including the TRAINS 91-93 interactive transportation planning systems (Allen and Schubert, 1993; Traum et al., 1996), a message processing application for airplane repair reports (Namioka et al., 1992), and systems for understanding and answering questions about miscellaneous story fragments (small excerpts from the fairy tale Little Red Riding Hood and from terrorist incident reports). Except for the TRAINS application, this work has made use of the EPILOG system, a full implementation of EL designed to support NLU and input-driven and goal-driven inference for large knowledge bases. For a recent comprehensive overview of both EL and the EPILOG system and applications see (Schubert and Hwang, 2000).

However, I should point out that the semantics of '*' and '**' assumed (to date) in EL and EPILOG differs from the one developed here. In particular, the EL semantics is intensional, treating the semantic values of sentences (including ones with a top-level '*' or '**' operator) as partial 0, 1-valued functions on situations; furthermore, the meaning of '**' is regarded as derivative from that of '*', with predicate intensions supplying partial-description information about situations, rather than characterizing information (as in the present approach). Still, the laws governing '*' '**' ' \sqcup ', ' \sqsubseteq ', and ' \preceq ' are mostly the same in EL and FOL^{**}. In future work, we intend to adjust the EL semantics so that EL becomes a true syntactic and semantic extension of FOL^{**}. The corresponding adjustments in the EPILOG system should be quite feasible, since the basic laws of EL are represented in EPILOG as axiom schemas (meaning postulates), and these are easily replaced. However, the current study indicates the need for not only some altered and additional basic laws, but also for additional inference rules (**EMB**, **CONSI**). Their implementation should significantly improve the inferential adequacy of EPILOG.

Acknowledgements

The paper owes its existence to Jack Minker's instigation (jointly with John McCarthy) and effective planning of the very timely Workshop on Logic-Based AI, LBAI'99. My thanks to David Ahn for his careful reading of the paper, uncovering several slips. Don Perlis, Norm McCain, Jack Minker, and the two anonymous referees offered perceptive and useful comments that have helped to improve the paper.

References

- Allen, J. F. and Schubert, L. K. (1993). Language and discourse in the TRAINS project. In Ortony, A., Slack, J., and Stock, O., editors, *Communication from an Artificial Intelligence Perspective*, pages 91-120. Theoretical Springer-Verlag, Heidelberg.
- Barwise, J. and Perry, J. (1983). *Situations and Attitudes*. MIT Press, Bradford Books, Cambridge, MA.
- Carlson, G. N. and Pelletier, F. J. (1995). *The Generic Book*. University of Chicago Press, Chicago.
- Davidson, D. (1967a). Causal relations. *The J. of Philosophy*, 64:691-703. Reprinted in Donald Davidson and Gilbert Harman, editors, *The Logic of Grammar*, pp. 246-254. Dickenson Publ., Encino, CA., 1975.
- Davidson, D. (1967b). The logical form of action sentences. In Donald Davidson and Gilbert Harman, editors, *The Logic of Grammar*, pp. 235-245. Dickenson Publ., Encino, CA., 1975. (Reprinted from *The Logic of Decision and Action*, Nicholas Rescher, ed., U. of Pittsburg Pr., 1967.).
- Hobbs, J. R. (1985). Ontological promiscuity. In *23rd Ann. Meet. of the Assoc. for Computational Linguistics (COLING-85)*, pages 61-69, Univ. of Chicago, Chicago, Ill.
- Hobbs, J. R., Croft, W., Davies, T., Edwards, D., and Laws, K. (1986). Commonsense metaphysics and lexical semantics. In *24th Ann. Meet. of the Assoc. for Computational Linguistics (COLING-86)*, pages 231-240, Columbia Univ., New York, NY.
- Hwang, C. H. and Schubert, L. K. (1993a). Episodic Logic: A comprehensive, natural representation for language understanding. *Minds and Machines*, 3(4):381-419.
- Hwang, C. H. and Schubert, L. K. (1993b). Episodic Logic: A situational logic for natural language processing. In Peter Aczel, David Israel, Y. K. and Peters, S., editors, *Situation Theory and its Applications*, volume 3, pages 303-338, Stanford, CA. CSLI.
- Hwang, C. H. and Schubert, L. K. (1994). Interpreting tense, aspect and time adverbials: A compositional, unified approach. In Gabbay, D. M. and Ohlbach, H. J., editors, *Proc., 1st Int'l. Conf. on Temporal Logic*, pages 238-264, Bonn, Germany. Springer-Verlag.
- 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.
- Moore, R. C. (1989). Events, situations, and adverbs. In Martins, J. and Morgado, E., editors, *Proc. of the 4th Portuguese Conf. on Artificial Intelligence*, Berlin. Springer-Verlag. Reprinted in R.C. Moore, *Logic and Representation*, (ch. 9), pages 159-170, CSLI Lecture Notes No. 39, CSLI Publications, 1995.
- Muskens, R. (1995). *Meaning and Partiality*. CSLI Books, Stanford, CA.
- Namioka, A., Hwang, C. H., and Schaeffer, S. (1992). Using the inference tool EPILOG for a message processing application. *Int'l. J. of Expert Systems*, 5(1):55-82.

- Parsons, T. (1990). *Events in the Semantic of English*. MIT Press, Cambridge, MA.
- Reichenbach, H. (1947). *Elements of Symbolic Logic*. Macmillan, New York, NY.
- Schubert, L. K. and Hwang, C. H. (1989). An episodic knowledge representation for narrative texts. In *Proc., 1st Int'l. Conf. on Principles of Knowledge Representation and Reasoning (KR'89)*, pages 444-458, Toronto, Canada.
- Schubert, L. K. and Hwang, C. H. (2000). Episodic Logic meets Little Red Riding Hood: A comprehensive, natural representation for language understanding. In Iwanska, L. and Shapiro, S. C., editors, *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*, pages 111-174. MIT/AAAI Press.
- Traum, D., Schubert, L., Poesio, M., Martin, N., Light, M., Hwang, C. H., Heeman, P., Ferguson, G., and Allen, J. (1996). Knowledge representation in the TRAINS-93 conversation system. *Int. J. of Expert Sys., special issue on Knowledge Representation and Inference for Natural Language Processing*, 9(1):173-223.
- Wilensky, R. (1991). Sentences, situations, and propositions. In Sowa, J. F., editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, pages 191-227 (ch. 6). Morgan Kaufmann, San Mateo, CA.

XIII

COMPUTATIONAL LOGIC

Chapter 19

LINEAR TIME DATALOG AND BRANCHING TIME LOGIC

Georg Gottlob,

Institut für Informationssysteme

Technische Universität Wien, Austria

gottlob@dbai.tuwien.ac.at

Erich Grädel,

Mathematische Grundlagen der Informatik

RWTH Aachen, Germany

graedel@informatik.rwth-aachen.de

Helmut Veith

School of Computer Science

Carnegie Mellon University, Pittsburgh, USA

veith@cs.cmu.edu

Abstract We survey recent results about the relation between Datalog and temporal verification logics. Datalog is a well-known database query language relying on the logic programming paradigm. We introduce *Datalog LITE*, a fragment of Datalog with well-founded negation, which has an easy stratified semantics and a linear time model checking algorithm. Datalog LITE subsumes temporal languages such as CTL and the alternation-free μ -calculus. We give easy syntactic characterizations of these temporal languages by fragments of Datalog LITE, and show that Datalog LITE has the same expressive power as the alternation-free portion of guarded fixed point logic.

Keywords: Datalog, temporal logic, computer-aided verification, linear time

1 INTRODUCTION

During the last decade, model checking has become one of the preeminent applications of logic in computer science. Temporal model checking is

a technique for verifying that a system satisfies its specifications by (i) representing the system as a Kripke structure, (ii) writing the specification in a suitable temporal logic, and (iii) algorithmically checking that the Kripke structure is a model of the specification formula. Model checking has been successfully applied in hardware verification, and has become an industrial standard tool for hardware design.

The main practical problem in model checking is the so-called state explosion problem caused by the fact that the Kripke structure represents the *state space* of the system under investigation, and thus it is of size exponential in the number of variables. The state explosion problem is alleviated by a number of techniques, in particular symbolic techniques which represent the Kripke structure by Binary Decision Diagrams (Bryant, 1986; Burch et al., 1992; McMillan, 1993), abstraction techniques (e.g. (Clarke et al., 1994; Kurshan, 1994; Clarke et al., 2000a)), and reduction techniques such as the cone of influence reduction and the partial order reduction. For recent overviews of model checking, we refer the reader to (Clarke et al., 2000b; Clarke and Schlingloff, 2000).

The temporal logics used in model checking provide a number of important properties which are crucial for the method, distinguish them from many other logics in computer science, and serve as a basis for the above mentioned methods. The most outstanding properties are fast (in fact, often linear time) model checking procedures, decidability of the satisfiability problem, and structure theorems about the models of temporal formulas such as the tree model property. Some of the most prominent examples of temporal logics are branching time logics such as computational tree logic CTL (Clarke and Emerson, 1981), the modal μ -calculus L_μ (Kozen, 1983), and its alternation-free fragment L_{μ_1} .

The contribution of this paper is to present Datalog LITE (LInear Time Extended Datalog), a powerful language based on logic programming which subsumes CTL and L_{μ_1} . Datalog LITE is a deductive query language which simultaneously inherits the intuitive semantics of Datalog and the favorable properties of temporal logics, in particular linear time model checking. Datalog LITE is intended as a first step in comparing and combining methods from two quite distinct fields. Thus, Datalog LITE has potential applications in verification systems as well as in databases. It is expected that for verification of software and e-commerce protocols as in (Abiteboul et al., 1998), such hybrid languages may be of great value.

Example 1 In the verification system SMV, the state space S of a Kripke structure is given by the possible assignments to the system variables. Thus, a system with 3 variables x, y, reset and variable domains $D_x = D_y = \{0, 1, 2, 3\}$ and $D_{\text{reset}} = \{0, 1\}$ has state space $S = D_x \times D_y \times D_{\text{reset}}$, and $|S| = 32$.

A binary transition relation E is defined by transition blocks which for each variable define its possible next value in the next time cycle, as in the following example:

init (<i>reset</i>) := 0;	init (<i>x</i>) := 0;	init (<i>y</i>) := 1;
next (<i>reset</i>) := {0, 1};	next (<i>x</i>) := case	next (<i>y</i>) := case
	<i>reset</i> = 1 : 0;	<i>reset</i> = 1 : 0;
	<i>x</i> < <i>y</i> : <i>x</i> + 1;	(<i>x</i> = <i>y</i>) \wedge (<i>y</i> = 2) : <i>y</i> + 1;
	<i>x</i> = <i>y</i> : 0;	(<i>x</i> = <i>y</i>) : 0;
	else : <i>x</i> ;	else : <i>y</i> ;
	esac ;	esac ;

Here, **next**(*reset*) := {0, 1} means that the value of *reset* is chosen nondeterministically. Such situations occur frequently when *reset* is determined by the environment, or when the model of the system is too abstract to determine the values of *reset*. For details about the SMV input language, we refer the reader to (McMillan, 1993). Typical CTL properties to be verified by the system include the following:

<i>CTL</i>	<i>Informal Semantics</i>
------------	---------------------------

AGEF <i>reset</i> = 1	<i>"From all reachable states, it is possible to reset the system in the future."</i>
------------------------------	---

EFAG <i>x</i> = 1	<i>"There exists a reachable state, after which <i>x</i> = 1 becomes an invariant."</i>
--------------------------	---

The formal semantics of CTL is defined in Section 2.

In database theory, a database is identified with a finite relational structure. A *query* associates to each input database over a given schema (vocabulary) a result consisting of one or more output relations. Queries are formulated in *query languages*. *Datalog* (cf. (Abiteboul et al., 1995; Ceri et al., 1990; Ullman, 1989)) is a very powerful, well-studied declarative query language based on Horn clause logic. Datalog adapts the paradigm of Logic Programming to the database setting. A Datalog query (Π, P) is a finite set of (recursive or nonrecursive) rules Π (i.e., a Datalog program) and a distinguished output relation P . Since queries are defined by Datalog programs, we shall usually not distinguish between queries and programs.

Example 2 The query (Π, T) , where Π consists of the rules

$$T(x, y) \leftarrow E(x, y) \quad T(x, y) \leftarrow T(x, z), E(z, y).$$

associates with any input relation *E* its transitive closure.

A brief description of Datalog is given in Section 3. Pure Datalog can be made more expressive by using *negated literals* in the rule bodies. For our purposes, there are two relevant extensions of pure Datalog by negation: Datalog with *stratified negation* introduced by (Apt et al., 1988) where negation does not interact with recursion, and the more expressive Datalog with *well-founded negation* defined by (Gelder et al., 1991).

Example 3 We continue Example 1. In order to evaluate a Datalog program over a Kripke structure, we use the transition relation *E* as input relation.

Moreover, for all atomic properties of states s such as $x = 1$, monadic relations such as $I_{x=1}$ are provided with the input. For better readability, we shall write $s.x = 1$ instead of $I_{x=1}(s)$.

Then the specifications of Example 1 can be expressed in Datalog as follows:

AGEF $reset = 1$	$\text{Reset_Reachable}(s) \leftarrow s.reset = 1$ $\text{Reset_Reachable}(s) \leftarrow E(s, s'), \text{Reset_Reachable}(s')$ $\text{BadState}(s) \leftarrow \neg \text{Reset_Reachable}(s)$ $\text{BadState}(s) \leftarrow E(s, s'), \text{BadState}(s')$ $\text{GoodState}(s) \leftarrow \neg \text{BadState}(s)$
EFAG $x = 1$	$\text{No_Invariant}(s) \leftarrow \neg s.x = 1$ $\text{No_Invariant}(s) \leftarrow E(s, s'), \text{No_Invariant}(s')$ $\text{Result}(s) \leftarrow \neg \text{No_Invariant}(s)$ $\text{Result}(s) \leftarrow E(s, s'), \text{Result}(s')$

The above programs belong to a simple fragment of both Datalog LITE and Stratified Datalog. While at first sight these programs are longer than their counterparts in CTL, they have several advantages including easy readability and reusability of (sub)programs. Moreover, we shall see that the expressive power of Datalog LITE is much higher than that of CTL and L_{μ_1} .

Research Program. The favorable logical properties of temporal logics have attracted interest from various communities. In pure logic, modal fragments of first order logic and generalizations thereof such as the bounded fragment have been studied by (Andréka et al., 1998; Grädel, 2000; Grädel and Walukiewicz, 1999), and general insight in model theoretic and algorithmic properties of temporal logics has been gained.

For databases, fast model checking procedures and decidability are natural properties of query languages which have received wide consideration in the literature. The advantages of Datalog and its capability of expressing and checking temporal properties of Kripke structures suggest to study fragments of Datalog that are suited for model checking. We believe that from the point of view of both verification and databases it is a rewarding task to investigate the exact relation between temporal logics and deductive query languages. Thus, our research program was aimed at identifying a suitable Datalog fragment which meets the following criteria:

- The fragment should be clearly syntactically identifiable, but syntactically ample enough to allow the programmer to make use of the main paradigms of Logic Programming (including, in particular, some form of negation).
- It should be expressive enough to state a large number of useful temporal properties, in particular those in CTL.

- The fragment should have linear time data complexity. This means that the complexity of evaluating a fixed query over variable input structures is $O(n)$, where n is the structure size.
- The fragment should also have low (preferably linear time) program complexity, i.e. one should be able to evaluate programs in linear time with respect to the length of the program.
- Given the increasing importance of model checking on infinite transition systems, the fragment should be able to express relevant temporal properties not just on finite Kripke structures but also on infinite ones.

In the current paper we survey recent results in this direction. We refer the interested reader to the technical report (Gottlob et al., 1998) which contains all proofs and more technical details.

Note that a mere *ad hoc* translation of CTL to Datalog does not provide us with a suitable fragment fulfilling the above requirements. The fragment suggested by such a translation is – on one hand – syntactically extremely limited, e.g., no predicate symbols of arity greater than two occur in it. On the other hand this fragment is not known to admit a linear time evaluation algorithm.

Results. We present *Datalog LITE*, a fragment of Datalog that fulfills all desired criteria. We first define the fragment *Datalog LIT* (LIT stands for linear time) as the set of all stratified Datalog queries (over arbitrary input structures) whose rules are either monadic or guarded. A *monadic* rule is a rule that contains only monadic literals. A rule is *guarded*¹ if its body contains a positive atom (the *guard*) in which all variables of the rule occur.

We prove that Datalog LIT has linear time data complexity (Theorem 1). Moreover, for bounded arity programs or for programs whose guards are all input relations, also the combined complexity is in linear time. Although Datalog LIT can express a large number of interesting temporal properties, the language is not powerful enough for expressing full CTL. For example, (as shown in Section 3, Theorem 2), the simple CTL formula $\mathbf{AF}\varphi$ (“on all paths, eventually φ ”) cannot be expressed in Datalog LIT. Consequently, we define an extended language *Datalog LITE*. The latter is obtained from Datalog LIT by including the possibility of using a limited (guarded) kind of universal quantification in the rule bodies.

A *generalized literal* G is an expression of the form $(\forall y_1 \cdots y_n. \alpha)\beta$ where α and β are atoms, and the free variables in β also occur in α . The intended meaning of $(\forall y_1 \cdots y_n. \alpha)\beta$ is its standard first order semantics, that is, $\forall y_1 \cdots y_n(\alpha \rightarrow \beta)$. (For a precise definition, see Section 3.) It is important to note that generalized literals are not just artificial “plug ins” to Datalog, but that they can be expressed in Datalog with well-founded negation. In fact, universal quantification can be easily resolved by double negation under the

¹This notion is completely unrelated to the concept of *guarded Horn clause* as defined in (Murakami, 1990).

well-founded semantics. Thus Datalog LITE can be considered a fragment of Datalog with well-founded negation. Datalog LITE has the following important properties:

- For fixed Datalog LITE programs model-checking is feasible in linear time (Theorem 5). This remains true even for variable Datalog LITE programs whose predicate-arithes are bounded by a constant or whose guards are all input atoms.
- Datalog LITE can be easily embedded into well-founded Datalog. Thus, Datalog LITE can be considered an intuitive yet powerful fragment of well-founded Datalog with linear time model checking.
- We show that semantically, every Datalog LITE program is equivalent to a Datalog LITE program where all guards are input atoms; for arbitrary programs, the translation requires exponential time. In fact, evaluating variable Datalog LITE programs with unbounded predicate arities is EXPTIME- complete. (Proposition 1).
- Datalog LITE is equivalent in expressive power to alternation-free guarded fixed point logic, a natural fragment of the guarded fixed point logic μGF which has recently been studied by (Grädel and Walukiewicz, 1999); several important properties like satisfiability in EXPTIME and a generalized tree model property are thus obtained for Datalog LITE.
- Propositional multi-modal logic, CTL, the alternation free modal μ -calculus, and guarded first order logic each correspond to well-defined and syntactically simple subfragments of Datalog LITE.

Hence, Datalog LITE (with bounded arities) is at least as expressive as CTL and the alternation-free μ -calculus L_{μ_1} . Actually Datalog LITE is more expressive than these logics. For example, it is obvious that *past modalities* (cf. (Vardi, 1998)) are expressible in Datalog LITE. Furthermore Datalog LITE can define new modalities, e.g. via Boolean combinations of existing ones.

Related work. The definitions of guarded rules and generalized literals are motivated by the notion of *guarded quantification* as introduced by (Andréka et al., 1998) (see Section 7). The variables that appear in the body but not in the head of a Datalog rule are implicitly existentially quantified. The condition that all variables of the rule appear in one atom implies that this existential quantification is guarded in the sense of (Andréka et al., 1998). Similarly the variable occurrence condition in the definition of generalized literals implies that the universal quantifier is guarded in this sense. In fact we will see that Datalog LITE can be viewed as a clausal presentation of the alternation-free fragment of guarded fixed point logic, cf. (Grädel and Walukiewicz, 1999).

It is not new that CTL, the modal μ -calculus and related formalisms can be translated into logic programming. Such translations were carried out by (Ramakrishnan et al., 1997; Cui et al., 1998; Charatonik and Podelski, 1998). In a recent paper by (Immerman and Vardi, 1997) it is shown how various

temporal logics can be translated into first order logic augmented by a transitive closure construct (FO+TC). Since the latter is expressible in Datalog, that approach implicitly yields a translation into Datalog. Note, however, that all previously proposed translations are ad hoc translations of existing modal model-checking formalisms. None of them identifies an ample fragment of Datalog with guaranteed linear time data complexity suited for model checking. To our best knowledge, the first such fragment is Datalog LITE.

2 MODAL AND TEMPORAL LOGICS

In this subsection, we recall the definitions of some logics commonly used in verification. For more background on temporal logics, the reader is referred to (Emerson, 1990; Clarke and Schlingloff, 2000). The formulae of the logics we describe here are evaluated on Kripke structures (with one or more transition relations) at a particular state.

A Kripke structure \mathcal{K} is usually given as a tuple $(S, (E_a)_{a \in A}, L)$ where S is the state space, A is a set of actions, E_a is the transition relation associated with action a , and $L : S \rightarrow 2^{\{P_1, \dots, P_k\}}$ is a labeling function that assigns a set of atomic propositions to each state. \mathcal{K} can be trivially identified with the relational structure $(S, (E_a)_{a \in A}, P_1, \dots, P_k)$ where P_1, \dots, P_k are monadic relations representing all the atomic propositions and for each $s \in S$, $P_i(s)$ is true iff $P_i \in L(s)$.

Given a formula ψ and a Kripke structure \mathcal{K} with state v , we will write $\mathcal{K}, v \models \psi$ to denote that the formula ψ holds in the Kripke structure \mathcal{K} at state v . Such formulae are sometimes called state formulae. A state formula ψ can also be understood as a (monadic) query that associates with each Kripke structure \mathcal{K} the set of states v such that $\mathcal{K}, v \models \psi$.

Propositional modal logic. The simplest of these formalisms is propositional modal logic ML.

Syntax of ML. The formulae of ML are defined by the following rules.

- (S1) Each atomic proposition P_i is a formula.
- (S2) If ψ and φ are formulae of ML, then so are $(\psi \wedge \varphi)$ and $\neg\psi$.
- (S3) If ψ is a formula of ML and $a \in A$ is an action, then $\langle a \rangle \psi$ and $[a] \psi$ are formulae of ML.

If there is only one action, hence one transition relation in the Kripke structure one simply writes \Box and \Diamond for the modal operators.

Semantics of ML. Let ψ be a formula of ML, $\mathcal{K} = (S, (E_a)_{a \in A}, P_1, P_2, \dots)$ a Kripke structure, and $v \in S$ a state. In the case of atomic propositions, $\varphi = P_i$, we have $\mathcal{K}, v \models \varphi$ iff $v \in P_i$. Boolean connectives are treated in the natural way. Finally for the semantics of the modal operators we put

$$\begin{aligned}\mathcal{K}, v \models \langle a \rangle \psi &\text{ iff } \mathcal{K}, w \models \psi \text{ for some } w \text{ such that } (v, w) \in E_a \\ \mathcal{K}, v \models [a] \psi &\text{ iff } \mathcal{K}, w \models \psi \text{ for all } w \text{ such that } (v, w) \in E_a\end{aligned}$$

CTL and CTL^{*}. The logics CTL (computation tree logic) and CTL^{*} extend ML by path quantification and temporal operators on paths. We define first CTL^{*}, and obtain CTL as a fragment of CTL^{*}. For computation tree logics, one usually restricts attention to Kripke structures with a single binary relation which is required to be *total*, i.e. for every state v there exists a least one state w that is reachable from v . (Note that this definition of totality which is common in model checking differs from standard terminology.)

Syntax of CTL^{}.* We have state formulae and path formulae in CTL^{*}. The path formulae are auxiliary formulae to make the inductive definition more transparent. Actually we are interested only in state formulae. The state formulae are defined by the rules (S1) and (S2) (read everywhere state formula instead of formula) and the rule

- (S4) If ϑ is a path formula, then $\mathbf{E}\vartheta$ and $\mathbf{A}\vartheta$ are state formulae.

The path formulae are defined by

- (P1) Each state formula is a path formula.

- (P2) If ϑ, η are path formulae, then so are $(\vartheta \wedge \eta)$ and $\neg\vartheta$.

- (P3) If ϑ, η are path formulae, then so are $\mathbf{X}\vartheta$ and $(\vartheta \mathbf{U} \eta)$.

Semantics of CTL^{}.* Consider a total Kripke structure $\mathcal{K} = (S, E, P_1, P_2, \dots)$. An *infinite E-path* (or just an infinite path) is an infinite sequence $\bar{s} = s_0, s_1, s_2, \dots$ of states such that for all $i \geq 0$, $(s_i, s_{i+1}) \in E$ for all i . \bar{s}^i denotes the suffix path $s_i, s_{i+1}, s_{i+2}, \dots$. As above, we write $\mathcal{K}, s_0 \models \psi$ to denote that the state formula ψ is true in \mathcal{K} at state s_0 . We write $\mathcal{K}, \bar{s} \models \vartheta$ to denote that the path formula ϑ is true on the infinite path \bar{s} in \mathcal{K} . We define \models inductively as follows. For formulae defined via rules (S1), (S2) and (P2) the meaning is obvious.

- (S3) $\mathcal{K}, s_0 \models \mathbf{E}\vartheta$ iff there exists an infinite path \bar{s} from s_0 such that $\mathcal{K}, \bar{s} \models \vartheta$.
 $\mathcal{K}, s_0 \models \mathbf{A}\vartheta$ iff $\mathcal{K}, \bar{s} \models \vartheta$ for all infinite paths \bar{s} that start at s_0 .

- (P1) $\mathcal{K}, \bar{s} \models \psi$ iff $\mathcal{K}, s_0 \models \psi$ where s_0 is the starting point of \bar{s} .

- (P3) $\mathcal{K}, \bar{s} \models (\vartheta \mathbf{U} \eta)$ iff for some $i \geq 0$, $\mathcal{K}, \bar{s}^i \models \eta$ and $\mathcal{K}, \bar{s}^j \models \vartheta$ for all $j < i$.
 $\mathcal{K}, \bar{s} \models \mathbf{X}\vartheta$ iff $\mathcal{K}, \bar{s}^1 \models \vartheta$.

CTL is the fragment of CTL^{*} without nestings and without Boolean combinations of path formulae. Thus, rules (P1) – (P3) are replaced by

- (P0) If ψ, φ are state formulae, then $\mathbf{X}\psi$ and $(\psi \mathbf{U} \varphi)$ are path formulae.

Actually, for CTL we can disregard path formulae altogether and equivalently define the set of state formulae by the rules (S1), (S2) of ML together with rules

- (S3') If ψ is a (state) formula, then so are $\mathbf{EX}\psi$ and $\mathbf{AX}\psi$.

- (S4) If ψ and φ are (state) formula, then so are $\mathbf{E}(\psi \mathbf{U} \varphi)$ and $\mathbf{A}(\psi \mathbf{U} \varphi)$.

Note that indeed the formulae $\mathbf{EX}\psi$ and $\mathbf{AX}\psi$ defined by (S3') are equivalent to $\Diamond\psi$ and $\Box\psi$, respectively.

Other temporal operators are defined by abbreviations as follows:

- $\mathbf{EF}\psi$ abbreviates $\mathbf{E}(\mathbf{trueU}\psi)$,
- $\mathbf{AG}\psi$ abbreviates $\neg\mathbf{EF}\neg\psi$,
- $\mathbf{AF}\psi$ abbreviates $\mathbf{A}(\mathbf{trueU}\psi)$, and
- $\mathbf{EG}\psi$ abbreviates $\neg\mathbf{AF}\neg\psi$.

The μ -calculus L_μ . The propositional μ -calculus L_μ is propositional modal logic augmented with least and greatest fixed points. It subsumes almost all of the commonly used modal logics, in particular LTL, CTL, CTL*, PDL and also many logics used in other areas of computer science, for instance most description logics.

Syntax of L_μ . The μ -calculus extends propositional modal logic ML (including propositional variables X, Y, \dots) by the following rule for building fixed point formulae.

(S μ) If ψ is a formula in L_μ , and X is a propositional variable that does not occur negatively in ψ , then $\mu X.\psi$ and $\nu X.\psi$ are L_μ formulae.

An L_μ -formula is *alternation-free* if no occurrence of any propositional variable is inside the scope of both a μ - and a ν -operator.

Semantics of L_μ . The semantics of the μ calculus is given as follows. A formula $\psi(X)$ with propositional variable X defines on every Kripke structure \mathcal{K} (with state set S) an operator $\psi^\mathcal{K} : 2^S \rightarrow 2^S$ assigning to every set $X \subseteq S$ the set

$$\psi^\mathcal{K}(X) := \{s \in S : (\mathcal{K}, X), s \models \psi\}.$$

If X occurs only positively in ψ , then $\psi^\mathcal{K}$ is *monotone* for every \mathcal{K} , and therefore has a least and a greatest fixed point. Now we put

(S μ) $\mathcal{K}, s \models \mu X.\psi$ iff s is contained in the least fixed point of the operator $\psi^\mathcal{K}$.
Similarly $\mathcal{K}, s \models \nu X.\psi$ iff s is contained in the greatest fixed point of $\psi^\mathcal{K}$.

3 FROM DATALOG TO DATALOG LIT

Definition 1 A *Datalog rule* is an expression of the form $H \leftarrow B_1, \dots, B_r$, $r \geq 1$, where H , the *head* of the rule, is an atomic formula $Ru_1 \dots u_s$, and B_1, \dots, B_r , the *body* of the rule, is a collection of literals (i.e. atoms or negated atoms) of the form $Sv_1 \dots v_t$ or $\neg Sv_1 \dots v_t$ where $u_1, \dots, u_s, v_1, \dots, v_t$ are variables. The relation symbol R is called the *head predicate* of the rule. Note that we use the symbol \neg to denote default negation as common in database theory; in AI, the symbol *not* is more common.

We first define *basic Datalog programs* where negation is applied only to input relations. A basic Datalog program Π is a finite collection of rules

such that none of its head predicates occurs negated in the body of any rule. The predicates that appear only in the bodies of the rules are called *extensional* or *input predicates*. Given a structure \mathfrak{A} over the vocabulary of the extensional predicates, the program computes, via the usual fixed point semantics, an interpretation for the head predicates. A Datalog query is a pair (Π, Q) consisting of a Datalog program Π and a designated head predicate Q of Π . With every structure \mathfrak{A} , the query (Π, Q) associates the result $(\Pi, Q)[\mathfrak{A}]$, i.e. the interpretation of Q as computed by Π on input \mathfrak{A} . For details, we refer to (Abiteboul et al., 1995).

Note that atoms may be written in the form Axy or in the form $A(x, y)$. Usually, we shall prefer the first form in the context of proofs and definitions (where atoms are usually denoted by single letters), and the second form in real programs as in Example 3, where atoms are denoted by longer names like Invariant.

Definition 2 A *stratified Datalog program* is a sequence $\Pi = (\Pi_0, \dots, \Pi_r)$ of basic Datalog programs which are called the *strata* of Π , such that Π_0 is a basic Datalog program, and every Π_i is a basic Datalog program whose extensional predicates are either extensional in the entire program Π or are head predicates of a lower stratum. Thus, if a head predicate of stratum Π_j occurs *positively* in the body of a rule of stratum Π_i , then $j \leq i$, and if a head predicate of stratum Π_j occurs *negatively* in the body of a rule of stratum Π_i , then $j < i$. The semantics of a stratified program is defined stratum by stratum. Hence, once the lower strata are evaluated, we can compute the interpretation of the head predicates of Π_i as in the case of basic Datalog-programs.

Definition 3 A Datalog rule is *monadic* if each of its literals has at most one free variable. Note that this does not necessarily imply that only unary predicates are used. We permit the appearance of literals $(\neg)Sv_1 \dots v_k$ with $k > 1$ that may contain repeated occurrences of a single variable. A Datalog rule is *guarded* if its body contains a positive atom (the *guard* of the rule) in which all free variables of the rule occur. A Datalog LIT program is a stratified Datalog program whose rules are either monadic or guarded.

We adopt the convention that the input relations of a query are given by lists of tuples. We shall refer to this representation as the *list representation* of the input structure.

Definition 4 Let \mathfrak{A} be a finite relational structure. Then the size $|\mathfrak{A}|$ of \mathfrak{A} is the sum of the sizes of the tuples contained in the relations of \mathfrak{A} , plus the number of domain elements.

In our algorithms we shall also use the *array representation*, which is more common in finite model theory. There, the characteristic membership function of a relation is stored in an array whose dimension equals the arity of the relation. (For graphs, this means that the graph is described by its adjacency matrix.) With array representation, membership of an individual tuple can be checked in constant time on Random Access Machines (cf. (van Emde Boas,

1990)), but on the other hand, we obtain a non-realistic notion of linear time computability when the adjacency matrix is sparse. It is easy to see that a Random Access Machine can translate the list representation into array representation in linear time.

Theorem 1 The time complexity of model checking (i.e., evaluation) of Datalog LIT is

1. linear in the input size and the program size for programs where all guards are input relations.
2. linear in the input size and the program size for bounded arity programs.
3. linear in the input size and exponential in the program size for arbitrary programs.

We defer the proof to Section 5, where it follows from a more general result. In the rest of this section we show that Datalog LIT is too restricted to express important linear time computable properties. We exemplify these restrictions by the well-foundedness query which is linear time computable and important in many applications. (Recall that a vertex x is well-founded, if no cycle can be reached from x , and no infinite path originates at x .)

Example 4 Well-foundedness statements for subgraphs of the transition relation are expressed in CTL by formulae of the form $\mathbf{AF}\varphi$ and in the μ -calculus by $\mu X.\varphi \vee \square X$, where φ is any (non-contradictory) formula (for instance, an atomic proposition).

Well-foundedness in finite graphs. It is easy to see that in finite graphs, detecting well-foundedness essentially reduces to detecting cycles in the graph; in full stratified Datalog, this is done by defining the transitive closure of the graph, i.e. by using a binary relation as intermediate result. The following results state that this is not possible in Datalog LIT, and that indeed Datalog LIT does not express well-foundedness.

Theorem 2 1. Over finite structures, the well-foundedness query cannot be expressed by stratified Datalog programs with only monadic head predicates.

2. Over Kripke structures, every Datalog LIT query with monadic output predicate is equivalent to a Datalog LIT query where all head predicates are monadic.

Proof of 1. Let Π be a stratified Datalog program with a single binary input predicate E and monadic head predicates H_1, \dots, H_r . Then there exist formulae $\chi_{H_i}(x)$ in monadic second-order logic (MSO) which are equivalent to the queries (Π, H_i) . Suppose, towards a contradiction, that (Π, H_1) computes the well-founded elements of the input graph, that is, the set of nodes v such that the graph contains no infinite E -path originating in v .

For each n , let S_n be the graph $(\{0, \dots, n\}, \{(j, j+1) : 0 \leq j < n\})$ (i.e. the path of length n). The evaluation of Π on S_n leads to the word structure $W_n = (S_n, \chi_{H_1}^{S_n}, \dots, \chi_{H_r}^{S_n})$ of vocabulary $\tau := (E, H_1, \dots, H_r)$. For two ordered structures \mathfrak{A} and \mathfrak{B} , let $\mathfrak{A} \triangleleft \mathfrak{B}$ denote the ordered sum of \mathfrak{A} and \mathfrak{B} , cf. (Ebbinghaus and Flum, 1999). (Note that \triangleleft is associative.) Moreover, let \mathfrak{A}^k denote the k -fold ordered sum of \mathfrak{A} .

Let L be the language $\{W_n : n \in \mathbb{N}\}$, and let ρ be the MSO-sentence $\bigwedge_{1 \leq i \leq k} (\forall x. \chi_{H_i}(x) \leftrightarrow H_i x)$. Then for every word structure W of vocabulary τ , $W \models \rho$ if and only if W is isomorphic to some $W_i \in L$. Hence, by Büchi's Theorem, L describes a regular language. Since L is infinite, the Pumping Lemma implies that there exist word structures W_i, W_j, W_k , such that for all $\ell \geq 1$, $W_i \triangleleft W_j^\ell \triangleleft W_k \in L$. Note that we can without loss of generality choose i, j and k arbitrarily large.

We conclude that for sufficiently large i, j, k the Datalog program Π cannot distinguish input structures of the form $S_{i+\ell j+k}$ from the root. Now consider the structure $\mathfrak{A} = S_i \triangleleft S_j \triangleleft S_j \triangleleft S_k$. In \mathfrak{A} , let c be the node connecting the two copies of S_j , and obtain a second structure \mathfrak{B} by adding a cycle of length j to the node c .

Obviously, the root of \mathfrak{A} is well-founded, while the root of \mathfrak{B} is not. We claim that Π does not distinguish \mathfrak{A} and \mathfrak{B} on their common elements. To see this, we consider the contribution of the strata Π_1, \dots, Π_p of Π . For simplicity, let us identify relations on the graphs with colors, and let m be the number of variables of Π .

We show that after evaluation of each stratum of Π , the common elements of \mathfrak{A} and \mathfrak{B} are colored identically, and the j -cycle in \mathfrak{B} is colored isomorphically to the copies of S_j in \mathfrak{A} and \mathfrak{B} . Suppose that this condition is satisfied before the evaluation of stratum Π_i (clearly this is the case for $i = 1$).

Each stratum corresponds to an existential least fixed point formula, and every Datalog rule expresses an existential first-order statement which is positive in the new colors of that stratum. By induction hypothesis the m -neighborhoods around corresponding elements of \mathfrak{A} and \mathfrak{B} satisfy the same existential statements concerning the old colors. Therefore, for sufficiently large i, j, k , the loop and the copies of S_j are colored isomorphically. Operationally, one can imagine that first the substructure \mathfrak{A} of \mathfrak{B} is colored, and then the loop has to be colored identically, since the existential statements are true on the loop if they were true on S_j . \square

Corollary 1 Datalog LIT does not express the well-foundedness query. Hence, Datalog LIT does not express all of CTL.

Well-foundedness in infinite graphs. The intuitive reason why, over infinite structures, well-foundedness is not expressible by stratified programs is that they require recursion through a universal statement, whereas Datalog (even stratified Datalog) has only recursion through existential statements. We establish a more general result, in terms of an appropriate fragment of infinitary logic.

Definition 5 The logic $L_{\omega_1\omega}$ extends first-order logic by the possibility to take disjunctions $\bigvee \Phi$ and conjunctions $\bigwedge \Phi$ over countable sets Φ of formulae. Inside this logic we define a hierarchy of fragments $L(\Sigma_k)$ and $L(\Pi_k)$ for $k < \omega$. The bottom level $L(\Sigma_0) = L(\Pi_0)$ is the set of quantifier-free formulae in $L_{\omega_1\omega}$. Further, for every k , $L(\Pi_k) := \{\neg\varphi : \varphi \in L(\Sigma_k)\}$ and $L(\Sigma_{k+1})$ is the class of formulae $\bigvee \Phi$ where Φ is a countable subset of $\{\exists x_1 \dots \exists x_m \varphi : m < \omega, \varphi \in L(\Pi_k)\}$.

Lemma 1 Every query in stratified Datalog is equivalent to some formula in $\bigcup_{k < \omega} L(\Sigma_k)$.

Theorem 3 Over countable structures, well-foundedness statements are not expressible in $\bigcup_{k < \omega} L(\Sigma_k)$. In particular they are not expressible by stratified Datalog programs.

4 DATALOG LITE

The linear time evaluation of Datalog LIT can be extended to Datalog LITE, which is obtained from Datalog LIT by introducing a new kind of literal:

Definition 6 A *generalized literal* G is an expression of the form $(\forall y_1 \dots y_n . \alpha)\beta$ where α and β are atoms, and $\text{free}(\alpha) \supseteq \text{free}(\beta)$. The free variables $\text{free}(G)$ of G are given by $\text{free}(\alpha) - \{y_1, \dots, y_n\}$.

The intended meaning of $(\forall y_1 \dots y_n . \alpha)\beta$ is its standard first-order semantics, that is, $\forall y_1 \dots y_n (\alpha \rightarrow \beta)$. Therefore, when the generalized literal is used within a program Π , the notion of stratification has to be adapted in such a way that for $(\forall \bar{y}. R\bar{x}\bar{y})S\bar{x}\bar{y}$, the occurrences of R are to be regarded as negative, and the occurrences of S are positive. In a stratified program, R must therefore belong to a lower stratum while S can be from the current stratum, too. In the context of a rule, the generalized literal is regarded as a negative literal. Since the notion of free variables is well-defined for generalized literals, Definition 3 is applicable to rules with generalized literals.

Definition 7 A Datalog LITE program is a Datalog LIT program containing (unnegated) generalized literals where the notions of guardedness and stratification are extended to generalized literals as described above.

Note that according to this definition, generalized literals cannot be used as guards. Formally, the (operational) semantics of Datalog Lite programs can be obtained from the standard semantics of Datalog as follows.

The program is evaluated bottom-up stratum by stratum. After the evaluation of each stratum, the predicate values of all predicates at this stratum and at lower strata are fixed (i.e., their interpretation is fixed). The evaluation proceeds exactly as for stratified Datalog programs except that every generalized literal $L(\bar{y}) := (\forall \bar{x}. \alpha(\bar{x}, \bar{y}))\beta(\bar{x}, \bar{y})$ is instantiated (for any tuple \bar{d}

interpreting \bar{y}) by the conjunction

$$\bigwedge_{\bar{c}:\alpha(\bar{c}, \bar{d}) \text{ true}} \beta(\bar{c}, \bar{d}).$$

Remarks.

1. Datalog LITE is indeed stronger than Datalog LIT, since the program $Wx \leftarrow (\forall y. Exy)Wy$ computes the well-foundedness query on both finite and infinite structures.
2. In a non-recursive query (Π, W) , universal quantification can be rewritten by double negation, obtaining an equivalent query $(\Pi^{\neg\neg}, W)$. For example, the single rule query $(\Pi, W) : Wx \leftarrow (\forall z. Exz)Mz$ is rewritten by the query $(\Pi^{\neg\neg}, W)$ which contains the two rules $Wx \leftarrow \neg W\neg x$ and $W\neg x \leftarrow Exz, \neg Mz$. For recursive Π however, $\Pi^{\neg\neg}$ will in general not be stratified. The well-founded semantics by (Gelder et al., 1991) assigns the correct meaning to the query.

Theorem 4 The rewriting operation $\Pi \mapsto \Pi^{\neg\neg}$ is a conservative embedding of Datalog LITE into well-founded Datalog.

Thus, up to trivial rewriting operations, Datalog LITE is a fragment of well-founded Datalog; by the following result, we have thus identified a linear time computable fragment of well-founded Datalog.

Theorem 5 The complexity results about Datalog LIT (Theorem 1) hold also for Datalog LITE.

We shall see later (cf. Lemma 3) that surprisingly, the use of head predicates as guards does not increase the expressive power of Datalog LITE. Therefore, from the point of expressive power one can always use Datalog LITE with input guards. Head predicates as guards permit writing programs in a much more compact way. By the following proposition, the exponential time bound for this case is indeed optimal:

Proposition 1 With respect to data complexity, Datalog LIT and Datalog LITE are PTIME -complete. With respect to program complexity, they are EXPTIME -complete for unbounded arity, and remain PTIME -complete for bounded arity and for programs with input guards only.

Proof Sketch. Since Datalog LITE is obviously contained in fixed point logic whose data complexity is PTIME, membership follows for both formalisms. For hardness, it is easy to see that the problem *Monotone Circuit Value* is expressible in Datalog LIT. For program complexity, the result for the bounded case follows immediately from the fact that evaluating ground Horn clause problems is PTIME -complete. Since this language is a trivial fragment of Datalog LIT, hardness follows. Membership is a trivial consequence of Theorem 1. For general program complexity, it is again sufficient to show hardness

for Datalog LIT; since we know that Monotone Circuit Value is expressible in Datalog LIT, the general method for program complexity worked out in (Gottlob et al., 1999) is applicable. For details, consult (Gottlob et al., 1999, Section 6.4) and (Gottlob et al., 1998). \square

5 LINEAR TIME ALGORITHMS

This section outlines the proof of Theorems 1 and 5. We proceed in two phases: (1) we show that we can reduce the problem to programs with input guards only, and (2) we solve the problem for this restricted case.

Phase 1: Locality Invariance and Input Guards. Given a structure \mathfrak{A} , we define the binary relation $E^{\mathfrak{A}}$ on \mathfrak{A} by

$E^{\mathfrak{A}} = \{(a, b) : \text{there is a tuple in a relation in } \mathfrak{A} \text{ containing both } a \text{ and } b\}$.
The graph $(|\mathfrak{A}|, E^{\mathfrak{A}})$ is called the *Gaifman graph* of \mathfrak{A} .

Lemma 2 Let Π be a Datalog LITE program with head predicates T_1, \dots, T_r , and let $T_1^{\mathfrak{A}}, \dots, T_r^{\mathfrak{A}}$ be the relations computed by Π on input structure \mathfrak{A} . Then the Gaifman graphs of \mathfrak{A} and $(\mathfrak{A}, T_1^{\mathfrak{A}}, \dots, T_r^{\mathfrak{A}})$ coincide.

Thus, the new relations computed by a Datalog LITE program do not change the Gaifman graph of the input structure. We shall refer to this property of programs as the *locality invariance* of Datalog LITE. The most important application of locality invariance is the following lemma:

Lemma 3 Every Datalog LIT and Datalog LITE program is equivalent to a program where all guards are input atoms. Moreover, the time complexity of eliminating non-input guards is

1. linear in the program size if the program has bounded arity.
2. exponential in the program size if the program has unbounded arity.

Proof Sketch. The proof is based on the observation that due to locality invariance, all head predicates contain only tuples which are obtained as permutations of subtuples of input relations. Hence, every rule can be replaced by new rules obtained by adding an input relation to the body as new guard, and unifying the old guard and the new guard in such a way that all the variables which occur in the old guard appear in the new guard. The number of new rules is easily seen to be exponential in the number of the old rules. The algorithm is immediately obtained from the exact proof. It is easily checked that the algorithm is exponential in the program arity, as exemplified in the following remark. \square

Example 5 Consider the rule $Rx \leftarrow Gxyz$ where G is a guard, but not an input guard, and there is only one input relation E of arity 2. Then the rule can be replaced by the following set of new rules which are all guarded by E .

$Rx \leftarrow Exy, Gxyy$	$Rx \leftarrow Eyx, Gxyy$
$Rx \leftarrow Exy, Gxxy$	$Rx \leftarrow Eyx, Gxxy$
$Rx \leftarrow Exy, Gyxz$	$Rx \leftarrow Eyx, Gyxz$
$Rx \leftarrow Exy, Gxxx$	$Rx \leftarrow Eyx, Gxxx$

Phase 2: Evaluation of Input Guard Programs. A propositional Datalog program is a program where no rule contains free variables. Thus, every rule is equivalent to a propositional Horn clause of the form $h \vee \neg b_1 \vee \dots \vee \neg b_n$. It is well-known (cf. (Dowling and Gallier, 1984; Itai and Makowsky, 1987; Minoux, 1988)) that propositional Horn clause programs can be evaluated by a variant of unit resolution in *linear time* on RAMs. We shall refer to this linear time algorithm as **EvalHORN**. **EvalHORN** makes use of a special data structure to store the Horn clauses and the result in such a way that literals can be stored, deleted and read in constant time. We shall call such a data structure a *Horn clause base* (HCB), and use it in our algorithm.

The standard way to evaluate a negation-free Datalog program Π over an input structure \mathfrak{A} is called *grounding*. This means that the language is extended by constant symbols for all domain elements from \mathfrak{A} , and the program is replaced by the rules obtained by instantiating constant symbols for the variables. The Gelfond-Lifschitz transform $GL(\Pi, \mathfrak{A})$ over \mathfrak{A} then is the set of rules obtained from the set of ground instantiations of rules in Π , such that input literals which are true over \mathfrak{A} are removed from rule bodies, and rules containing false input literals are removed completely.

The resulting propositional program is of polynomial size and can therefore be evaluated in polynomial time by **EvalHORN**. This procedure can be extended to stratified programs by evaluating the strata one by one as in (Berman et al., 1995).

Algorithm EvalLIT. The algorithm proceeds in two phases:

- (1) Monadic rules are transformed in such a way that every rule contains at most one variable. This can be easily achieved by introducing new rules with variable-free heads. If there is a e.g. a rule $Fx \leftarrow Gx, My$, it will be replaced by $Fx \leftarrow Gx, M'$, and $M' \leftarrow My$. This translation increases the size of the program only by a constant factor, and can be done in linear time by a subprogram **Normalize**. **Normalize** guarantees that the number of ground instances of monadic rules is bounded by $|\mathfrak{A}|$.

Algorithm EvalLIT(Π, \mathfrak{A})

```

Normalize( $\Pi$ )
store  $\mathfrak{A}$  in array  $A$ 
for all strata  $\Pi_i$  from  $\Pi_0$  to  $\Pi_n$ 
  for all rules  $r \in \Pi_i$ 
    let  $G(\bar{y}) :=$  guard of  $r$ 
    for all tuples  $\bar{d} \in G$ 
       $\sigma :=$  unifier of  $\bar{d}$  and  $\bar{y}$ 
       $r' := GL(\{r\sigma\}, A)$ 
      store  $r'$  in HCB  $H_i$ 
    call EvalHORN( $H_i$ )
    and store results in array  $A$ 

```

(2) The Datalog program is evaluated stratum by stratum as a propositional Horn program similarly as sketched by (Berman et al., 1995). Since the ground instances of a guarded rule are determined by the ground substitutions of the variables in the guard, the number of ground instantiations of a guarded rule is bounded by the number of tuples in the guard relation, and thus by $|\mathfrak{A}|$. We conclude that for every rule, the number of ground instances is bounded by $|\mathfrak{A}|$, and obtain the algorithm **EvalLIT** which performs the grounding stratum by stratum. Thus, the algorithm is linear both in the program size and $|\mathfrak{A}|$.

Algorithm EvalLITE. For Datalog LITE, we shall adapt the algorithm to handle generalized literals. Over an input structure \mathfrak{A} , a generalized literal $\varphi(\bar{y}) := (\forall \bar{x} \cdot \alpha \bar{x} \bar{y}) \beta \bar{x} \bar{y}$ can for a fixed $\bar{y} = \bar{d}$ be instantiated by a finite conjunction $\bigwedge_{\bar{c} : \bar{c} \bar{d} \in \alpha^{\mathfrak{A}}} \beta \bar{c} \bar{d}$. If for some \bar{d} , the set $\{\bar{c} : \bar{c} \bar{d} \in \alpha^{\mathfrak{A}}\}$ is empty, the conjunction is equal to true. Thus, we can apply the following strategy: For every generalized literal $\varphi(\bar{y}) = (\forall \bar{x} \cdot \alpha \bar{x} \bar{y}) \beta \bar{x} \bar{y}$, we introduce a new relation symbol $R_\varphi(\bar{y})$, and replace all occurrences of φ by R_φ . In addition, we add ground rules with head predicate R_φ to the program.

Algorithm EvalLITE(Π, \mathfrak{A})

```

store  $\mathfrak{A}$  in array  $A$ 
for all generalized literals  $\varphi(\bar{y}) = (\forall \bar{x} \cdot \alpha \bar{x} \bar{y}) \beta \bar{x} \bar{y}$  in  $\Pi$ ;
  add the rule  $R_\varphi \leftarrow$  to HCB  $H_0$ 
  for all tuples  $\bar{c} \bar{d} \in \alpha^{\mathfrak{A}}$ 
    add  $\beta \bar{c} \bar{d}$  to the body of the rule for  $R_\varphi$ 
for all rules  $r \in \Pi$ 
  replace generalized literals  $\varphi$  in  $r$  by  $R_\varphi$ 
call EvalLIT( $\Pi, \mathfrak{A}$ )

```

Note that **EvalLITE** uses the HCB H_0 of **EvalLIT**. Since the size of $\alpha^{\mathfrak{A}}$ is bounded by $c|\mathfrak{A}|$ (cf. the proof of Theorem 1), the new loop increases the size of the propositional program only linearly. It is easy to see that on a RAM, the program transformation can be done in linear time. (Note that the standard RAM model of (van Emde Boas, 1990) assumes an array to be initialized by zeroes. Alternatively, the initialization can be simulated by the *lazy array* technique as described by (Moret and Shapiro, 1990) with time overhead linear in $|\mathfrak{A}|$.)

According to the most widely used RAM model our algorithms **EvalLIT** and **EvalLITE** use linear space because only those registers (array positions) which are effectively used are taken into account and because initialization is—as mentioned—not necessary. In the case where relation symbols are at most binary, adjacency lists of linear size can be used as data structures. In this case the space complexity is linear even when measured according to less liberal RAM models where the index of the largest used register determines space usage.

6 MODAL DATALOG AND CTLOG

In this section we show that three common temporal verification formalisms, namely propositional (multi-)modal logic ML, CTL, and the alternation-free portion of the μ -calculus can be captured by appropriate fragments of Datalog LITE.

Definition 8 A *modal* Datalog program Π is a Datalog LITE program such that

1. All extensional predicates of Π are unary or binary and all head predicates are unary.
2. All rules have one of the following forms:

$$\begin{aligned} Hx &\leftarrow (\neg)P_1x, \dots, (\neg)P_rx \\ Hx &\leftarrow Exy, (\neg)P_1y, \dots, (\neg)P_ty \\ Hx &\leftarrow (\forall y. Exy)P_0y, (\neg)P_1y, \dots, (\neg)P_ty \end{aligned}$$

Here, (\neg) denotes that negation is optional.

The notion of bisimulation plays a central role in modal logic and model checking. Bisimilarity is usually defined as a kind of two-player game, cf. (Clarke et al., 2000b). For our purposes, it is sufficient to know that two finite structures are bisimilar iff they cannot be distinguished by ML formulas. In fact, it is a characteristic feature of modal logics that they cannot distinguish between bisimilar structures. Thus, the following lemma justifies the terminology *modal* Datalog program.

Lemma 4 Every query defined by a modal Datalog program is invariant under bisimulation.

We say that a modal logic L , like e.g. ML, CTL or the μ -calculus, is equivalent to a class \mathcal{C} of modal Datalog programs if for every Datalog query (Π, H) with $\Pi \in \mathcal{C}$ there exists a sentence $\psi \in L$, and vice versa, such that, for all Kripke structures \mathfrak{K} and all states v of \mathfrak{K} , $\mathfrak{K}, v \models \psi$ iff $v \in (\Pi, H)[\mathfrak{K}]$. Recall that a Datalog program is non-recursive if there are no circular dependencies of its predicates (for a formal definition, see (Abiteboul et al., 1995, Chapter 5.2).)

Definition 9 A CTLog program is a modal Datalog LITE program Π satisfying the following conditions:

1. Every head predicate appears in at most one recursive rule (i.e. a rule where the head predicate occurs also in the body of the rule). Moreover, this rule has the form $Hx \leftarrow Px, E_{\alpha}xy, Hy$ or $Hx \leftarrow Px, (\forall y . E_{\alpha}xy)Hy$.
2. If we remove these rules, the remaining program is non-recursive.

Theorem 6 The following equivalences are valid:

1. non-recursive modal Datalog = propositional multi-modal logic
2. CTLog = CTL on total Kripke structures.
3. modal Datalog = alternation-free μ -calculus

Proof of 1. and 2. (1) Given a non-recursive modal program Π with binary predicates E_a ($a \in A$) we construct for each predicate H of Π a formula ψ_H of propositional multi-modal logic with modalities (actions) $a \in A$, such that ψ_H expresses the query (Π, H) .

By renaming variables, if necessary, we can assume that all rules with head predicate H have head variable x . Further, we can assume that all these rules have the form $Hx \leftarrow (\neg)Px, (\neg)Qx$ or $Hx \leftarrow E_{\alpha}xy, Py$, or $Hx \leftarrow (\forall y . E_{\alpha}xy)Py$ where each $S \in \{P, Q\}$ is either an extensional predicate of Π (in which case $\psi_S := S$) or a head predicate for which the formula ψ_S has already been constructed. For every rule r of the first form let $\varphi_r := (\neg)\psi_P \wedge (\neg)\psi_Q$, for every rule r of the second form, let $\varphi_r := \langle a \rangle \psi_P$, and for every rule of the third form, let $\varphi_r := [a]\psi_Px$. Finally take for ψ_H the disjunction of all the formulae φ_r for all rules with head Hx . It is easy to see that ψ_H is equivalent to (Π, H) .

Conversely, we construct for every sentence $\psi \in \text{ML}$ a non-recursive modal Datalog program Π_ψ with distinguished head predicate H_ψ such that the query (Π_ψ, H_ψ) is equivalent to ψ . If ψ is an atomic proposition P , then Π_ψ consists of the single rule $H_\psi x \leftarrow Px$. If programs for the subformulae of ψ have already been constructed, then extend these programs as follows:

1. $\psi = \varphi \wedge \vartheta$. Add the rule $H_\psi x \leftarrow H_\varphi x, H_\vartheta x$.
2. $\psi = \varphi \vee \vartheta$. Add the rules $H_\psi x \leftarrow H_\varphi x$ and $H_\psi x \leftarrow H_\vartheta x$.
3. $\psi = \neg\varphi$. Add a new stratum consisting of the rule $H_\psi x \leftarrow \neg H_\varphi x$.
4. $\psi = \langle a \rangle \varphi$. Add the rule $H_\psi x = E_{\alpha}xy, H_\varphi y$.
5. $\psi = [a]\varphi$. Add the rule $H_\psi x = (\forall y . E_{\alpha}xy)H_\varphi y$.

(2a) Every CTL-sentence ψ is equivalent to a query (Π_ψ, H_ψ) where Π_ψ is a CTLog-program.

The translation is precisely the same as above except for CTL-sentences of form $\mathbf{E}(\varphi \mathbf{U} \vartheta)$ and $\mathbf{A}(\varphi \mathbf{U} \vartheta)$. (Recall that $\mathbf{EX}\varphi$ and $\mathbf{AX}\varphi$ are equivalent to $\Diamond\varphi$ and $\Box\varphi$, respectively.)

- $\psi := \mathbf{E}(\varphi \mathbf{U} \vartheta)$. Add to the programs Π_φ and Π_ϑ the two rules

$$\begin{aligned} H_\psi x &\leftarrow H_\vartheta x \\ H_\psi x &\leftarrow H_\varphi x, Exy, H_\vartheta y \end{aligned}$$

- $\psi = \mathbf{A}(\varphi \mathbf{U} \vartheta)$. Add to Π_φ and Π_ϑ two strata as follows:

$$\begin{array}{ll} Rx \leftarrow H_\vartheta x & Rx \leftarrow (\forall y . Exy) Ry \\ Sx \leftarrow \neg H_\varphi x, \neg H_\vartheta x & Sx \leftarrow \neg H_\vartheta x, Exy, Sy \\ H_\psi x \leftarrow Rx, \neg Sx & \end{array}$$

The correctness of the translation is obvious for $\mathbf{E}(\varphi \mathbf{U} \vartheta)$. To see that the program for $\psi = \mathbf{A}(\varphi \mathbf{U} \vartheta)$ is correct, note that $\mathbf{A}(\varphi \mathbf{U} \vartheta) \equiv \mathbf{AF} \vartheta \wedge \neg \mathbf{E}(\neg \vartheta \mathbf{U} (\neg \varphi \wedge \neg \vartheta))$. The first stratum computes the set R of states at which $\mathbf{AF} \vartheta$ holds and the set S of states where $\mathbf{E}(\neg \vartheta \mathbf{U} (\neg \varphi \wedge \neg \vartheta))$ is true (see case (5)). The second stratum takes the conjunction of R with the complement of S .

(2b) *Every query computed by a CTLog-program is expressible by a CTL-formula.*

Let Π be a CTLog program and H be a head predicate of Π . There is in Π at most one recursive rule with head Hx and, further, a collection r_1, \dots, r_m of rules with head Hx from the non-recursive portion of Π . The latter can be translated as in part (1) of the proof into sentences $\varphi_{r_1}, \dots, \varphi_{r_m}$ that are built via the basic propositional connectives and the modal operators \Diamond and \Box (or, in CTL-syntax, \mathbf{EX} and \mathbf{AX}) from previously constructed CTL-sentences ψ_P where P are extensional monadic predicates or head predicates from lower strata.

Let $\varphi_H := \varphi_{r_1} \vee \dots \vee \varphi_{r_m}$. If there is no recursive rule with head H , set $\psi_H := \varphi_H$. If the recursive rule with head H has the form $Hx \leftarrow Px, Exy, Hy$ then set $\psi_H := \mathbf{E}(\psi_P \mathbf{U} \varphi_H)$ and if the recursive rule with head H has the form $Hx \leftarrow Px, (\forall y . Exy) Hy$ then set $\psi_H := \mathbf{A}(\psi_P \mathbf{U} \varphi_H)$. It is easily verified that ψ_H is equivalent to (Π, H) . \square

Proposition 2 With respect to data complexity, CTLog is NLOGSPACE-complete. With respect to program complexity, CTLog is PTIME-complete. With respect to data complexity and program complexity, modal Datalog is PTIME-complete.

7 GUARDED FIXED POINT LOGIC

In this section we show that Datalog LITE is equivalent to a natural fragment of the guarded fixed point logic μGF which has recently been studied by (Grädel and Walukiewicz, 1999). The idea to consider *guarded fragments* of first-order logic and its extensions is due to (Andréka et al., 1998). Their main goal was to identify the reasons for the convenient model-theoretic and algorithmic

properties of modal logics and to generalize the modal fragment of first-order logic.

Definition 10 The *guarded fragment* GF of first-order logic is defined inductively as follows:

1. Every relational atomic formula belongs to GF.
2. GF is closed under propositional connectives.
3. If \bar{x}, \bar{y} are tuples of variables, $\alpha(\bar{x}, \bar{y})$ is a positive atomic formula and $\psi(\bar{x}, \bar{y})$ is a formula in GF such that $\text{free}(\psi) \subseteq \text{free}(\alpha) = \{\bar{x}, \bar{y}\}$, then the formulae $\exists \bar{y}(\alpha(\bar{x}, \bar{y}) \wedge \psi(\bar{x}, \bar{y}))$ and $\forall \bar{y}(\alpha(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}, \bar{y}))$ belong to GF.

Here $\text{free}(\psi)$ means the set of free variables of ψ . An atom $\alpha(\bar{x}, \bar{y})$ that relativizes a quantifier as in rule (3) is the *guard* of the quantifier. Notice that the guard must contain *all* the free variables of the formula in the scope of the quantifier.

Notation. We will use the notation $(\exists \bar{y} . \alpha)$ and $(\forall \bar{y} . \alpha)$ for relativized quantifiers, i.e., we write guarded formulae in the form $(\exists \bar{y} . \alpha)\psi(\bar{x}, \bar{y})$ and $(\forall \bar{y} . \alpha)\psi(\bar{x}, \bar{y})$. When this notation is used, then it is always understood that α is indeed a proper guard as specified by condition (3).

The guarded fragment GF extends the modal fragment and turns out to have interesting properties (Andréka et al., 1998; Grädel, 2000): (1) The satisfiability problem for GF is decidable; (2) GF has the finite model property, i.e., every satisfiable formula in the guarded fragment has a finite model; (3) GF has (a generalized variant of) the tree model property; (4) Many important model theoretic properties which hold for first-order logic and modal logic, but not, say, for the bounded-variable fragments FO^k , do hold also for the guarded fragment; (5) The notion of equivalence under guarded formulae can be characterized by a straightforward generalization of bisimulation.

Further, in (Grädel, 2000) it is shown that the satisfiability problem for GF is 2EXPTIME-complete in the general case and EXPTIME-complete in the case where all relation symbols have bounded arity.

The guarded fixed point logic μGF is the extension of GF by least and greatest fixed points, and it relates to GF in the same way as the μ -calculus relates to propositional modal logic and as least fixed point logic $\text{FO}(\text{LFP})$ relates to first-order logic.

Definition 11 The guarded fixed point logic μGF is obtained by adding to GF the following rules for constructing fixed-point formulae: Let ψ be a formula, W a k -ary relation variable that occurs only positively in ψ , and let $\bar{x} = x_1, \dots, x_k$ be a k -tuple of distinct variables. Then we can build the formulae $[\text{LFP } W \bar{x} . \psi](\bar{x})$ and $[\text{GFP } W \bar{x} . \psi](\bar{x})$. A sentence in μGF is *alternation-free* if it does not contain subformulae $\psi := [\text{LFP } T \bar{x} . \varphi](\bar{x})$ and $\vartheta := [\text{GFP } S \bar{y} . \eta](\bar{y})$ such that T occurs in η and ϑ is a subformula φ , or S occurs in φ and ψ is a subformula of η .

The semantics of the fixed point formulae is the usual one, cf. (Ebbinghaus and Flum, 1999). Note that ψ may contain besides \bar{x} and W other free first-order and second-order variables. The fixed-point operator binds W and \bar{x} but leaves all other variables free. It is obvious that μGF generalizes the μ -calculus. On the other side it is not difficult to see that μGF does *not* have the finite model property (see (Grädel and Walukiewicz, 1999) for an example of an infinity axiom in μGF). However, μGF shares most of the other model-theoretic and algorithmic properties of the μ -calculus. In particular, μGF has the generalized tree model property and its satisfiability problem is decidable via automata-theoretic methods. The complexity of μGF could be identified precisely (Grädel and Walukiewicz, 1999).

Theorem 7 (Grädel, Walukiewicz) The satisfiability problem for guarded fixed point logic is 2EXPTIME-complete. For every $k \geq 2$, the satisfiability problem for μGF -sentences with relation symbols of arity at most k is EXPTIME-complete.

Theorem 8 Every alternation-free sentence in μGF is equivalent to a Datalog LITE query. Conversely, every Datalog LITE query is equivalent to an alternation free formula in μGF .

Theorem 9 Recursion-free Datalog LITE has the same expressive power as the guarded fragment of first order logic.

8 CONCLUSION

In summary, Datalog LITE is a robust variant of Datalog with guaranteed linear-time model checking complexity and rich expressive power. This new language facilitates the direct specification of temporal properties of finite (and infinite) state systems by using the paradigm of logic programming. Since Datalog (both with stratified or well-founded negation) is a well-studied database query language for which a number of optimization methods for secondary-storage access have been developed, our results can also be used for applications, where a finite state system is stored in a relational database.

There are several interesting questions related to Datalog LITE which we are currently studying. One is to find suitable extensions of the language to express CTL* and LTL. This can be done by adding very few new primitives along the lines of a general approach to extending Datalog presented in (Eiter et al., 1997). Another interesting issue currently under investigation is the relationship between Datalog LITE and automata, and more general, Datalog and tree automata.

Acknowledgments

This work was supported by the Austrian Science Fund Project N Z29-INF, by Deutsche Forschungsgemeinschaft (DFG), and the Max Kade Foundation. Most of this research has been carried out while the third author was with TU Wien.

We are thankful to Harald Ganzinger, Martin Grohe, Jörg Flum, Thomas Henzinger, Sergey Vorobyov, Jack Minker and an anonymous referee for valuable comments.

References

- Abiteboul, S., Hull, R., and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley.
- Abiteboul, S., Vianu, V., Fordham, B. S., and Yesha, Y. (1998). Relational transducers for electronic commerce. In Paredaens, J., editor, *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 179–187. ACM Press.
- Andréka, H., van Benthem, J., and Németi, I. (1998). Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274.
- Apt, K. R., Blair, H. A., and Walker, A. (1988). Towards a theory of declarative knowledge. In Minker, J., editor, *Foundations of DD and LP*, pages 89–148.
- Berman, K. A., Schlipf, J. S., and Franco, J. V. (1995). Computing well-founded semantics faster. In Marek, V. and Nerode, A., editors, *LPNMR'95*, LNCS, pages 113–126. Springer.
- Bryant, R. E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Transaction on Computers*, pages 35(8):677–691.
- Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., and Hwang, L. J. (1992). Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170.
- Ceri, S., Gottlob, G., and Tanca, L. (1990). *Logic Programming and Databases. Surveys in Computer Science*. Springer.
- Charatonik, W. and Podelski, A. (1998). Set-based analysis of reactive infinite-state systems. In Steffen, B., editor, *TACAS'98*, volume 1384 of *LNCS*. Springer.
- Clarke, E. and Emerson, E. A. (1981). Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs: Workshop*, volume 131 of *LNCS*, pages 52–71. Springer.
- Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. (2000a). Counterexample-guided abstraction refinement. Technical Report CMU-CS-00-103, Computer Science, Carnegie Mellon University. Extended abstract to appear in CAV 2000.
- Clarke, E., Grumberg, O., and Peled, D. (2000b). *Model Checking*. MIT Press.
- Clarke, E. and Schlingloff, H. (2000). Model checking. In Robinson, J. and Voronkov, A., editors, *Handbook of Automated Reasoning*. Elsevier. to appear.
- Clarke, E. M., Grumberg, O., and Long, D. E. (1994). Model checking and abstraction. *ACM Transactions on Programming Languages and System (TOPLAS)*, Vol.16(5):pp.1512 – 1542.
- Cui, B., Dong, Y., Du, X., Kumar, K. N., Ramakrishnan, C. R., Ramakrishnan, I. V., Roychoudhury, A., Smolka, S. A., and Warren, D. S. (1998). Logic programming and model checking. In Palamidessi, C., Glaser, H., and

- Meinke, K., editors. *PLAP/ALP'98*, volume 1490 of *LNCS*, pages 1–20. Springer.
- Dowling, W. F. and Gallier, J. H. (1984). Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming*, 1(3):267–284.
- Ebbinghaus, H.-D. and Flum, J. (1999). *Finite Model Theory (2nd edition)*. Springer.
- Eiter, T., Gottlob, G., and Veith, H. (1997). Modular logic programming and generalized quantifiers. In Dix, J., Furbach, U., and Nerode, A., editors, *LPNMR'97*, volume 1265 of *LNCS*, pages 290–309. Springer.
- Emerson, E. (1990). Temporal and modal logic. In van Leeuwen, J., editor, *Handbook of Theor. Comp. Science. Vol. B.*, pages 995–1072. Elsevier.
- Gelder, A. V., Ross, K. A., and Schlipf, J. S. (1991). The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650.
- Gottlob, G., Grädel, E., and Veith, H. (1998). Datalog LITE: Temporal versus deductive reasoning in verification. Technical Report DBAI-TR-98-22, Institut für Informationssysteme, TU Wien. Available from CoRR at <http://www.acm.org/pubs/corr/>.
- Gottlob, G., Leone, N., and Veith, H. (1999). Succinctness as a source of complexity in logical formalisms. *Annals of Pure and Applied Logic*, 97(1–3):231–260.
- Grädel, E. (2000). On the restraining power of guards. *J. Symb. Logic*. To appear.
- Grädel, E. and Walukiewicz, I. (1999). Guarded fixed point logic. In Longo, G., editor, *Proc. 14th IEEE Symp. on Logic in Computer Science*, pages 45–54.
- Immerman, N. and Vardi, M. Y. (1997). Model checking and transitive-closure logic. In Grumberg, O., editor, *CAV 1997*, volume 1254 of *LNCS*, pages 291–302. Springer.
- Itai, A. and Makowsky, J. A. (1987). Unification as a complexity measure for logic programming. *J. of Logic Programming*, 4(2):105–117.
- Kozen, D. (1983). Results on the propositional μ -calculus. *Theor. Comp. Science*, 27(3):333–354.
- Kurshan, R. P. (1994). *Computer-Aided Verification of Coordinating Processes*. Princeton University Press.
- McMillan, K. L. (1993). *Symbolic Model Checking*. Kluwer.
- Minoux, M. (1988). LTUR: A simplified linear-time unit resolution algorithm for Horn formulae and computer implementation. *Inf. Proc. Let.*, 29(1):1–12.
- Moret, B. and Shapiro, H. (1990). *Algorithms from P to NP*. Benjamin/Cummings.
- Murakami, M. (1990). A declarative semantics of flat guarded Horn clauses for programs with perpetual processes. *Theor. Comp. Science*, 75(1–2):67–83.
- Ramakrishnan, Y. S., Ramakrishnan, C. R., Ramakrishnan, I. V., Smolka, S. A., Swift, T., and Warren, D. S. (1997). Efficient model checking using tabled resolution. In Grumberg, O., editor, *CAV'97*, volume 1254 of *LNCS*, pages 143–154. Springer.
- Ullman, J. D. (1989). *Principles of Data Base Systems*. Computer Science Press.

- van Emde Boas, P. (1990). Machine models and simulations. In van Leeuwen, J., editor, *Handbook of Theor. Comp. Science. Vol. A.*, pages 1–66. Elsevier.
- Vardi, M. Y. (1998). Reasoning about the past with two-way automata. In Larsen, K. G., Skyum, S., and Winskel, G., editors, *ICALP*, volume 1443 of *LNCS*, pages 628–641. Springer.

Chapter 20

ON THE EXPRESSIVE POWER OF PLANNING FORMALISMS

Conditional Effects and Boolean Preconditions in the STRIPS Formalism

Bernhard Nebel

*Institut für Informatik, Albert-Ludwigs-Universität
Freiburg, Germany*
`nebel@informatik.uni-freiburg.de`

Abstract The notion of “expressive power” is often used in the literature on planning. However, it is usually only used in an informal way. In this paper, we will formalize this notion using the “compilability framework” and analyze the expressive power of some variants of STRIPS allowing for conditional effects and arbitrary Boolean formulae in preconditions. One interesting consequence of this analysis is that we are able to confirm a conjecture by Bäckström that preconditions in conjunctive normal form add to the expressive power of propositional STRIPS. Further, we will show that STRIPS with conditional effects is incomparable to STRIPS with Boolean formulae as preconditions. Finally, we show that preconditions in conjunctive normal form do not add any expressive power once we have conditional effects.

Keywords: Action planning, STRIPS, conditional effects, Boolean preconditions, expressiveness, computational complexity

1 INTRODUCTION

The term *expressive power* is often used when planning formalisms are compared.¹ For instance, Pednault (1989) states that the expressiveness of the formalism he proposed is between STRIPS (Fikes and Nilsson, 1971) and the situation calculus (McCarthy and Hayes, 1969), and many people seem to believe that conditional effects increase the expressiveness of STRIPS

¹This paper is a revised and extended version of a paper first presented at ECP-99 (Nebel, 1999b).

significantly (Anderson et al., 1998; Kambhampati et al., 1997; Koehler et al., 1997). Further, Anderson et al. (1998) seem to conjecture that STRIPS with conditional effects is expressively equivalent to STRIPS with conditional effects and Boolean preconditions.² However, there are only a few approaches that try to address the problem of capturing this term formally.

Bäckström (1995) proposed to measure the expressiveness of planning formalisms using his *ESP-reductions*. These reductions are, roughly speaking, polynomial many-one reductions³ on planning instances that *do not change the plan length*. Using this notion, he showed that all of the propositional variants of basic STRIPS not containing conditional effects or arbitrary logical formulae can be considered as expressively equivalent. Furthermore, he conjectured that “disjunctive preconditions most likely increase the expressive power [of STRIPS].”⁴ Bäckström’s approach, however, has two drawbacks. First, it does not seem to be intuitive to define expressiveness using a resource-limited mapping. Second, it does not seem to be possible to prove negative results.

In order to address these problems, we will use the *compilability framework* (Nebel, 1999a; Nebel, 1998), which is inspired by Bäckström’s (1995) approach, by the work of Gazen and Knoblock (1997) to compile conditional effects away, and by the *knowledge compilation* framework (Cadoli and Donini, 1997). The main idea behind this approach is that a language feature does not increase the expressiveness if planning operators containing this language feature can be translated into operators that do not contain this feature without blowing up the operator description too much and without enlarging the resulting plans too much. It differs from Bäckström’s (1995) *ESP-reductions* in that we only consider *structured* transformations that do not translate the initial state or goal specification, in that we allow for arbitrary computational power in the transformation, and in that we do not require that translated plans have *exactly* the same length.

Using this approach, we show that Bäckström’s conjecture that preconditions in conjunctive normal form (CNF) are more expressive than basic STRIPS is indeed correct. We also prove that CNF preconditions do not add to the expressive power if we have already conditional effects – proving a weak version of Anderson’s et al. (1998) conjecture. However, general Boolean formulae as preconditions are incomparable to conditional effects. From that it also follows that Gazen and Knoblock’s (1997) preprocessing scheme for conditional effects is optimal in the sense that we always get a super-polynomial blowup when translating conditional effects to basic STRIPS, provided we require that the plans do not grow more than linearly.

² Actually the statement is that “disjunctive preconditions ... are ... essential prerequisites for handling conditional effects.”

³ We assume that the reader has a basic knowledge of complexity theory (Garey and Johnson, 1979; Papadimitriou, 1994), and is familiar with the notion of *polynomial many-one reductions* and the *complexity classes* P, NP, coNP, and PSPACE. All other notions will be introduced in the paper when needed.

⁴ Bäckström meant *CNF preconditions* when he wrote *disjunctive preconditions*.

Although these results are purely theoretical, they also have some significance for designing planning algorithms. Provided we can prove that a language feature can be “compiled away” easily, i.e., that it can be regarded as “syntactic sugar,” a planning algorithm for the original language can be easily extended to deal with the new feature. Conversely, if we can prove that a language feature cannot be compiled away, we most probably will have significant problems when we want to extend the planning algorithm for the original language to deal with the new language feature.

The rest of the paper is structured as follows. The next section introduces general terminology and definitions. In Section 3 we introduce the notion of compilability between planning formalisms. Using this framework, we show in Section 4 that conditional effects cannot be compiled away. We then analyze the relationship between STRIPS with Boolean preconditions and basic STRIPS in Section 5. We show that DNF preconditions can be compiled away and that Bäckström’s (1995) conjecture holds in the compilability framework. In Section 6 we show that CNF preconditions can be compiled away if we have conditional effects, but that general Boolean preconditions cannot be compiled to conditional effects. Finally, in Section 7 we summarize and discuss the results.

2 PROPOSITIONAL PLANNING FORMALISMS

Let Σ be a finite set of *propositional atoms* and $\hat{\Sigma}$ be the set consisting of the constants \top (denoting truth) and \perp (denoting falsity) as well as atoms and negated atoms, i.e., the *literals*, over Σ . The set of all *Boolean formulae* over Σ is denoted by \mathbf{B}_Σ . The set of *formulae in conjunctive normal form (CNF)* over Σ is denoted by \mathbf{C}_Σ and the set of *formulae in disjunctive normal form (DNF)* over Σ by \mathbf{D}_Σ . Finally, by \mathbf{L}_Σ we refer to the set of *formulae that are conjunctions of literals* over Σ , and the set of *formulae that are conjunctions of atoms* is denoted by \mathbf{A}_Σ . In general, we will use the symbol \mathcal{L}_Σ to refer to a possibly restricted language over Σ .

Given a set of literals $L \subseteq \hat{\Sigma}$, by $pos(L)$ we refer to the *positive literals* in L , by $neg(L)$ we refer to the *negative literals* in L , and $\neg L$ denotes the *element-wise negation* of the literals in L . *Operators* are pairs $o = \langle pre, post \rangle$. We use the notation $pre(o)$ and $post(o)$ to refer to the first and second part of an operator o , respectively. The *precondition* pre is an element of \mathcal{L}_Σ . The set $post$, the set of *postconditions*, consists of *conditional effects* each having the form $\varphi \Rightarrow L$, where $\varphi \in \mathcal{L}_\Sigma$ is called *effect condition* and the elements of $L \subseteq \hat{\Sigma}$ are called *effects*. If all postconditions of an operator have the form $\top \Rightarrow L$, then we say that the operator is *unconditional* and we write the postconditions as a set of literals containing all *effects*.

A state S is a *truth-assignment* for the atoms in Σ , which is represented by the set of atoms that are true in this state. By \perp we represent the *illegal state*. Given a state S and an operator o , we define the *active effects* $A(S, o)$ as follows:

$$A(S, o) = \bigcup \{L \mid (\varphi \Rightarrow L) \in post(o), S \models \varphi\}.$$

Using this function, the result of executing operator o in state S can be specified as:

$$R(S, o) = \begin{cases} S - \neg \text{neg}(A(S, o)) \cup \text{pos}(A(S, o)) & \text{if } S \neq \perp \text{ and} \\ & S \models \text{pre}(o) \text{ and} \\ & A(S, o) \not\models \perp, \\ \perp & \text{otherwise} \end{cases}$$

A *planning instance* is now a tuple $\Pi = \langle \Xi, \mathbf{I}, \mathbf{G} \rangle$, where

- $\Xi = \langle \Sigma, \mathbf{O} \rangle$ is the *domain structure* consisting of a finite set of propositional atoms Σ and a finite set of operators \mathbf{O} ,
- $\mathbf{I} \subseteq \Sigma$ is the *initial state*, and
- $\mathbf{G} \subseteq \hat{\Sigma}$ is the *goal specification*.

When we talk about the *size of an instance*, symbolically $||\Pi||$, in the following, we mean the size of a (reasonable) encoding of the instance.

Let Δ be a finite sequence of operators, which is called *plan*. Then $||\Delta||$ denotes the size of the plan, i.e., the number of operators in Δ . We say that Δ is a *c-step plan* if $||\Delta|| \leq c$. The result of applying Δ to a state S is recursively defined as follows:

$$\begin{aligned} \text{Res}(S, \langle \rangle) &= S, \\ \text{Res}(S, \langle o_1, o_2, \dots, o_n \rangle) &= \text{Res}(R(S, o_1), \langle o_2, \dots, o_n \rangle). \end{aligned}$$

A sequence of operators Δ is said to be a *plan for* Π or a *solution of* Π iff

1. $\text{Res}(\mathbf{I}, \Delta) \neq \perp$ and
2. $\text{Res}(\mathbf{I}, \Delta) \models \mathbf{G}$.

The most general planning language we consider in this paper is STRIPS C_B , which permits conditional effects and general Boolean formulae in preconditions and effect conditions. Without the index C , we refer to planning languages without conditional effects, i.e., all conditional effects have the form $T \Rightarrow L$. If we have C , D or L instead of B , we refer to languages that permit only for CNF or DNF formulae or conjunctions of literals in preconditions and effect conditions, respectively. The language STRIPS (without any index), finally, is identical to basic STRIPS, i.e., it requires that all preconditions are conjunctions of atoms and all effects are unconditional. In this paper, however, we assume that all formulae may contain literals, i.e., the least expressive language we consider is STRIPS $_L$.⁵ In Figure 20.1, the partial order induced by the syntactic restrictions is shown. In the sequel we say that \mathcal{X} is a *specialization* of \mathcal{Y} , written $\mathcal{X} \sqsubseteq \mathcal{Y}$, iff \mathcal{X} is identical to \mathcal{Y} or below \mathcal{Y} in the diagram depicting the partial order.

⁵The reason for leaving out basic STRIPS is that it has already been shown that STRIPS and STRIPS $_L$ as well as STRIPS C and STRIPS C_L are equivalent with respect to expressiveness (Nebel, 1998). Furthermore, ignoring the case $\mathcal{L}_\Sigma = \mathcal{A}_\Sigma$ simplifies some of the technical problems when specifying compilation schemes.

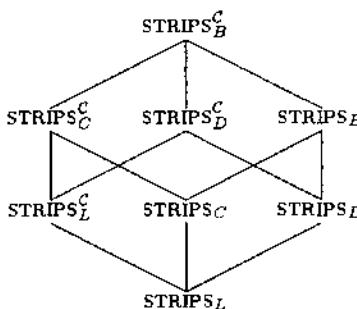


Figure 20.1 Partial order of STRIPS-variants induced by syntactic restrictions

While one would expect that planning in STRIPS_L is much easier than planning in STRIPS_B^C , it turns out that this is not the case, provided one takes a computational complexity perspective. The plan existence problem (PLANEX) is PSPACE-complete for all the formalisms we consider in this paper.

Theorem 1 \mathcal{X} -PLANEX is PSPACE-complete for all \mathcal{X} with $\text{STRIPS}_L \sqsubseteq \mathcal{X} \sqsubseteq \text{STRIPS}_B^C$.

Proof. PSPACE-hardness of STRIPS_L -PLANEX follows from Corollary 3.2 of Bylander's (1994) analysis. Membership of STRIPS_B^C -PLANEX in PSPACE follows because guessing a plan step by step and storing each state can be done in polynomial space. Further, each plan step can be verified in polynomial time. Hence, the problem is in NPSPACE, which is identical to PSPACE. ■

3 COMPIILATION SCHEMES

We will consider a planning formalism \mathcal{X} as expressive as another formalism \mathcal{Y} if planning domains and plans formulated in formalism \mathcal{Y} are concisely expressible in \mathcal{X} . In order to say that something in one formalism \mathcal{Y} can be expressed in another formalism \mathcal{X} , there must exist a *mapping* from \mathcal{Y} -objects to \mathcal{X} -objects that preserves some important properties. The mapping needs not to be computable, though, because expressibility does not mean that there is an easy way to transform the objects from one formalism to another. In particular, if we want to prove negative results, we better show that there exists no mapping whatsoever.

While the computational resources of the mapping are irrelevant, the size of the result is important, provided we want to express something concisely. In the following, we will only consider mappings with results that are polynomially bounded in size.

Finally, we have to make up our mind what this mapping should preserve. Certainly, it should preserve solution existence. Moreover, since we required above that planning domain and plans should be expressed concisely, the mapping should preserve the length of a plan to a certain degree. We could, of course, require more than that. For instance, we may want to require that

there is a function that maps plans in the target formalism back to plans in the source formalism. As we will see, however, such functions can be constructed in all cases we are interested in.

If we now use mappings from \mathcal{Y} planning instances to \mathcal{X} instances that (1) preserve solution existence, (2) do not blow up the plans too much, and (3) have a polynomially sized result, we get the somewhat counter-intuitive result that all the planning formalisms we introduced above have the same expressiveness. The reason is that the mapping can be constructed as follows. Because it is not limited in its computational power, it can test whether there exists a plan for the original \mathcal{Y} instance. If so, it generates an \mathcal{X} instance that has a plan of the same length. Otherwise it constructs an unsolvable \mathcal{X} instance.

This problem could be circumvented by requiring that the computational resources of the mapping are limited, for instance, to polynomial time. In this case, the mapping would be a variation of Bäckström's (1995) ESP-reduction. However, then the question is what the right resource bound is.

Having a closer look at the mapping reveals that it does not measure expressiveness at all but solves the planning instance. However, when we talk about the expressiveness of planning formalisms, we usually mean the expressiveness of the language that is used to specify operators. So the right way to go seems to be to require that the domain structure is transformed independently from the initial state and goal description. And this is, in fact, what we will do. Mappings between domain structures can be viewed as *compiling* the domain structure off-line and enabling us to use the compiled domain structure for different pairs of initial states and goals. For this reason, we will call these mappings *compilation schemata*⁶.

A compilation schema maps a \mathcal{Y} domain structure Ξ into an \mathcal{X} domain structure Ξ' , which is only polynomially larger than Ξ . In addition, the compilation schema may introduce some auxiliary propositional atoms that are used to control the execution of newly introduced operators. These atoms should most likely have an initial value and may appear in the goal specification of planning instances in the target formalism. Finally, it is required that the resulting minimal \mathcal{X} -plan Δ' for Ξ' is bounded in length by the length of the minimal plan Δ for Ξ (see Figure 20.2).

Although Figure 20.2 gives a good picture of the *compilation framework*, it is not completely accurate. If we want to compile a formalism that permits for literals in preconditions and goals to one that requires atoms, some trivial translations of the initial state and goal description are necessary. Similarly, if we want to compile a formalism that permits us to use partial states to a formalism that requires complete state, a translation of the initial state specification is necessary. However, since we do not deal with incomplete state

⁶Note that these compilation schemata are very similar to knowledge compilations, which compile the *fixed part* of a problem in order to speed up the overall processing (Cadoli and Donini, 1997).

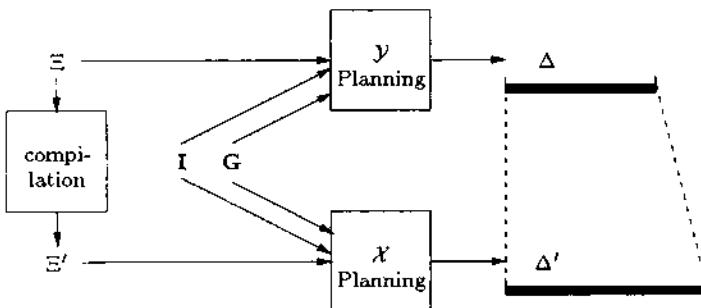


Figure 20.2 The compilation framework

specifications or planning formalisms that allow only atoms in the precondition, we can ignore this issue here.⁷

Let us now define formally what a compilation scheme is. A *compilation scheme from \mathcal{X} to \mathcal{Y}* is a tuple of functions $\mathbf{f} = \langle f_\xi, f_i, f_g \rangle$ that induces a function F from \mathcal{X} -instances $\Pi = \langle \Xi, \mathbf{I}, \mathbf{G} \rangle$ to \mathcal{Y} -instances $F(\Pi)$ as follows:

$$F(\Pi) = \langle f_\xi(\Xi), \mathbf{I} \cup f_i(\Xi), \mathbf{G} \cup f_g(\Xi) \rangle$$

and satisfies the following conditions:

1. there exists a plan for Π iff there exists a plan for $F(\Pi)$,
2. and the size of the results of f_ξ , f_i , and f_g is polynomial in the size of the arguments.

Although there are no resource bounds on f_ξ , f_i , and f_g in the general case, we are also interested in *efficient compilation schemes*. We say that \mathbf{f} is a *polynomial-time compilation scheme* if f_ξ , f_i , and f_g are polynomial-time computable functions.

In addition to that we measure the size of the corresponding plans in the target formalism. If a compilation scheme \mathbf{f} has the property that for every plan Δ solving an instance Π , there exists a plan Δ' solving $F(\Pi)$ such that $\|\Delta'\| \leq \|\Delta\| + k$ for some positive integer constant k , \mathbf{f} is a *compilation scheme preserving plan size exactly* (modulo an additive constant). If $\|\Delta'\| \leq c \times \|\Delta\| + k$ for positive integer constants c and k , then \mathbf{f} is a *compilation scheme preserving plan size linearly*, and if $\|\Delta'\| \leq p(\|\Delta\|, \|\Pi\|)$ for some polynomial p , then \mathbf{f} is a *compilation scheme preserving plan size polynomially*. More generally, we say that a planning formalism \mathcal{X} is *compilable* to formalism \mathcal{Y} (in poly. time, preserving plan size exactly, linearly, or polynomially), if there exists a compilation scheme with the appropriate properties. We write $\mathcal{X} \preceq^x \mathcal{Y}$ in case \mathcal{X} is compilable to \mathcal{Y} or $\mathcal{X} \preceq_p^x \mathcal{Y}$ if the compilation can be done in polynomial time. The super-script x can be 1, c , or p depending on whether the scheme preserves plan size exactly, linearly, or polynomially, respectively.

⁷This means, we do not need the *state-translation functions* as introduced in the definition of a compilation scheme in an earlier paper (Nebel, 1998).

From a practical point of view, one could regard compilability preserving *plan size exactly* or *linearly* as an indication that the planning formalism we use as the target formalism is *at least as expressive* as the source formalism. Conversely, if a *super-linear* (be it polynomial or super-polynomial) blowup of the plans in the target formalism is required by any compilation scheme, this is an indication that the source formalism is *more expressive* than the target formalism – since it indicates that a planning algorithm for the target formalism would be forced to generate significantly longer plans for compiled instances, making it probably infeasible to try to solve such instances. However, compilations preserving plan size polynomially may nevertheless be of practical value (see also Section 7).

As is easy to see, all the notions of compilability introduced above are reflexive and transitive, i.e., compilability induces a pre-order on planning formalisms.

Proposition 1 The relations \preceq^x and \preceq_p^x are transitive and reflexive.

Furthermore, it is obvious that when moving upwards in the diagram displayed in Figure 20.1, there is always a polynomial-time compilation scheme preserving plan size exactly. If π_i denotes the projection to the i -th argument and \emptyset the function that returns always the empty set, the generic compilation scheme for moving upwards in the partial order is $f = (\pi_1, \emptyset, \emptyset)$.

Proposition 2 If $\mathcal{X} \sqsubseteq \mathcal{Y}$, then $\mathcal{X} \preceq_p^1 \mathcal{Y}$.

Now there is, of course, the question whether there are more compilability relationships than those implied by the two propositions. This is the question, we will analyze in the next three sections.

4 COMPIILING CONDITIONAL EFFECTS AWAY

Gazen and Knoblock (1997) proposed a particular way of transforming STRIPS_L^C domains to STRIPS_L domains, which blows up the operator set exponentially. There is, of course, the question whether there are better transformations. As we will show, this is not the case. A simple counting argument shows that with conditional effects there are more solvable initial state/goal pairs than there are when conditional effects are not allowed.

In order to illustrate this point, let us consider an example. We start with a set of n propositional atoms $\Sigma_n = \{p_1, \dots, p_n\}$. Consider now the following STRIPS_L^C domain structure:

$$\begin{aligned} o_n &= \langle T, \{p_i \Rightarrow \neg p_i, \neg p_i \Rightarrow p_i \mid p_i \in \Sigma_n\} \rangle \\ O_n &= \{o_n\}, \\ \Xi_n &= \langle \Sigma_n, O_n \rangle. \end{aligned}$$

This means that the operator o_n “inverts” the truth value of all atoms. From that it is evident that there exist $O(2^n)$ pairs (I, G) such that there exists a one-step plan that solves $\langle \Xi_n, I, G \rangle$.

Trying to find a STRIPS_L^c or STRIPS_B domain structure polynomially sized in $||\Xi_n||$ with the same property seems to be impossible, even if we allow for c -step plans. The reason is that there are only polynomially many different c -step plans.

Theorem 2 $\text{STRIPS}_L^c \not\leq^c \text{STRIPS}_B$.

Proof. Assume for contradiction that there exists a compilation scheme f from STRIPS_L^c to STRIPS_B preserving plan size linearly, which compiles the STRIPS_L^c domain structure Ξ_n defined above into the STRIPS_B domain structure

$$f_\xi(\Xi_n) = \Xi'_n = (\Sigma'_n, \mathbf{O}'_n).$$

Let us now consider all pairs of initial states and goals (\mathbf{I}, \mathbf{G}) such that \mathbf{G} is the “negation” of \mathbf{I} , i.e.,

$$\mathbf{G} = \neg \mathbf{I} \cup (\Sigma - \mathbf{I}).$$

For these pairs, there exist obviously one-step plans. By assumption, the STRIPS_B instance

$$\langle \Xi'_n, \mathbf{I} \cup f_i(\Xi_n), \mathbf{G} \cup f_g(\Xi_n) \rangle$$

should then have a c -step plan. Since there are only $O(|\mathbf{O}'_n|^c)$ different c -step plans, which is a number polynomial in the size of Ξ_n , the same plan Δ is used for different (\mathbf{I}, \mathbf{G}) pairs—provided n is sufficiently large.

Suppose the plan Δ is used for the pairs $(\mathbf{I}'_1, \mathbf{G}'_1), (\mathbf{I}'_2, \mathbf{G}'_2)$, which result from $(\mathbf{I}_1, \mathbf{G}_1)$ and $(\mathbf{I}_2, \mathbf{G}_2)$. Since $\mathbf{I}_1 \neq \mathbf{I}_2$, \mathbf{I}_1 and \mathbf{I}_2 must differ on at least one atom, say p . Without loss of generality we assume $p \in \mathbf{I}_1$ and $p \notin \mathbf{I}_2$. Since Δ is a successful plan from \mathbf{I}'_1 to \mathbf{G}'_1 , it follows that $\text{Res}(\mathbf{I}'_1, \Delta) \models \neg p$. Similarly, we have $\text{Res}(\mathbf{I}'_2, \Delta) \models p$. Since Δ is a plan without conditional effects it always adds and deletes the same atoms. Further, since we assumed $p \in \mathbf{I}_1$ (hence, $p \in \mathbf{I}'_1$), the plan must delete p and not reestablish it. Then, however, it is impossible that we have $\text{Res}(\mathbf{I}'_2, \Delta) \models p$, which is a contradiction.

This means, our initial assumption that there exists a compilation scheme from STRIPS_L^c to STRIPS_B preserving plan size linearly must be wrong. Such a scheme cannot exist. ■

In other words, conditional effects are essential and they cannot be compiled away even if we allow for general Boolean preconditions and a linear increase in plan length.

5 COMPIILING BOOLEAN PRECONDITIONS AWAY

Now the question is whether we can compile Boolean preconditions away. We start with restricted forms of Boolean preconditions and continue to move the partial order displayed in Figure 20.1 upwards.

It is folklore in the planning community that DNF preconditions can be regarded as syntactic sugar. For each operator $o = \langle (c_1 \vee c_2 \vee \dots \vee c_k), L \rangle$, where $c_i \in \mathbf{L}_\Sigma$ and $L \subseteq \widehat{\Sigma}$, one simply generates k new operators $o_i = \langle c_i, L \rangle$. This translation is obviously a *polynomial-time compilation scheme preserving plan size exactly*.

Proposition 3 $\text{STRIPS}_D \preceq_p^1 \text{STRIPS}_L$.

For CNF preconditions the situation is much less clear. As mentioned above, Bäckström (1995) conjectured that CNF preconditions add to the expressiveness of STRIPS_L , but he was not able to prove this conjecture using his framework of ESP-reductions. Using our compilability framework for measuring expressiveness, we can, however, prove his conjecture. In order to do so, we first introduce a variation of the planning problem.

The *fixed plan-size initial-state existence problem (c-FISEX)* is defined as follows. Given a domain structure $\Xi = (\Sigma, \mathbf{O})$, a goal $\mathbf{G} \subseteq \widehat{\Sigma}$, a state $\mathbf{I} \subseteq \Sigma$, and a subset of the atoms called *choice set* $\mathbf{C} \subseteq \Sigma$, the question is whether there exists a set $C \subseteq \mathbf{C}$ such that $\langle \Xi, \mathbf{I} \cup C, \mathbf{G} \rangle$ can be solved by a plan with size c , where c is a positive constant.

Although this problem appears to be slightly harder than the ordinary plan existence problem, fixing the plan length to the positive constant c makes the problem easy, at least for the planning formalism STRIPS_L .

Theorem 3 STRIPS_L -c-FISEX can be decided in polynomial time.

Proof. Given an STRIPS_L -c-FISEX instance $(\Xi, \mathbf{I}, \mathbf{G}, \mathbf{C})$, the number of possible operator sequences is $O(|\mathbf{O}|^c)$, which is polynomial in the instance size. For each such sequence, we can do a regression analysis starting with the goal \mathbf{G} computing the weakest precondition, which is a set of literals. This can be done in polynomial time. Finally, one can easily check in polynomial time, whether there is a subset $C \subseteq \mathbf{C}$ that leads to an initial state $\mathbf{I} \cup C$ that entails the weakest precondition. This means, the problem can be solved in polynomial time for any fixed c . ■

If we allow for CNF preconditions, the problem becomes harder. Even if the plan length is restricted to one, 3SAT can be obviously reduced to the 1-FISEX problem in STRIPS_C .

Proposition 4 STRIPS_C -1-FISEX is NP-complete.

From that it follows immediately that there cannot exist any *polynomial-time compilation scheme* from STRIPS_C to STRIPS_L preserving plan size linearly—provided $P \neq NP$.⁸ We will show a stronger result, namely that there cannot exist *any* compilation scheme preserving plan size linearly by employing a proof technique first used by Kautz and Selman (1992).

⁸Here the difference between Bäckström's (1995) ESP-reductions and compilation schemes should become obvious because the former do not allow us to derive such a conclusion.

In order to prove this result, we need the notion of *advice-taking Turing machines*. These are machines with an *advice oracle*, which is a (not necessarily recursive) function a from positive integers to bit strings. On input I , the machine loads the bit string $a(\|I\|)$ and then continues as usual. Note that the oracle derives its bit string only from the length of the input and not from the contents of the input. An advice is said to be a *polynomial advice* if the oracle string is polynomially bounded by the instance size. Further, if X is a complexity class defined in terms of resource-bounded machines, e.g., P or NP , then $X/poly$ (also called *non-uniform* X) is the class of problems that can be decided on machines with the same resource bounds and polynomial advice.

Because of the advice oracle, the class $P/poly$ appears to be much more powerful than P . However, it seems unlikely that $P/poly$ contains all of NP . In fact, one can prove that $NP \subseteq P/poly$ implies certain relationships between uniform complexity classes that are believed to be very unlikely. In particular, Karp and Lipton (1980) have shown that $NP \subseteq P/poly$ implies that the polynomial hierarchy collapses on the second level—which is considered to be very unlikely.

Theorem 4 $\text{STRIPS}_C \not\leq^c \text{STRIPS}_L$, unless the polynomial hierarchy collapses.

Proof. As a first step, we construct for each n a STRIPS domain structure Ξ_n , a choice set C_n , and a goal specification G_n with size polynomial in n and the following properties. Satisfiability of an arbitrary 3CNF formula φ_n of size n is equivalent to 1-step plan existence for the STRIPS_C -1-FISEX instance $(\Xi_n, G_n, I_{\varphi_n}, C_n)$, where I_{φ_n} can be computed in polynomial time from φ_n .

Given a set of n atoms, denoted by P_n , we define the set of clauses A_n to be the set containing all clauses with three literals that can be built using these atoms. The size of A_n is $O(n^3)$, i.e., polynomial in n . Let D_n be new atoms p_γ corresponding one-to-one to the clauses γ in A_n . Finally, let s be a new atom which is not in $P_n \cup D_n$.

Now we construct a STRIPS_C domain structure $\Xi_n = (\Sigma_n, O_n)$ goal G_n , and the choice set C_n as follows:

$$\begin{aligned}\Sigma_n &= P_n \cup D_n \cup \{s\}, \\ O_n &= \{o_n\}, \\ o_n &= ((\bigwedge_{\gamma \in A_n} (p_\gamma \vee \gamma)), \{s\}), \\ G_n &= \{s\}, \\ C_n &= P_n.\end{aligned}$$

Let $cl(\varphi)$ be the set of clauses appearing in φ . Based on that we define I_{φ_n} as follows:

$$I_{\varphi_n} = \{p_\gamma \in D_n \mid \gamma \notin cl(\varphi_n)\}.$$

Now it is easy to see that φ_n is satisfiable iff for the STRIPS_C -1-FISEX instance $(\Xi_n, G_n, I_{\varphi_n}, C_n)$ there exists a set of choices $C \subseteq C_n$ such that the resulting planning instance $(\Xi_n, I_{\varphi_n} \cup C, G_n)$ is solved by a one-step plan.

Let us now assume that there exists a compilation scheme from STRIPS_C to STRIPS_L preserving plan size linearly. Using this compilation scheme, we compile the STRIPS_C domain structure Ξ_n into the STRIPS_L domain structure $\Xi'_n = \langle \Sigma'_n, \mathbf{O}'_n \rangle$. Further, \mathbf{G}'_n , \mathbf{I}'_{φ_n} , and \mathbf{C}'_n are defined as follows:

$$\begin{aligned}\mathbf{G}'_n &= \{\mathbf{s}\} \cup f_g(\Xi_n), \\ \mathbf{I}'_{\varphi_n} &= \mathbf{I}_{\varphi_n} \cup f_i(\Xi_n), \\ \mathbf{C}'_n &= \mathbf{C}_n.\end{aligned}$$

Note that all these sets can be computed in time polynomial in n , once we know the values of $f_g(\Xi_n)$ and $f_i(\Xi_n)$.

From the construction, it follows that the following statements are equivalent:

1. φ_n is satisfiable,
2. for the STRIPS_L -1-FISEX instance $(\Xi_n, \mathbf{G}_n, \mathbf{I}_{\varphi_n}, \mathbf{C}_n)$, there exists a set of choices $C \subseteq \mathbf{C}_n$ such that $\Pi = \langle \Sigma_n, \mathbf{O}_n, \mathbf{I}_{\varphi_n} \cup C, \mathbf{G}_n \rangle$ has a one-step plan,
3. for the STRIPS_L -c-FISEX instance $(\Xi'_n, \mathbf{G}'_n, \mathbf{I}'_{\varphi_n}, \mathbf{C}'_n)$, there exists a set of choices $C' \subseteq \mathbf{C}'_n$ such that $\Pi' = \langle \Sigma'_n, \mathbf{O}'_n, \mathbf{I}'_{\varphi_n} \cup C', \mathbf{G}'_n \rangle$ has a c-step plan,

One can now construct an advice-taking Turing machine that on input of a formula φ_n of size n loads the polynomial advice $\langle \Xi'_n, f_g(\Xi_n), f_i(\Xi_n) \rangle$ and then decides STRIPS_L -c-FISEX for the instance $(\Xi'_n, \mathbf{G}'_n, \mathbf{I}'_{\varphi_n}, \mathbf{C}'_n)$, which by Theorem 3 can be done in polynomial time. Since the problem 3SAT, which is solved by this deterministic, advice-taking machine in polynomial time is NP-complete, we conclude that $\text{NP} \subseteq \text{P/poly}$. This implies by Karp and Lipton's (1980) result that the polynomial hierarchy collapses on the second level, which proves the claim. ■

The result above implies that adding CNF preconditions to STRIPS_L adds to the expressiveness. However, it is not immediately obvious whether a further generalization from CNF formulae to arbitrary Boolean formulae would add another level of expressiveness. We will defer this question to the next section.

6 COMPILING BOOLEAN PRECONDITIONS INTO CONDITIONAL EFFECTS

As mentioned in Section 1, sometimes the expressive power of conditional effects and of Boolean preconditions are claimed to be related. In this section, we will analyze this claim using the compilability framework.

As in the case of unconditional actions, it is commonly agreed that DNF formulae can be regarded as "syntactic sugar." Any operator containing a DNF precondition with k disjuncts can be split into k new operators containing only conjunctions of literals in the precondition. Similarly, any conditional effect with a DNF effect condition $c_1 \vee \dots \vee c_n \Rightarrow L$ can be equivalently expressed

by a set of n conditional effects $c_i \Rightarrow L$. Obviously, this transformation can be viewed as a polynomial-time compilation scheme preserving plan size exactly.

Proposition 5 $\text{STRIPS}_D^C \preceq_p^1 \text{STRIPS}_L^C$.

Interestingly, CNF preconditions and effect conditions do not appear to add to the expressive power once we have conditional effects—provided we accept that two formalisms have the same expressive power, if they are compilable to each other preserving plan size *linearly*. The main idea behind proving this is that operators with conditional effects can be used to evaluate the truth of clauses.

Theorem 5 $\text{STRIPS}_C^C \preceq_p^c \text{STRIPS}_L^C$.

Proof. Assume that the operators of the STRIPS_C^C domain structure $\Xi = (\Sigma, \mathbf{O})$ contain n clauses $\gamma_1, \gamma_2, \dots, \gamma_n$ with $\gamma_i = l_{i1} \vee \dots \vee l_{ik_i}$ in preconditions and effect conditions. For each clause γ_i , a new atom p_{γ_i} is introduced, and the set of these new atoms is denoted by Γ . Now, the operator *eval*, which will evaluate the truth values of all the clauses in a given state, can be defined as follows:

$$\text{eval} = \langle \top, \{l_{ij} \Rightarrow p_{\gamma_i}\} \rangle.$$

If all clauses γ_i in \mathbf{O} are replaced by the new atoms p_{γ_i} —leading to the new set $\widehat{\mathbf{O}}$ —the only remaining changes that are necessary are that we enforce that the *eval* operator is always executed before an operator from $\widehat{\mathbf{O}}$ is executed and that all operators from \mathbf{O} set all the atoms from Γ to false.

In order to enforce sequences of operators alternating between operators from $\widehat{\mathbf{O}}$ and the *eval*-operator, one can introduce a new atom e that is added to the initial state. In addition, we modify the *eval* operator and all operators $\widehat{o} \in \widehat{\mathbf{O}}$ as follows:

$$\begin{aligned} \text{eval}' &= \langle e, \text{post}(\text{eval}) \cup \{\top \Rightarrow \{\neg e\}\} \rangle \\ o' &= \langle \neg e \wedge \text{pre}(\widehat{o}), \text{post}(\widehat{o}) \cup \{\top \Rightarrow \{e\}\} \cup \{\top \Rightarrow \neg \Gamma\} \rangle. \end{aligned}$$

We can now specify a compilation scheme from STRIPS_C^C to STRIPS_L^C as follows:

$$\begin{aligned} f_E: \langle \Sigma, \mathbf{O} \rangle &\mapsto \langle \Sigma \cup \Gamma \cup \{e\}, \{o' | \widehat{o} \in \widehat{\mathbf{O}}\} \cup \{\text{eval}'\} \rangle, \\ f_I: \langle \Sigma, \mathbf{O} \rangle &\mapsto \{e\}, \\ f_G: \langle \Sigma, \mathbf{O} \rangle &\mapsto \emptyset. \end{aligned}$$

This is obviously a polynomial-time compilation scheme that leads to STRIPS_L^C plans that are twice as long as the original STRIPS_C^C plans. ■

This result appears to be relevant for practical planning algorithms because it suggests how to extend planning algorithms for conditional operators to algorithms for dealing with CNF preconditions and effect conditions. However, one may wonder whether we can improve on this result, coming up with a

compilation scheme preserving plan size exactly. Interestingly, there does not appear to be an obvious way to do that. Further, it is completely unclear how to prove that such a compilation scheme is impossible.

Having shown that CNF preconditions can be compiled away when conditional effects are present, one might hope that this can also be done with general Boolean preconditions. Unfortunately, this does not work, though. In order to show that, we need the notion of *Boolean circuits* and *families of circuits*.

A *Boolean circuit* is a directed, acyclic graph $C = (V, E)$, where the nodes V are called *gates*. Each gate $v \in V$ has a type $\text{type}(v) \in \{\neg, \vee, \wedge, 1, 0\} \cup \{x_1, x_2, \dots\}$. The gates with $\text{type}(v) \in \{1, 0, x_1, x_2, \dots\}$ have in-degree zero, the gates with $\text{type}(v) \in \{\neg\}$ have in-degree one, and the gates with $\text{type}(v) \in \{\wedge, \vee\}$ have in-degree two. All gates except one have at least one outgoing edge. The gate with no outgoing edge is called the *output gate*. The gates with no incoming edges are called the *input gates*. The *depth* of a circuit is the length of the longest path from an input gate to the output gate. The *size* of a circuit is the number of gates in the circuit. Given a *value assignment* to the variables $\{x_1, x_2, \dots\}$, the circuit computes the value of the output gate in the obvious way.

Instead of using circuits for computing Boolean functions, we can also use them for accepting words of length n in $\{0, 1\}^*$. A word $w = x_1 \dots x_n \in \{0, 1\}^n$ is now interpreted as a value assignment to the n input variables x_1, \dots, x_n of a circuit. The word is *accepted* iff the output gate has value 1 for this word. In order to deal with words of different length, we need one circuit for each possible length. A *family of circuits* is an infinite sequence $C = (C_0, C_1, \dots)$, where C_n has n input variables. The language accepted by such a family of circuits is the set of words w such that $C_{\|w\|}$ accepts w .

Usually, one considers so-called *uniform* families of circuits, i.e., circuits that can be generated on a Turing machine with a $\log n$ -space bound. Sometimes, however, also non-uniform families are interesting. For example, the class of languages accepted by non-uniform families of polynomially-sized circuits is just the class P/poly introduced in Section 5.

Using restrictions on the size and depth of the circuits, we can now define new complexity classes, which in their uniform variants are all subsets of P . One class that is important in the following is the class of languages accepted by uniform families of circuits with polynomial size and logarithmic depth, named NC^1 . Another class which proves to be important for us is defined in terms of non-standard circuits, namely circuits with gates that have *unbounded fan-in*. Instead of restricting the in-degree of each gate to be two at maximum, we now allow an unbounded in-degree. The class of languages accepted by families of polynomially sized circuits with unbounded fan-in and constant depth is called AC^0 .

From the definition, it follows almost immediately that $\text{AC}^0 \subseteq \text{NC}^1$. Moreover, it has been shown that there are some languages in NC^1 that are not in the non-uniform variant of AC^0 , which implies that $\text{AC}^0 \neq \text{NC}^1$ (Furst et al., 1984).

In order to prove that we cannot compile Boolean preconditions to conditional effects, we will view families of domain structures with fixed goals and fixed size plans as “machines” that accept languages, similar to families of circuits. For all words w consisting of n bits, let

$$\Xi_n = \langle \Sigma_n \cup \{g\}, \mathbf{O}_n \rangle.$$

Assume that the atoms in Σ_n are numbered from 1 to n . Then a word w consisting of n bits could be encoded by an initial state

$$\mathbf{I}_w = \{p_i \mid \text{iff the } i\text{th bit of } w \text{ is } 1\}.$$

We now say that the n -bit word w is *accepted with a one-step or c-step plan* by Ξ_n iff there exists a one-step or c-step plan, respectively, for the instance

$$\Pi_n = \langle \langle \Sigma_n \cup \{g\}, \mathbf{O}_n \rangle, \mathbf{I}_w \cup \{\neg g\}, \{g\} \rangle.$$

Similarly to families of circuits, we also define families of domain structures, $\Xi = (\Xi_0, \Xi_1, \dots)$. The language accepted by such a family with a one-step (or c-step) plan is the set of words accepted using the domain structure Ξ_n for words of length n . Borrowing the notion of uniformity as well, we say that a family of domain structures is *uniform* if it can be generated by a $\log n$ -space Turing machine.

Papadimitriou (1994) has pointed out that the languages accepted by *uniform polynomially-sized Boolean expressions* is identical to (uniform) NC^1 . As is easy to see, a family of STRIPS_B domain structures is nothing more than a family of Boolean expressions, provided we use one-step plans for acceptance.

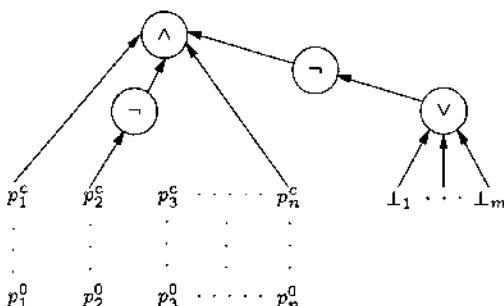
Proposition 6 The class of languages accepted by uniform families of STRIPS_B domain structures using one-step plan acceptance is identical to NC^1 .

If we now have a closer look at what the power of c-step plan acceptance for families of STRIPS_L^C domain structures is, it turns out that it is less powerful than NC^1 . In order to show that, we will first prove the following lemma that relates c-step STRIPS_L^C plans to circuits with gates of unbounded fan-in.

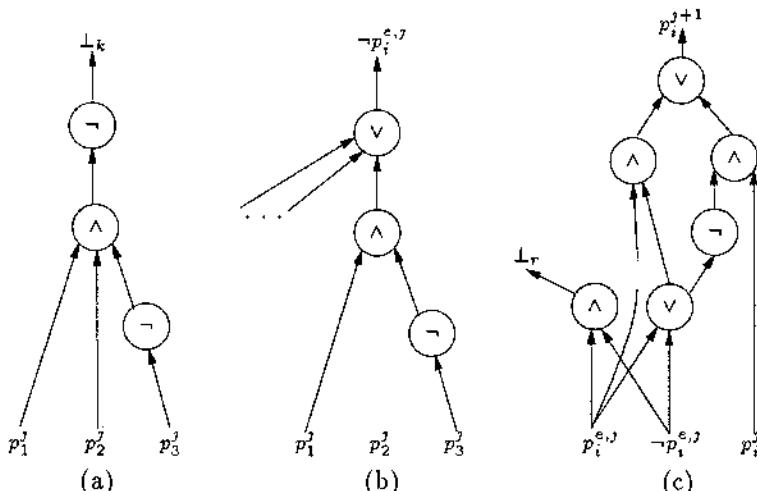
Lemma 1 Let $\Xi = \langle \Sigma, \mathbf{O} \rangle$ be a STRIPS_L^C domain structure, let $\mathbf{G} \subseteq \hat{\Sigma}$, and let Δ be a c-step plan over Ξ . Then there exists a polynomially sized Boolean circuit C with unbounded fan-in and depth $7c + 2$ such that Δ is a plan for $(\Xi, \mathbf{I}, \mathbf{G})$ iff the circuit C has value 1 for the input $w_{\mathbf{I}}$.

Proof. The general structure of a circuit for a c-step STRIPS_L^C plan is displayed in Figure 20.3. For each plan step (or level) j and each atom p_i , there is a connection p_i^j . The connections on level 0 are the input gates, i.e., $p_i^0 = x_i$. The goal test is performed by an \wedge -gate that checks that all the goals are true on level c , in our case $\mathbf{G} = \{p_1, \neg p_2, p_n\}$. Further, using the \vee -gate, it is checked that no inconsistency was generated when executing the plan.

For each plan step j , it must be computed whether the precondition is satisfied and what the result of the conditional effects are. Figure 20.4 (a)

Figure 20.3 Circuit structure and goal testing for a c -step STRIPS_L^C plan

displays the precondition test for the precondition $\{p_1, p_2, \neg p_3\}$. If the conjunction of the precondition literals is not true, \perp_k becomes true, which is connected to the V-gate in Figure 20.3.

Figure 20.4 Circuit structure for precondition testing (a), conditional effects (b), and the computation of effects (c) for STRIPS_L^C operators

Without loss of generality (using a polynomial transformation), we assume that all conditional effects have the form $L \Rightarrow l$. Whether the effect l is activated on level j is computed by a circuit as displayed in Figure 20.4 (b), which shows the circuit for $\{p_1, \neg p_3\} \Rightarrow \neg p_i$.

Finally, all activated effects are combined by the circuit shown in Figure 20.4 (c). For all atoms p_i , we check whether both p_i and $\neg p_i$ have been activated, which would set \perp_r true. This is again one of the inputs of the V-gate in Figure 20.3. If neither p_i nor $\neg p_i$ have been activated, the value of p_i on level $j + 1$ is determined by the value of p_i on level j . Otherwise the value of p_i on level $j + 1$ is determined by the value of $p_i^{e,j}$, i.e., the activation value of the positive effect p_i on level j .

The depths of the circuits in Figure 20.4 (b) and (c) dominate the depth of the circuit necessary to represent one plan step leading to the conclusion that a plan step can be represented using a circuit of depth 7. Adding the depth of the goal testing circuit, the claim follows. ■

The lemma implies that STRIPS_L^C c -step plan acceptance is indeed less powerful than STRIPS_B 1-step plan acceptance, which means that a compilation scheme from STRIPS_B to STRIPS_L^C preserving plan size linearly is impossible.

Theorem 6 $\text{STRIPS}_B \not\leq^c \text{STRIPS}_L^C$.

Proof. Assume for contradiction that $\text{STRIPS}_B \preceq^c \text{STRIPS}_L^C$. Let $\Xi = (\Xi_0, \Xi_1, \dots)$ be a uniform family of STRIPS_B domain structures and $\Xi' = (\Xi'_0, \Xi'_1, \dots)$ be the STRIPS_L^C domain structures generated by a compilation scheme f that preserves plan size linearly. By Lemma 1 we know that for each STRIPS_L^C domain structure $\Xi'_n = (\Sigma'_n, O'_n)$ and given goal G' we can generate a polynomially sized, unbounded fan-in circuit with depth $7c + 2$ that tests whether a particular c -step plan achieves the goal. In order to decide c -step plan existence, we must test $O(|O'_n|^c)$ different plans, which is polynomial in the size of Ξ_n because f is a compilation scheme. For each plan, we can generate one test circuit, and by adding another V -gate we can decide c -step plan existence using a circuit with depth $7c + 3$ and size polynomial in the size of Ξ_n . Since by Proposition 6 all languages in NC^1 are accepted by uniform families of STRIPS_B domain structures using one-step plan acceptance, our assumption $\text{STRIPS}_B \preceq^c \text{STRIPS}_L^C$ implies that we can accept all language in NC^1 by (possibly non-uniform) AC^0 circuits, which is impossible by the result of Furst *et al.* (1984). ■

Using the two results above, can now easily give an answer to the question posed in the end of the previous section, namely, whether general Boolean preconditions are more expressive than CNF preconditions.

Theorem 7 $\text{STRIPS}_B \not\leq^c \text{STRIPS}_C$.

Proof. Assume for contradiction that there is a compilation scheme from STRIPS_B to STRIPS_C preserving plan size linearly. Since by Proposition 2 we have $\text{STRIPS}_C \preceq^c \text{STRIPS}_L^C$ and by Theorem 5 we have $\text{STRIPS}_C^C \preceq^c \text{STRIPS}_L^C$, we can conclude $\text{STRIPS}_B \preceq^c \text{STRIPS}_L^C$ using Proposition 1 twice. This, however, contradicts Theorem 6. ■

This leaves us with the question whether general Boolean preconditions and effect conditions are more expressive than CNF preconditions and effect conditions. However, assuming that $\text{STRIPS}_B^C \preceq^c \text{STRIPS}_C^C$ leads immediately to the conclusion that $\text{STRIPS}_B^C \preceq^c \text{STRIPS}_L^C$ (using Theorem 5 and Proposition 1), which is impossible because of Theorem 6.

Proposition 7 $\text{STRIPS}_B^C \not\leq^c \text{STRIPS}_C^C$.

7 SUMMARY AND DISCUSSION

Using the *compilability framework* (Nebel, 1998), we analyzed the expressive power of disjunctive preconditions and conditional effects. In general, our results provide a complete classification of the relative expressiveness of STRIPS-like languages with restricted formulae and conditional effects – provided that literals are always allowed and states are always complete. Table 20.1 gives an overview of the results (without the trivial STRIPS_B^C column). The “ \sqsubseteq ”

\sqsubseteq^x	STRIPS_L	STRIPS_D	STRIPS_C	STRIPS_B	STRIPS_L^C	STRIPS_D^C	STRIPS_C^C
STRIPS_L	=	\sqsubseteq	\sqsubseteq	\sqsubseteq	\sqsubseteq	\sqsubseteq	\sqsubseteq
STRIPS_D	$\sqsubseteq_p^1(3)$	=	$\sqsubseteq_p^1(3)$	\sqsubseteq	$\sqsubseteq_p^1(3)$	\sqsubseteq	$\sqsubseteq_p^1(3)$
STRIPS_C	$\sqsubseteq^c(4)$	$\sqsubseteq^c(4,3)$	=	\sqsubseteq	$\sqsubseteq_p^c(5)$	$\sqsubseteq_p^c(5)$	\sqsubseteq
STRIPS_B	$\sqsubseteq^c(7)$	$\sqsubseteq^c(7,3)$	$\sqsubseteq^c(7)$	=	$\sqsubseteq^c(6)$	$\sqsubseteq^c(6,5)$	$\sqsubseteq^c(6,5)$
STRIPS_L^C	$\sqsubseteq^c(2)$	$\sqsubseteq^c(2)$	$\sqsubseteq^c(2)$	$\sqsubseteq^c(2)$	=	\sqsubseteq	\sqsubseteq
STRIPS_D^C	$\sqsubseteq^c(2)$	$\sqsubseteq^c(2)$	$\sqsubseteq^c(2)$	$\sqsubseteq^c(2)$	$\sqsubseteq_p^1(5)$	=	$\sqsubseteq_p^1(5)$
STRIPS_C^C	$\sqsubseteq^c(2)$	$\sqsubseteq^c(2)$	$\sqsubseteq^c(2)$	$\sqsubseteq^c(2)$	$\sqsubseteq_p^c(5)$	$\sqsubseteq_p^c(5)$	=
STRIPS_B^C	$\sqsubseteq^c(2)$	$\sqsubseteq^c(2)$	$\sqsubseteq^c(2)$	$\sqsubseteq^c(2)$	$\sqsubseteq^c(6)$	$\sqsubseteq^c(6,5)$	$\sqsubseteq^c(7)$

Table 20.1 Compilability between STRIPS variants

entries mark syntactic specialization relationships (see Figure 20.1). For all other entries we give the strongest compilability result or impossibility result. The number indicates the theorem from which the result has been derived. If the number is in bold face, it is just the statement of the theorem. Otherwise, the result can be derived from the theorem and the application of Propositions 1 and 2. Two particular interesting results are

1. CNF preconditions add to the power of basic STRIPS, confirming an earlier conjecture by Bäckström (1995);
2. Conditional effects cannot be compiled away even if we allow for general Boolean preconditions and a linear growth of the resulting plans. This result shows that we cannot improve on the preprocessing scheme for conditional effects as proposed by Gazen and Knoblock (1997).
3. CNF preconditions and CNF effect conditions do not add anything to the expressive power if we already have conditional effects, confirming a weak version of a conjecture by Anderson *et al.* (1998).

In particular the latter result may have practical value for the design of planning algorithms. It suggests that when normalizing preconditions and effect conditions it is not necessary to convert them to disjunctive normal form, but conjunctive normal form is another option that can be easily dealt with. This option may sometimes help to avoid excessive space consumption, provided the formulae are already almost CNF.

Is this all one can say regarding practical issues? As has been pointed out in Section 3, instead of considering only compilation schemes which preserve plan

size linearly, we might also be interested in compilation schemes that preserve plan size polynomially. What do we get in this case? Interestingly, from this point of view all the formalisms considered in this paper are expressively equivalent (even if we restrict ourselves to polynomial-time compilations) (Nebel, 1998). However, it should be noted that a polynomial blowup of the plan size implies that a planning algorithm has to cope with a potentially much larger search space – which limits the practical value of this result.

Acknowledgments

The research reported in this paper was started and partly carried out while the author enjoyed being a visitor at the AI department of the University of New South Wales. Many thanks go to Norman Foo, Maurice Pagnucco, and Abhaya Nayak and the rest of the AI department for the discussions and cappuccinos.

References

- Anderson, C. R., Smith, D. E., and Weld, D. S. (1998). Conditional effects in Graphplan. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems (AIPS-98)*, pages 44–53. AAAI Press, Menlo Park.
- Bäckström, C. (1995). Expressive equivalence of planning formalisms. *Artificial Intelligence*, 76(1–2):17–34.
- Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204.
- Cadoli, M. and Donini, F. M. (1997). A survey on knowledge compilation. *AI Communications*, 10(3,4):137–150.
- Fikes, R. E. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.
- Furst, M., Saxe, J. B., and Sipser, M. (1984). Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27.
- Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability—A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA.
- Gazen, B. C. and Knoblock, C. (1997). Combining the expressiveness of UCPOP with the efficiency of Graphplan. In Steel and Alami, 1997, pages 221–233.
- Kambhampati, S., Parker, E., and Lambrecht, E. (1997). Understanding and extending Graphplan. In Steel and Alami, 1997, pages 260–272.
- Karp, R. M. and Lipton, R. J. (1980). Some connections between nonuniform and uniform complexity classes. In *Conference Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing (STOC-80)*, pages 302–309, Los Angeles, California.
- Kautz, H. A. and Selman, B. (1992). Forming concepts for fast inference. In *Proceedings of the 10th National Conference of the American Association for Artificial Intelligence (AAAI-92)*, pages 786–793, San Jose, CA. MIT Press.
- Koehler, J., Nebel, B., Hoffmann, J., and Dimopoulos, Y. (1997). Extending planning graphs to an ADL subset. In Steel and Alami, 1997, pages 273–285.

- 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*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, UK. Also published in Webber and Nilsson, 1981.
- Nebel, B. (1998). On the compilability and expressive power of propositional planning formalisms. Technical Report 101, Albert-Ludwigs-Universität, Institut für Informatik, Freiburg, Germany.
- Nebel, B. (1999a). Compilation schemes: A theoretical tool for assessing the expressive power of planning formalisms. In Burgard, W., Cremers, A. B., and Christaller, T., editors, *KI-99: Advances in Artificial Intelligence*, pages 183–194, Bonn, Germany. Springer-Verlag.
- Nebel, B. (1999b). What is the expressive power of disjunctive preconditions? In Biundo, S. and Fox, M., editors, *Recent Advances in AI Planning. 5th European Conference on Planning (ECP'99)*, Durham, UK. Springer-Verlag. To appear.
- Papadimitriou, C. H. (1994). *Computational Complexity*. Addison-Wesley, Reading, MA.
- Pednault, E. P. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In Brachman, R., Levesque, H. J., and Reiter, R., editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 1st International Conference (KR-89)*, pages 324–331, Toronto, ON. Morgan Kaufmann.
- Steel, S. and Alami, R., editors (1997). *Recent Advances in AI Planning. 4th European Conference on Planning (ECP'97)*, volume 1348 of *Lecture Notes in Artificial Intelligence*, Toulouse, France. Springer-Verlag.
- Webber, B. L. and Nilsson, N. J., editors (1981). *Readings in Artificial Intelligence*. Tioga, Palo Alto, CA.

XIV

**KNOWLEDGE BASE SYSTEM
IMPLEMENTATIONS**

Chapter 21

EXTENDING THE SMODELS SYSTEM WITH CARDINALITY AND WEIGHT CONSTRAINTS

Ilkka Niemelä and Patrik Simons

*Helsinki University of Technology, Department of Computer Science and Engineering,
Laboratory for Theoretical Computer Science, P.O.Box 5400, FIN-02015 HUT, Finland
{Ilkka.Niemela,Patrik.Simons}@hut.fi*

Abstract The **Smodels** system is one of the state-of-the-art implementations of stable model computation for normal logic programs. In order to enable more realistic applications, the basic modeling language of normal programs has been extended with new constructs including cardinality and weight constraints and corresponding implementation techniques have been developed. This paper summarizes the extensions that have been included in the system, demonstrates their use, provides basic application methodology, illustrates the current level of performance of the system, and compares it to state-of-the-art satisfiability checkers.

Keywords: Logic programs, stable model semantics, constraints, implementations.

1 INTRODUCTION

Recently logic programs with the stable model semantics have emerged as a novel paradigm for applying declarative logic programming techniques. We call the paradigm *answer set programming*, a term that was coined by Vladimir Lifschitz. It nicely captures the main idea behind the approach: solutions or answers are represented as sets of objects. Furthermore, logic programs with the answer set semantics (Gelfond and Lifschitz, 1990), a generalization of the stable model semantics to handle programs with classical negation, can be seen as an instance of the approach. Answer set programming has been proposed as a methodology for solving, e.g., combinatorial, graph and planning problems (Lifschitz, 1999; Marek and Truszczyński, 1999b; Niemelä, 1999).

We have been developing an implementation for the stable model semantics of normal logic programs called the **Smodels** system (available at <http://www.tcs.hut.fi/Software/smodels/>). When working towards

applications we have observed that (i) the applicability of the system can be widened by extending the underlying language of normal rules with more expressive constructs for representing choices and restrictions on them and (ii) that such extensions can be supported with efficient implementation techniques.

The purpose of this paper is to summarize the extensions that we have included in our system, demonstrate their use, provide basic application methodology, and illustrate the current level of performance of our system.

We start by explaining the basic ideas how the answer set programming approach can be used for applying logic programs with the stable model semantics. More details can be found in (Marek and Truszczyński, 1999b; Niemelä, 1999). In answer set programming a problem is solved by devising a logic program such that the stable models of the program provide the answers to the problem and by using an implementation of the stable model semantics to compute an answer, i.e., a stable model of the resulting program. As an example consider the 3-coloring problem, i.e., the problem of finding an assignment of one of three colors to each vertex of a graph such that vertices connected with an edge do not have the same color. Given a graph we build a program where for each vertex v in the graph we take the three rules on the left and for each edge (v, u) the three rules on the right:

$$\begin{array}{ll} v(1) \leftarrow \text{not } v(2), \text{not } v(3) & \leftarrow v(1), u(1) \\ v(2) \leftarrow \text{not } v(1), \text{not } v(3) & \leftarrow v(2), u(2) \\ v(3) \leftarrow \text{not } v(1), \text{not } v(2) & \leftarrow v(3), u(3) \end{array} \quad (21.1)$$

Stable models are sets of ground atoms and, for instance, the first rule on the left says that if neither of $v(2), v(3)$ is in a stable model, then $v(1)$ is. Thus the rules on the left force each stable model to contain one of $v(1), v(2), v(3)$ for each vertex v in the graph. The first rule on the right excludes stable models containing atoms $v(1)$ and $u(1)$ and similarly for the other two rules. Hence, the rules on the right exclude any stable model having atoms $v(n), u(n)$ for some edge (v, u) in the graph where $n \in \{1, 2, 3\}$. This implies that a stable model of the program gives a legal coloring of the graph where a node v is colored with the color n iff $v(n)$ is included in the stable model.

Normal logic programs provide a framework where a variety of combinatorial, constraint satisfaction, and planning problems can be handled (Niemelä, 1999). However, there are conditions which are hard to capture using normal programs. These are typically related to choices with cardinality, costs and resources. Consider as an example the vertex cover problem where the task is to find a subset of vertices of size less than k such that for each edge (v, u) at least one of the vertices v, u is included in the cover. Given a suitable notion of *cardinality constraints* vertex covers can be captured for a given graph as follows. For each edge (v, u) in the graph a rule

$$1 \{v, u\} \leftarrow \quad (21.2)$$

is included and then an integrity constraint type of a rule

$$\leftarrow k \{v_1, \dots, v_n\} \quad (21.3)$$

is added where $\{v_1, \dots, v_n\}$ is the set of vertices in the graph. The first rule expresses a choice with a cardinality constraint saying that at least one end point for each edge should be selected and the second rule states a cardinality restriction saying that the cover must have size less than k (if the cover contains k or more vertices, it is not acceptable). Such a choice rule (21.2) is not expressible by normal rules without introducing new atoms in the program because for normal rules stable models are subset minimal, i.e., a stable model cannot be a proper subset of another stable model. However, for rules with cardinality constraints this is possible. Furthermore, there seems to be no compact encoding of the cardinality restriction (21.3) using normal rules.

Hence, cardinality constraints appear to be a useful extension to normal logic programs. However, sometimes even more general constraints are needed, e.g., when representing conditions about resources and costs. As an example consider the knapsack problem where we have a set of items with given values and weights and we need to choose a subset such that a given weight limit W is not exceeded but a value limit V is achieved. Such constraints can be captured by generalizing cardinality constraints. The idea is that in a cardinality constraint every atom is treated as having a weight of one and a natural generalization is to allow arbitrary weights. Now the knapsack problem can be captured using the following rules:

$$0 \leq \{i_1 = w_1, \dots, i_n = w_n\} \leq W \leftarrow \quad (21.4)$$

$$\leftarrow \{i_1 = v_1, \dots, i_n = v_n\} \leq V \quad (21.5)$$

The first rule expresses a selection of a subset of the items $\{i_1, \dots, i_n\}$ such that the sum of the weights of the selected items is at most W and the second rule states a constraint eliminating all subsets of the items whose sum of values is not more than V .

Such weight constraints provide an expressive and uniform framework for handling large classes of combinatorial problems. The **Smodels** system takes weight constraints as the basic building blocks of the rules and considers *weight rules* of the form

$$C_0 \leftarrow C_1, \dots, C_n \quad (21.6)$$

where each C_i is a weight constraint of the form

$$\begin{aligned} L \leq & \{a_1 = w_1, \dots, a_n = w_n, \\ & \text{not } a_{n+1} = w_{n+1}, \dots, \text{not } a_m = w_m\} \leq U \end{aligned} \quad (21.7)$$

and every a_i is an atomic formula. This is a generalization of normal logic program rules where each C_i is a literal, i.e., an atomic formula a or its negation $\text{not } a$.

Weight rules have been given a declarative nonmonotonic semantics (Niemelä et al., 1999) that extends the stable model semantics of normal logic programs (Gelfond and Lifschitz, 1988) and generalizes the propositional choice rules presented in (Soininen and Niemelä, 1999). There are a number of approaches where priorities, preferences, costs, probabilities or certainty

factors are associated to rules, see e.g. (Brewka and Eiter, 1998; Marek and Truszczyński, 1999a; Ng and Subrahmanian, 1994; Lu et al., 1996) and the references there. This is different from weight rules where the aim is to provide a relatively simple way of associating weights or costs to atoms and representing constraints using the weights. Approaches such as NP-SPEC (Cadolí et al., 1999), constraint logic programs (CLP) (Jaffar and Lassez, 1987) and constraint satisfaction problems (Tsang, 1993) are not based on stable model semantics and thus do not include, e.g., default negation. In addition, the semantics of weight rules treats the constraints, rules and choices uniformly unlike the CLP and NP-SPEC approaches. There is also some related work based on stable models. For example, in (Buccafurri et al., 1997) priorities are added to integrity constraints. However, this is done to express *weak constraints*, as many of which as possible should be satisfied, and not weight constraints which must all be satisfied. In (Greco, 1999) several types of aggregates are integrated to Datalog in a framework based on stable models in order to express dynamic programming optimization problems using logic programs with stratified negation and choice constructs.

Disjunctive logic programs allow rules which have disjunctions in the heads of the rules and seem to be close to rules with cardinality constraints. However, weight rules are more general than disjunctive rules, since they allow cardinality and weight constraints in both the head and the body of a rule. Moreover, there is a fundamental difference in the semantical basis because the main semantics of disjunctive programs are built on the paradigm of minimal models (Eiter et al., 1997). This means that a model of the program cannot be a proper subset of another model. An exception is the possible model semantics (Sakama and Inoue, 1994) where non-minimal models are allowed. Weight rules may also have non-minimal models and, indeed, rules under the possible model semantics can be seen as a simple subclass of rules with cardinality constraints (Soininen and Niemelä, 1999).

There are a number of implementations of declarative semantics of logic programs already available. For example, the **XSB** system (Warren et al., 1999) is a WAM-based full logic programming system supporting the well-founded semantics and **LDL⁺⁺** (Zaniolo et al., 1999) is a deductive database system based on the stable model semantics supporting stratified programs with choice constructs and user defined aggregates. There are also systems supporting full stable model semantics. The **DeReS** system (Truszczyński et al., 1999) was originally developed as a system for computing extensions of Reiter's default logic but now includes tools tuned for computing stable models of normal logic programs. The **DLV** and **Smodels** systems have been developed specially for computing stable models of logic programs. The main differences are that **DLV** handles disjunctive logic programs and that considerable emphasis has been given to integrating deductive database techniques in **DLV**, e.g., for evaluating efficiently recursive stratified programs. The **Smodels** system does not handle disjunctive rules but supports normal programs and recursive database query evaluation techniques have not been included in **Smodels**. However, **Smodels** has been extended to support both cardinality and weight constraints for rules

with variables and built-in functions. Also constructs for solving optimization problems have been added.

The rest of the paper is organized as follows. We start by discussing the underlying stable model semantics for weight rules and illustrate the basic constructs with a number of examples. Then we present a simple generate and test method that can be used as a starting point for developing applications. The method is demonstrated with examples on propositional satisfiability and Hamiltonian cycles. In Section 4 we briefly explain the techniques that have been developed for implementing weight rules. Then we show the results of some experiments on propositional satisfiability and Hamiltonian cycle problems where the Smodels system is compared with DLV and with state-of-the-art propositional satisfiability checkers.

2 WEIGHT RULES AND THE STABLE MODEL SEMANTICS

In this section we explain the stable model semantics for weight rules

$$C_0 \leftarrow C_1, \dots, C_n$$

originally introduced in (Niemelä et al., 1999) through several steps. First we discuss ground rules, i.e., rules where no variables are allowed. We start with the original stable model semantics for normal rules where each constraint C_i is a literal. Then we move to rules where all constraints are cardinality constraints. This provides a suitable setting for explaining the basic intuition behind extending the stable model semantics. Then we show how to generalize the semantics to weight constraints. We discuss the constructs that have been provided for solving optimization problems and finally present the method for integrating variables and built-in functions used in the Smodels system.

2.1 NORMAL RULES

The stable model semantics (Gelfond and Lifschitz, 1988) generalizes the minimal model semantics of definite programs to normal logic program rules

$$A \leftarrow B_1, \dots, B_m, \text{not } C_1, \dots, \text{not } C_n \tag{21.8}$$

where negative body literals ($\text{not } C_i$) are allowed. For a variable-free (ground) program P , the stable models are defined as follows. The *reduct* P^S of a program P with respect to a set of atoms S is the program obtained from P by deleting

- each rule that has a negative literal $\text{not } C$ in its body with $C \in S$ and then
- all negative literals in the remaining rules.

The reduct P^S can be seen as the set of potentially applicable rules given the stable model S , i.e., as the rules where the negative body literals are satisfied by the model. Note that in the reduct the negative body literals of the potentially

applicable rules are removed and, hence, the rules are definite. The idea is that a stable model should be *justified* in the sense that every atom in the model is a consequence of the potentially applicable rules and every consequence of the potentially applicable rules is included in the model. The atomic consequences of a set of definite rules can be captured as the *deductive closure* of such rules defined in the following way. A set of atoms is *closed* under a set of rules if each rule is satisfied by the atom set, i.e., if the body atoms are in the set, then the head atom is as well. A set of definite rules P has a unique smallest set of atoms closed under P . We call it the *deductive closure* and denote it by $\text{cl}(P)$. The uniqueness is implied by the fact that definite rules are monotonic, i.e., if the body of a rule is satisfied by a model S , then it is satisfied by any superset of S . Note that the closure can be constructed iteratively by starting from the empty set of atoms and iterating over the set of rules and updating the set of atoms with the head of a rule not yet satisfied until no unsatisfied rules are left.

Example 1 Consider a definite program P

$$\begin{array}{ll} p \leftarrow & t \leftarrow r, s \\ q \leftarrow p & s \leftarrow s \\ r \leftarrow p, q & \end{array}$$

Now $\text{cl}(P) = \{p, q, r\}$ which is the least set of atoms closed under the rules. The closure can be constructed in a forward chaining manner by starting from the empty set and including the head of a rule to the set if the body is already contained in the set. In this way $\text{cl}(P)$ is constructed by including first p and then q followed by r .

Definition 1 Let P be a ground program. Then a set of ground atoms S is a *stable model* of P iff $S = \text{cl}(P^S)$.

Example 2 Program P

$$\begin{array}{l} p \leftarrow \text{not } q, r \\ q \leftarrow \text{not } p \\ r \leftarrow \text{not } s \\ s \leftarrow \text{not } p \end{array}$$

has a stable model $S = \{r, p\}$ because $S = \text{cl}(P^S)$ where the reduct P^S has two rules

$$\begin{array}{l} p \leftarrow r \\ r \leftarrow \end{array}$$

In addition to this model, P has another stable model $\{s, q\}$ which can be verified similarly by constructing the reduct and its closure.

2.2 CARDINALITY CONSTRAINTS

Now we consider rules where all constraints C_i are *cardinality constraints* of the form

$$L \{a_1, \dots, a_n, \text{not } b_1, \dots, \text{not } b_m\} U \quad (21.9)$$

where L and U are two integers giving the *lower* and *upper bound* of the constraint, respectively. For a cardinality constraint C , we denote by $\text{lit}(C)$ the corresponding set of literals $\{a_1, \dots, a_n, \text{not } b_1, \dots, \text{not } b_m\}$. The idea is that such a constraint is satisfied by a model for which the cardinality of the subset of the literals satisfied by the model is between the integers L and U . It is allowed to omit either of the bounds in which case a missing lower bound is to be taken as $-\infty$ and an upper bound as ∞ . Hence, we can write a rule such as

$$\{a_1, a_2, a_3\} \leftarrow 1 \{\text{not } b_1, \text{not } b_2\}, 2 \{c_1, c_2, c_3, c_4\} 3$$

which says that if at least one of b_1, b_2 is missing from a stable model and at least 2 but at most 3 are included from $\{c_1, c_2, c_3, c_4\}$, then some subset of $\{a_1, a_2, a_3\}$ is contained in the model. Note that the empty set is also a possible choice for the subset.

Notice also that cardinality constraint rules can be seen as a generalization of normal rules which are a special case with cardinality constraints of the form $1\{l\}1$ where l is a literal. For example, a normal rule $a \leftarrow \text{not } b$ can be seen as a cardinality constraint rule $1\{a\}1 \leftarrow 1\{\text{not } b\}1$.

Our semantics for cardinality constraint rules is a generalization of the stable model semantics for normal logic programs and is given in terms of models that are sets of atoms. First we define when a model satisfies a rule and then using this concept the notion of stable models. Given a model S (a set of atoms) we use the notation $S \models a$ iff $a \in S$ and $S \models \text{not } a$ iff $a \notin S$.

Definition 2 A set of atoms S satisfies a cardinality constraint C of the form (21.9) ($S \models C$) iff $L \leq W(C, S) \leq U$ where

$$W(C, S) = |\{l \in \text{lit}(C) : S \models l\}|$$

is the number of literals in C satisfied by S .

A rule $C_0 \leftarrow C_1, \dots, C_n$ is satisfied by S ($S \models C_0 \leftarrow C_1, \dots, C_n$) iff S satisfies C_0 whenever it satisfies each of C_1, \dots, C_n .

We also allow *integrity constraints*, i.e., rules without the *head* constraint C_0 , which are satisfied if at least one of the *body* constraints C_1, \dots, C_n is not.

Example 3 Consider a rule

$$1 \{a, b\} 1 \leftarrow 1 \{a, \text{not } b, \text{not } c\} 2$$

The constraint in the body is satisfied by the model $\{a, b\}$ because

$$W(1 \{a, \text{not } b, \text{not } c\} 2, \{a, b\}) = |\{a, \text{not } c\}| = 2.$$

However, it is not satisfied by the model $\{a\}$. Hence, the rule is satisfied by $\{a\}$ but it is not satisfied by $\{a, b\}$ because the constraint in the head is not satisfied by $\{a, b\}$. An integrity constraint

$$\leftarrow 1 \{a, b\} 1, 1 \{a, \text{not } b, \text{not } c\} 2$$

is satisfied by models $\{a, b\}$ and $\{a\}$ but not by a model $\{a, c\}$. The latter is the case as both constraints in the body are satisfied by $\{a, c\}$.

A stable model of normal rules is a set of atoms that satisfies the rules and which is justified by the rules in the sense that it coincides with the closure of the reduct for the rules. This justification property is essential in many applications where solutions should not contain any unnecessary (unjustified) elements. When devising semantics for rules with cardinality (and weight) constraints the aim is to preserve this justifiability property by suitably generalizing the concepts of a *reduct* and its *deductive closure* introduced in the previous section for normal programs. For cardinality (and weight) rules, the reduct turns out to be a set of rules in a special form

$$H \leftarrow C_1, \dots, C_n \quad (21.10)$$

where H is a ground atom and each constraint C_i contains only positive literals and has only a lower bound condition. We call such rules *Horn constraint rules*. The deductive closure $\text{cl}(P)$ of a set of Horn constraint rules P can be defined as the unique smallest set of atoms closed under P in the same way as for definite rules. The uniqueness is implied by the fact that Horn constraint rules are monotonic, i.e., if the body of a rule is satisfied by a model S , then it is satisfied by any superset of S . Recall that the closure can be constructed iteratively in a forward chaining manner by starting from the empty set and including the head of a rule to the set if the body is already contained in the set.

Example 4 Consider a set of Horn rules P

$$\begin{aligned} a &\leftarrow 1 \{a\} \\ b &\leftarrow \\ c &\leftarrow 2 \{b, d\}, 1 \{b, a\} . \end{aligned}$$

The deductive closure of P is the set of atoms $\{b\}$ which can be constructed iteratively by starting from the empty set and realizing that the (empty) body of the second rule is satisfied by the empty set and, hence, b should be added to the closure. This set is already closed under the rules. If a rule

$$d \leftarrow 1 \{a, b, c\}$$

is added, then the closure is $\{b, d, c\}$.

Next we introduce the reduct in two steps. First we define the reduct of a constraint and then generalize this to rules. The reduct C^S of a constraint C of the form (21.9) w.r.t. a set of atoms S is the constraint

$$L' \{a_1, \dots, a_n\} \quad (21.11)$$

where $L' = L - |\{\text{not } b \in \text{lit}(C) : S \models \text{not } b\}|$. Hence, in the reduct all negative literals and the upper bound are removed and the lower bound is decreased by the number of negative literals satisfied by S to account for the contribution of the negative literals towards satisfying the lower bound. For example, for a set $S = \{q\}$ and a constraint C

$$3 \{\text{not } q, \text{not } r, p\} 4$$

the reduct C^S is

$$2 \{p\}$$

as only one of the negative literals in the constraint is satisfied by S .

The reduct P^S for a program P w.r.t. a set of atoms S is a set of Horn constraint rules which contains a rule r' with an atom p as the head if $p \in S$ and there is a rule $r \in P$ such that p appears in the head and the upper bounds of the constraints in the body of r are satisfied by S . The body of r' is obtained by taking the reduct of the constraints in the body of r . Formally the reduct is defined as follows.

Definition 3 Let P be a ground program and S a set of ground atoms. The reduct P^S of P w.r.t. S is defined by

$$\begin{aligned} P^S &= \{p \leftarrow C_1^S, \dots, C_n^S : C_0 \leftarrow C_1, \dots, C_n \in P \text{ with } p \in \text{lit}(C_0) \cap S \\ &\quad \text{and for all } i = 1, \dots, n, \text{ for the constraint } C_i \text{ of the form} \\ &\quad L \{a_1, \dots, \text{not } b_m\} U, W(C_i, S) \leq U\} \end{aligned}$$

Now the idea is to follow the case of normal programs and define a stable model of a program P as an atom set S that *satisfies* all rules of P and that is the *deductive closure* of a *reduct* of P w.r.t. S . The role of the reduct is to provide the possible justifications for the atoms in S . Each atom in a stable model is justified by the program P in the sense that it is derivable from the reduct.

Definition 4 A set of ground atoms S is a *stable model* of a program P iff the following two conditions hold:

- (i) $S \models P$,
- (ii) $S = \text{cl}(P^S)$.

Example 5 Consider a program P

$$1 \{a_1, a_2, a_3\} 1 \leftarrow$$

Observe that a stable model of a program P must be a subset of the atoms appearing in the heads of the rules in the program. This is because for any other atom the reduct P^S would not contain a Horn rule with this atom as its head and, hence, the atom could not be in the deductive closure of the reduct

and thus not in a stable model. Hence, a stable model of P must be a subset of $\{a_1, a_2, a_3\}$.

The empty set is not a stable model because $\emptyset \not\models P$ and similarly for every subset having more than one of the atoms. However, $\{a_1\}$ is a stable model of P because it satisfies the rule and the reduct of P , $P^{\{a_1\}} = \{a_1 \leftarrow\}$ has $\{a_1\}$ as its closure. In fact, P has three stable models $\{a_1\}$, $\{a_2\}$, and $\{a_3\}$ as one would expect.

Notice that for

$$1 \{a_1, a_2, a_3\} 1 \leftarrow 1 \{a_1, b\} 2$$

the only stable model is \emptyset . For instance, $\{a_1\}$ is not a stable model because the reduct is $\{a_1 \leftarrow 1 \{a_1, b\}\}$ and its deductive closure is \emptyset .

A program

$$\{a_1, a_2, a_3\} \leftarrow$$

has all eight subsets of $\{a_1, a_2, a_3\}$ as its stable models. This demonstrates that stable models are not necessarily subset minimal.

2.3 WEIGHT CONSTRAINTS

Now we turn to extending the stable model semantics to handle weight constraints. The generalization can be explained by considering the following connection between cardinality constraints and linear inequalities. A cardinality constraint

$$L \{a_1, \dots, a_n, \text{not } b_1, \dots, \text{not } b_m\} U$$

can be seen as a linear inequality

$$L \leq a_1 + \dots + a_n + \overline{b_1} + \dots + \overline{b_m} \leq U$$

where $a_i, \overline{b_j}$ are variables with values 0 or 1 such that $x + \overline{x} = 1$ for every variable x . This can be generalized by allowing an arbitrary coefficient w_v for each variable v leading to linear inequalities of the form

$$\begin{aligned} L \leq & w_{a_1} \times a_1 + \dots + w_{a_n} \times a_n + \\ & w_{b_1} \times \overline{b_1} + \dots + w_{b_m} \times \overline{b_m} \leq U \end{aligned} \quad (21.12)$$

where a coefficient for a variable can be seen as the weight for the corresponding atom. Hence, we are considering constraints of the form

$$\begin{aligned} L \leq & \{a_1 = w_{a_1}, \dots, a_n = w_{a_n}, \\ & \text{not } b_1 = w_{b_1}, \dots, \text{not } b_m = w_{b_m}\} \leq U \end{aligned} \quad (21.13)$$

where, e.g., w_{a_1} is a weight for the atom a_1 . As before we denote by $\text{lit}(C)$ the set of literals in a weight constraint C . The numbers L and U give the *lower* and *upper bounds* of the constraint, respectively. By $w(C)$ we denote the *local weight*

function which gives the weights of the literals in C , i.e., $w(C)(a_i) = w_a$, and $w(C)(\text{not } b_j) = w_b$. The weights and bounds can be arbitrary real numbers. However, our current implementation allows only integers. This restriction has been adopted in order to avoid complications arising from the finite precision of real number arithmetic in standard programming languages.

The stable model semantics presented for cardinality constraints can be generalized to weight constraints in a straightforward way. The basic idea is to generalize the notion of the weight $W(C, S)$ of a constraint C in a model S . For a cardinality constraint C the weight $W(C, S)$ is defined as the number of literals satisfied in S . This corresponds to the case where every literal has weight equal to one. The obvious generalization for the case where arbitrary weights are allowed is to take the sum of the weights of the literals satisfied in S , i.e., for a weight constraint C the weight $W(C, S)$ of C in S is given by

$$W(C, S) = \sum_{l \in \text{lit}(C) \text{ and } S \models l} w(C)(l)$$

Now satisfiability of a weight constraint and a weight rule is defined as for the cardinality constraints in Definition 2.

Example 6 Consider a weight constraint C

$$1.02 \leq \{a = 1.0, b = 0.02, \text{not } c = 0.04\} \leq 1.03$$

for which $W(C, \{a, b, c\}) = 1.0 + 0.02 = 1.02$ and $W(C, \{a\}) = 1.0 + 0.04 = 1.04$. So $\{a, b, c\} \models C$ but $\{a\} \not\models C$.

Given the notion of satisfiability of a rule it is clear what the deductive closure of Horn constraint rules of the form (21.10) is where H is a ground atom and each weight constraint C_i contains only positive literals and has only a lower bound condition.

Example 7 Consider a set of Horn constraint rules P

$$\begin{aligned} a &\leftarrow 1 \leq \{a = 1\} \\ b &\leftarrow 0 \leq \{b = 100\} \\ c &\leftarrow 6 \leq \{b = 5, d = 1\}, 2 \leq \{b = 2, a = 2\} \end{aligned}$$

The deductive closure of P is $\{b\}$. If a rule

$$d \leftarrow 1 \leq \{a = 1, b = 1, c = 1\}$$

is added, then the closure is $\{b, d, c\}$.

When the rules have no negative weights, stable models can be defined using a straightforward generalization of the reduct for cardinality constraints. However, when negative weights are allowed, it is more tricky to guarantee the justifiability of models. This is because negative weights and negative literals are closely related. It turns out that they can replace each other and that one is inessential when the other is available. In order to avoid unnecessary

complications we assume that all weights are non-negative and that negative literals are allowed. First we present the semantics for weight constraint rules in this case. We then show how to transform a rule with negative weights to an equivalent rule with positive weights only.

Suppose weight rules have no negative weights. The notion of a reduct for cardinality constraints is now easy to extend to handle weight constraints: the reduct C^S of a weight constraint C is obtained by removing all negative literals and the upper bound as in (21.11) but with the lower bound defined as

$$L' = L - \sum_{\text{not } p \in \text{lit}(C) \text{ and } S \models \text{not } p} w(C)(\text{not } p)$$

to reflect the fact that arbitrary weights are allowed. A stable model of a set of weight rules with non-negative weights is defined as in Definition 4 using the notion of a reduct given in Definition 3.

Example 8 Consider the program P

$$2 \leq \{b = 2, c = 3\} \leq 4 \leftarrow 2 \leq \{\text{not } a = 2, b = 4\} \leq 5$$

As for cardinality constraints, the justifiability of models guarantees that a stable model of a weight program is a subset of the atoms appearing in the heads of the rules in the program, i.e., a subset of $\{b, c\}$ in this case. The empty set is not a stable model as $\emptyset \not\models P$. The same holds if $S = \{b\}$ because the reduct P^S is empty since the upper bound in the body is exceeded. However, $S = \{c\}$ is a stable model as $S \models P$ and $\text{cl}(P^S) = \{c\}$ where $P^S = \{c \leftarrow 0 \leq \{b = 4\}\}$. In fact, $\{c\}$ is the only stable model of P .

Weight constraints with negative weights can be transformed to constraints with non-negative weights by a simple linear algebraic manipulation which translates a constraint C

$$L \leq \{a_1 = w_{a_1}, \dots, a_n = w_{a_n}, \text{not } b_1 = w_{b_1}, \dots, \text{not } b_m = w_{b_m}\} \leq U$$

to an equivalent form C' with non-negative weights

$$\begin{aligned} L + \sum_{w_{a_i} < 0} |w_{a_i}| + \sum_{w_{b_j} < 0} |w_{b_j}| &\leq \\ \underbrace{\{a_i = w_{a_i}, \dots, a_j = |w_{b_j}|, \dots, \underbrace{\text{not } b_k = w_{b_k}, \dots, \underbrace{\text{not } a_l = |w_{a_l}|, \dots}}_{w_{a_l} < 0}\}}_{w_{a_i} \geq 0} & \end{aligned}$$

$$\leq U + \sum_{w_{a_i} < 0} |w_{a_i}| + \sum_{w_{b_j} < 0} |w_{b_j}|$$

where each negative weight is complemented together with the corresponding literal and the sum of absolute values of all negative weights is added to the bounds.

The equivalence of C and C' can be shown using the linear inequality (21.12) for C . We can eliminate any negative weight w_{a_i} by adding

$$|w_{a_i}| \times (a_i + \bar{a}_i) = |w_{a_i}| \quad (a_i + \bar{a}_i = 1)$$

to the inequality. This leaves the term $|w_{a_l}| \times \bar{a_l}$ in the middle corresponding to the weighted literal not $a_l = |w_{a_l}|$. Similarly, all negative weights w_b , can be eliminated.

Example 9 Consider the rule

$$1 \leq \{a = 2\} \leq 2 \leftarrow -1 \leq \{a = -4, \text{not } b = -1\} \leq 0$$

where we can eliminate the negative weights in the body using the method above. Then the resulting rule is

$$1 \leq \{a = 2\} \leq 2 \leftarrow 4 \leq \{\text{not } a = 4, b = 1\} \leq 5$$

Let $S = \{a\}$. Then the reduct for the resulting rule is $\{a \leftarrow 4 \leq \{b = 1\}\}$ and its deductive closure is \emptyset . Hence, S is not a stable model of the rule. In fact, the rule has no stable models.

Notice that although weight programs extend, e.g., normal logic programs considerably, the computational complexity remains unaffected, i.e., stays in NP. This is due to the fact that given a candidate model S it can be checked in polynomial time whether S is a stable model of a given program P . This holds because (i) the reduct P^S can be constructed in polynomial time and (ii) its deductive closure $\text{cl}(P^S)$ can be computed iteratively in polynomial time.

Theorem 1 (Niemelä et al., 1999) The problem of deciding whether a ground weight program has a stable model is NP-complete.

2.4 OPTIMIZATION CAPABILITIES

For solving optimization problems the Smodels system provides constructs for minimizing or maximizing a function given as a linear sum of weights of literals. For example, a minimization statement

$$\text{minimize } \{a_1 = w_1, \dots, a_n = w_n, \text{not } a_{n+1} = w_{n+1}, \dots, \text{not } a_m = w_m\}$$

specifies that the model given as a solution should be a stable model S of the program with the smallest value for the sum

$$\sum_{l \in L \text{ and } S \models l} w(C)(l)$$

where $L = \{a_1, \dots, a_n, \text{not } a_{n+1}, \dots, \text{not } a_m\}$.

Example 10 Consider the vertex cover problem but with an optimization task, i.e., we would like to find a vertex cover of minimum size. In order to do this we can take the rules (21.2) but replace the rule (21.3) by an optimization statement.

$$\text{minimize } \{v_1 = 1, \dots, v_n = 1\}$$

where $\{v_1, \dots, v_n\}$ is the set of vertices in the graph.

Even multiple minimization/maximization statements are allowed. Here the semantics is that the statements order models lexicographically according to the weights of the statements such that the first statement is the most significant.

Example 11 Consider statements

$$\text{minimize } \{a_1 = 1\}$$

⋮

$$\text{minimize } \{a_n = 1\}$$

which order the models such that the solution is a model not containing a_1 if such a stable model exists and among those a model not containing a_2 if such a model exists and so on.

Notice that the semantics of multiple minimization statements and that of weak constraints in (Buccafurri et al., 1997) are closely related. On the one hand, weak constraints can express lexicographic minimization principles such as that used in the previous example and, on the other hand, weight rules and minimization statements can be used for implementing weak constraints.

2.5 RULES WITH VARIABLES

Until now we have considered only ground rules, i.e., rules where all literals in the constraints are variable-free. In many applications it would be impractical and error-prone to manually write down the required ground rules in order to capture the solutions. Consider, e.g., rules (21.1) for the 3-coloring problem where there are three rules for each node and three rules for each edge of the graph. Usually it is possible to produce the set of ground rules automatically but this requires programming and makes maintaining and revising programs harder.

Hence, it would often be useful to employ rules with variables and functions. The semantics for rules with variables can be obtained by considering the ground instantiation of the program with respect to its Herbrand universe (Niemelä et al., 1999). However, even normal logic programs with the stable model semantics are highly undecidable when function symbols are allowed. Here we present a practical approach employed in the Smodels system to extending weight rules to allow variables and built-in functions in a limited way which guarantees decidability but which seems to be adequate for most practical applications. We start by introducing the basic subclass of rules that is supported by the Smodels system called *domain-restricted rules*. Then we discuss two useful extensions, conditional literals and built-in functions.

In a domain-restricted program P no proper function symbols are allowed. The program can be thought of as being divided into two parts: P_{Defs} that defines *domain predicates* and P_{Cs} that contains all other rules. The rules P_{Defs} for the domain predicates are non-recursive normal logic program rules and all the rules in P are domain-restricted in the sense that every variable in a rule must appear in a domain predicate which appears positively in the body of the rule. Notice that in domain-restricted programs it is possible to

define new domain predicates from previous ones using unions, intersections, complements, joins, and projections of relations. This is demonstrated by the following example.

Example 12 Consider a setting where a set R of ground facts of the form $r(x, y)$ is given. A predicate that appears in the head of a rule only for ground facts qualifies immediately as a domain predicate. Suppose we are to solve a 3-coloring problem for a graph which is defined by the relation r such that there is an edge (x, y) in the graph iff $r(x, y)$ holds in R but $r(y, x)$ does not. The problem can be formalized as a domain-restricted program P which includes the facts R and the rules

$$\text{col}(c_1) \leftarrow; \text{col}(c_2) \leftarrow; \text{col}(c_3) \leftarrow \quad (21.14)$$

$$\text{edge}(X, Y) \leftarrow r(X, Y), \text{not } r(Y, X) \quad (21.15)$$

$$\text{vertex}(X) \leftarrow \text{edge}(X, Y) \quad (21.16)$$

$$\text{vertex}(Y) \leftarrow \text{edge}(X, Y) \quad (21.17)$$

$$1 \{ \text{color}(X, c_1), \text{color}(X, c_2), \text{color}(X, c_3) \} 1 \leftarrow \text{vertex}(X) \quad (21.18)$$

$$\leftarrow \text{edge}(X, Y), \text{color}(X, C), \text{color}(Y, C), \text{col}(C) \quad (21.19)$$

where a term beginning with an upper-case letter, like X, Y , is a variable and other terms, such as c_1, c_2 , and c_3 , are constants. For P , P_{Defs} includes the facts R and the rules (21.14–21.17) and the domain predicates are r , col , edge , and vertex . Observe, e.g., that the last rule would not be domain-restricted if the last body literal $\text{col}(C)$ were removed because then the variable C would only appear in predicate color which is not a domain predicate.

For the Smodels system the subclass of domain-restricted programs has been chosen as a compromise. On the one hand, it enables a simple and efficient two phase implementation technique where in the first (grounding) phase rules with variables are compiled to ground (variable-free) rules which are processed in the second phase by an efficient method for computing stable models for ground programs. On the other hand, non-recursive domain predicates allow a fair amount of modeling power for defining domains of variables.

Next we briefly explain why domain-restricted programs with variables can be compiled efficiently to a set of ground rules which is typically considerably smaller than the Herbrand instantiation of the program but still has exactly the same stable models. Then we introduce two useful extensions of the language, conditional literals and built-in functions. More details on implementation techniques can be found in Section 4. Efficient compilation is possible because of the following properties of domain-restricted programs: (i) The domain rules P_{Defs} have a unique stable model D which can be computed using database techniques as rules for defining domain predicates can be seen as view definitions. (ii) For the model D , it holds that P has the same stable models as P_D where P_D contains those ground instances of rules in P where each ground instance of a domain predicate is in D . (iii) Given D , P_D can be computed efficiently by processing one rule in P at a time.

An obvious extension would be to allow recursive domain predicates, i.e., programs where the part P_{Defs} is stratified. This means that, e.g., the transitive closure tc_r of a domain predicate r could be defined as a domain predicate by the rules

$$\begin{aligned} tc_r(X, Y) &\leftarrow r(X, Y) \\ tc_r(X, Y) &\leftarrow r(X, Z), tc_r(Z, Y) \end{aligned}$$

Notice that tc_r given using these rules could not serve as a further domain predicate for domain-restricted programs because the second rule is recursive. In order to transform the rule to a domain-restricted one, a new domain predicate for the variable Y needs to be added. A straightforward way to define such a domain predicate d is to use a rule $d(Y) \leftarrow r(X, Y)$. Applications where recursive stratified rules are used extensively could benefit if recursive domain predicates where allowed and corresponding techniques for evaluating recursive rules were incorporated in the grounding phase. This has been done, e.g., in the DLV system (Leone et al., 1999). In the Smodels system the focus has not been on such deductive database applications and techniques for handling recursive rules in the grounding phase have not been included.

2.5.1 Conditional Literals. In order to be able to compactly write down sets of literals needed in constraint expressions we have introduced a notion of *conditional literals* of the form

$$l : d \tag{21.20}$$

where l is a literal and the conditional part d is a domain predicate. We require that also rules with conditional literals are *domain-restricted* in the sense that each variable in the rule appears in a domain predicate which is a positive body predicate or the conditional part of some conditional literal in the rule.

When using conditional literals we need to distinguish between *local* and *global* variables in a rule. The idea is that global variables quantify over the whole rule but the scope of a local variable is a single conditional literal. We do not introduce any notation to make the distinction explicit but use the following convention: a variable is local to a conditional literal if it appears only in this literal in the rule and all other variables are global to the rule. For example, in the rule

$$1 \{ \text{color}(X, C) : \text{col}(C) \} 1 \leftarrow \text{vertex}(X) \tag{21.21}$$

the variable C is local to the conditional literal in the head but X is a global variable.

The semantics of domain-restricted rules with conditional literals can be defined by regarding a rule with conditional literals as a shorthand for a set of ordinary ground rules in the following way. Consider a domain-restricted program P with conditional literals and its two parts: a set of non-recursive normal rules P_{Defs} defining the domain predicates and P_{Cs} containing all other rules. We denote by D the unique stable model of P_{Defs} . For a rule r with

conditional literals, the corresponding set of ground rules contains every ground rule that can be obtained by the following two steps. First, all global variables in r are eliminated. This means that every global variable in the rule is substituted by some ground term such that the following condition holds: for each proper domain predicate in the body of the rule (which is not in a conditional literal), the resulting ground instance is included in D . For example, consider the rule (21.21). If $\text{vertex}(v) \in D$, then the rule

$$1 \{ \text{color}(v, C) : \text{col}(C) \} 1 \leftarrow \text{vertex}(v) \quad (21.22)$$

is a possible resulting rule after eliminating the global variable X . In fact, there is a possible resulting rule of the form (21.22) for every ground term v such that $\text{vertex}(v) \in D$.

In the second step we eliminate conditional literals as follows. Because the original rule is domain-restricted, elimination of the global variables leads to rules containing only local variables such that for each conditional literal $l : d$, every variable in it appears also in d . Such a conditional literal is taken as a sequence of ground instances l' of the literal l such that $l' : d'$ is a ground instance of $l : d$ with $d' \in D$. For example, the conditional literal

$$\text{color}(v, C) : \text{col}(C)$$

corresponds to the sequence of literals

$$\text{color}(v, c_1), \text{color}(v, c_2), \text{color}(v, c_3)$$

if D contains the facts $\text{col}(c_1), \text{col}(c_2), \text{col}(c_3)$.

The stable models of a program P with conditional literals are defined to be the stable models of the ground program which consists of the union of the sets of corresponding ground rules for each rule in P .

Example 13 Using conditional literals we can rewrite the rule (21.18) as (21.21) such that the available colors can be changed by giving the corresponding set of facts $\text{col}(c)$. For example, if the facts

$$\text{col}(c_1) \leftarrow; \dots; \text{col}(c_5) \leftarrow$$

for five colors are given, then in the ground instantiation of the coloring program P there is a ground rule

$$1 \{ \text{color}(v, c_1), \dots, \text{color}(v, c_5) \} 1 \leftarrow \text{vertex}(v)$$

for each term v for which $\text{vertex}(v)$ is in the stable model D of P_{Defs} .

2.5.2 Built-in Functions. It is straightforward to extend domain predicates with built-in functions, e.g., for arithmetic. Domain-restrictedness of the rules guarantees that built-in functions can be evaluated together with the domain predicates because for each variable in a built-in function, the domain over which the variable ranges is restricted by domain predicates and

thus floundering problems are avoided. The `Smodels` system includes such an extension where integer arithmetic is available as well as mechanisms for providing user-defined built-in functions. This allows rules such as

$$\begin{aligned} \text{area}(C, W \times L) &\leftarrow \text{width}(C, W), \text{length}(C, L) \\ 0 \leq \{\text{circuit}(C) : \text{area}(C, A) = A\} &\leq 90 \leftarrow \end{aligned}$$

where `area` is a domain predicate defined using domain predicates `width` and `length` (giving the width and length of a circuit) and a built-in function ' \times ' for integer multiplication. The second rule specifies a choice of a subset of the circuits with the sum of areas at most 90. Our implementation also allows expressing weights by rules involving domain predicates as in the example.

3 APPLICATION METHODOLOGY

In this section we present a straightforward application methodology for rules with the stable model semantics which also works surprisingly well in practice. This approach is based on the following idea of dividing the program into two parts: a *generator* part and a *tester* part where the stable models of the generator corresponds to all possible candidate solutions for the problem and the tester then eliminates from the candidates all non-valid ones. Examples of a similar technique for applying disjunctive logic programs can be found, e.g., in (Eiter et al., 1997; Eiter et al., 1998).

Using the choice constructs expressible by cardinality and weight constraints it is straightforward to devise generators. For the tester, integrity constraints provide a powerful and simple-to-use technique to prune unwanted stable models because integrity constraints cannot introduce new stable models but can only eliminate them.

Proposition 1 Let P be a program and IC a set of integrity constraints. Then if S is a stable model of $P \cup IC$, then S is a stable model of P .

Next we demonstrate this basic application methodology using two examples.

3.1 PROPOSITIONAL SATISFIABILITY

Consider first the satisfiability problem for a set S of clauses, i.e., a set of formulae of the form

$$a_1 \vee \cdots \vee a_n \vee \neg b_1 \vee \cdots \vee \neg b_m \tag{21.23}$$

The idea is to construct a program P such that models of S and stable models of P coincide. The generator is now easy to write because potential solutions are truth assignments to the set of atoms $\{a_1, \dots, a_l\}$ appearing in S which correspond to subsets of $\{a_1, \dots, a_l\}$ (giving the atoms assigned true). A program having every subset of $\{a_1, \dots, a_l\}$ as its stable models is easy to write as a single rule

$$\{a_1, \dots, a_l\} \leftarrow$$

or as a rule $\{a_j\} \leftarrow$ for each atom a_j . The tester is also straightforward as it only needs to specify for each clause (21.23) the condition when the clause is not satisfied by a truth assignment. This can be done by an integrity constraint

$$\leftarrow \text{not } a_1, \dots, \text{not } a_n, b_1, \dots, b_m$$

Example 14 Consider the set of clauses S and the corresponding program P :

$$\begin{array}{ll} S : & a \vee \neg b \\ & \neg a \vee b \\ & \{a, b\} \leftarrow \\ P : & \leftarrow \text{not } a, b \\ & \leftarrow a, \text{not } b \\ & \{a, b\} \leftarrow \end{array}$$

P has two stable models $\{\}$ and $\{a, b\}$ which are also the models of S .

As a more involved example of constructing a tester part of a problem we consider satisfiability of general propositional formulae constructed from connectives $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$. Given a propositional formula φ , we construct a program $LP(\varphi)$ such that the models of φ and the stable models of the program correspond. A program $LP(\varphi)$ models the satisfiability of each subformula of φ and it is defined recursively as follows:

- $LP(\neg\varphi) = \{s_{\neg\varphi} \leftarrow \text{not } s_\varphi\} \cup LP(\varphi)$
- $LP(\varphi \wedge \psi) = \{s_{\varphi \wedge \psi} \leftarrow s_\varphi, s_\psi\} \cup LP(\varphi) \cup LP(\psi)$
- $LP(\varphi \vee \psi) = \{s_{\varphi \vee \psi} \leftarrow 1 \{s_\varphi, s_\psi\}\} \cup LP(\varphi) \cup LP(\psi)$
- $LP(\varphi \rightarrow \psi) = \{s_{\varphi \rightarrow \psi} \leftarrow 1 \{\text{not } s_\varphi, s_\psi\}\} \cup LP(\varphi) \cup LP(\psi)$
- $LP(\varphi \leftrightarrow \psi) = \left\{ \begin{array}{l} s_{\varphi \leftrightarrow \psi} \leftarrow s_\varphi, s_\psi \\ s_{\varphi \leftrightarrow \psi} \leftarrow \text{not } s_\varphi, \text{not } s_\psi \end{array} \right\} \cup LP(\varphi) \cup LP(\psi)$
- $LP(\varphi) = \{\{s_\varphi\} \leftarrow\}$ if φ is an atom.

Here all atoms s_χ are new atoms representing the fact that the subformula χ is satisfiable. Now each stable model represents a possible truth assignment (true atoms χ are given by atoms s_χ in the stable model where χ is an atom in the propositional formula φ). In addition, each stable model contains information about the truth values of the subformulae in the truth assignment specified by the stable model. Given this extra information, it is easy to write a tester to exclude models where φ is false: a propositional formula φ is satisfiable iff $\{\leftarrow \text{not } s_\varphi\} \cup LP(\varphi)$ has a stable model.

Example 15 Consider a formula $\varphi = (a \vee b) \leftrightarrow \neg(\neg a \wedge b)$. Now

$$\begin{aligned} \text{LP}(\varphi) : \quad & s_\varphi \leftarrow s_{a \vee b}, s_{\neg(\neg a \wedge b)} \\ & s_\varphi \leftarrow \text{not } s_{a \vee b}, \text{not } s_{\neg(\neg a \wedge b)} \\ & s_{a \vee b} \leftarrow 1\{s_a, s_b\} \\ & s_{\neg(\neg a \wedge b)} \leftarrow \text{not } s_{\neg a \wedge b} \\ & s_{\neg a \wedge b} \leftarrow s_{\neg a}, s_b \\ & s_{\neg a} \leftarrow \text{not } s_a \\ & \{s_a\} \leftarrow \\ & \{s_b\} \leftarrow \end{aligned}$$

3.2 HAMILTONIAN CYCLES

As demonstrated above, rules can capture propositional satisfiability problems. However, there are efficient standard techniques for solving satisfiability problems at least when they are represented as clauses (in conjunctive normal form). Furthermore, a rule representation of a set of clauses does not seem to offer any essential advantages as far as the compactness of the representation is concerned. In this subsection we demonstrate that rules enable succinct representations by discussing another problem which is challenging for propositional logic.

The example that we consider is the Hamiltonian cycle problem for undirected graphs, i.e., the problem of finding a path in a graph that visits each vertex of the graph exactly once and returns to the starting vertex. Cardinality constraint rules enable a straightforward and direct encoding of this problem as shown below. Logic program encodings of the Hamiltonian cycle problem for directed graphs can be found, e.g., in (Lifschitz, 1999; Marek and Truszczyński, 1999b; Niemelä, 1999). This problem seems to be challenging for propositional logic both conceptually and computationally. We illustrate this in Section 5 where an encoding of Hamiltonian cycles in propositional logic is given and compared experimentally to logic program encodings.

For our logic program encoding, we assume that the graph (V, E) is represented as a set of facts including $\text{vertex}(v)$ for each vertex $v \in V$ of the graph, and $\text{edge}(v, u)$ for every edge $\{u, v\} \in E$ in the graph. For simplicity suppose that only one of the facts $\text{edge}(v, u)$ and $\text{edge}(u, v)$ is included for an edge $\{u, v\} \in E$.

A program for Hamiltonian cycles can be constructed using the following ideas. A Hamiltonian cycle is represented by atoms of the form $hc(v, u)$ so that stable models and Hamiltonian cycles correspond. A fact $hc(v, u)$ in a stable model of the program says that the edge $\{u, v\}$ belongs to the corresponding Hamiltonian cycle. The generator part of the program selects edges in such a way that for each vertex v exactly two edges $\{v, u\}$ and $\{w, v\}$ (two facts $hc(v, u)$ and $hc(w, v)$) are present in any stable model. Hence, models represent cycles but there could be more than one separate cycle. To eliminate such models we devise a tester using two additional concepts: we take some arbitrary vertex w as the initial vertex (the fact $\text{initialvertex}(w)$ is added) and define a predicate

$r(v)$ to capture the fact that vertex v is reachable through the chosen edges from the initial vertex. Now the tester says simply that all vertices should be reachable from the initial vertex. These considerations lead to the following program

```

2 { $hc(V, U) : edge(V, U), hc(W, V) : edge(W, V)$ }  $2 \leftarrow vertex(V)$ 
 $r(V) \leftarrow initialvertex(V)$ 
 $r(V) \leftarrow hc(V, U), edge(V, U), r(U)$ 
 $r(V) \leftarrow hc(U, V), edge(U, V), r(U)$ 
 $\leftarrow vertex(V), \text{not } r(V)$ 

```

where the first rule provides the generator and the rest of the rules define the tester. Notice that conditional literals are used as a compact method to represent the set of literals needed in the generator rule and that the domain predicate *edge* is used to guarantee domain-restrictedness in the rules defining r .

4 IMPLEMENTATION TECHNIQUES

The **Smodels** system implements the stable model semantics for domain-restricted rules where cardinality and weight constraints as well as variables and built-in functions can be used. The basic function of **Smodels** is to compute stable models for a given program. This is done in two phases where the first phase, implemented in the program **1parse** (Syrijänen, 1999), compiles the logic program into a set of primitive ground rules for which in the second phase stable models are computed using a Davis-Putnam like procedure (Simons, 1999a), implemented in the program **smodels** (Simons, 1999b). Hence, **1parse** functions as a front-end for **smodels** but any program that creates primitive rules and wants to compute stable models can also use **smodels** as a library. Below we describe the two phases briefly. Further details can be found in (Simons, 1999a; Niemelä et al., 1999).

First Phase The first phase is implemented in two steps. During the first step the program is parsed and domain predicates are automatically detected. Then domain predicates are evaluated using database techniques by building a stratification of the rules defining domain predicates and evaluating them one stratum at a time starting from the lowest strata. For a program P this gives the unique stable model D of the part P_{Defs} for domain predicate rules. After this the ground instances P_D of the rules can be computed one rule at a time by essentially evaluating relational joins of domain predicates because the program is domain-restricted. During this process also conditional literals are extended to corresponding sequences of ordinary literals and built-in functions are evaluated. The result of the first step is a set of ground rules without built-in functions or conditional literals.

In the second step of the first phase the ground rules are compiled into four types of ground primitive rule forms: basic, choice, constraint, and weight rules. These particular primitive forms were chosen as they can be easily and

efficiently implemented, and they are the rules that are the input to the second phase, i.e., to the `smodels` procedure.

The basic rule is a normal logic program rule of the form

$$h \leftarrow a_1, \dots, a_n, \text{not } b_1, \dots, \text{not } b_m.$$

The choice rule

$$\{h_1, \dots, h_k\} \leftarrow a_1, \dots, a_n, \text{not } b_1, \dots, \text{not } b_m$$

is a limited form of a cardinality constraint rule and the primitive constraint rule

$$h \leftarrow k \{a_1, \dots, a_n, \text{not } b_1, \dots, \text{not } b_m\}$$

is another. Finally, there is the primitive weight rule

$$h \leftarrow \{a_1 = w_{a_1}, \dots, a_n = w_{a_n}, \text{not } b_1 = w_{b_1}, \dots, \text{not } b_m = w_{b_m}\} \geq w.$$

We restrict ourselves to positive integer weights. A rule with negative weights can be translated into a rule with only positive weights as described in Section 2.3.

We will illustrate the translation from constraint and weight rules into primitive rules by way of an example. A complete description of the translation can be found in (Niemelä et al., 1999).

Example 16 We want to translate the rule

$$2 \{h_1, \dots, h_k\} 4 \leftarrow 1 \{a_1, \dots, a_n\} 3, 2 \leq \{b_1 = w_1, \dots, b_m = w_m\} \leq 4$$

into primitive rules. We begin by creating two rules per constraint encoding whether the lower bound is satisfied and whether the upper bound is not, respectively. The constraint $2 \{h_1, \dots, h_k\} 4$ is translated into the rules

$$\begin{aligned} t_1 &\leftarrow 2 \{h_1, \dots, h_k\} \\ t_2 &\leftarrow 5 \{h_1, \dots, h_k\} \end{aligned}$$

where t_1 and t_2 are some new atoms. Similarly, the second constraint $1 \{a_1, \dots, a_n\} 3$ is translated into the rules

$$\begin{aligned} t_3 &\leftarrow 1 \{a_1, \dots, a_n\} \\ t_4 &\leftarrow 4 \{a_1, \dots, a_n\} \end{aligned}$$

and the weight constraint $2 \leq \{b_1 = w_1, \dots, b_m = w_m\} \leq 4$ is translated into

$$\begin{aligned} t_5 &\leftarrow \{b_1 = w_1, \dots, b_m = w_m\} \geq 2 \\ t_6 &\leftarrow \{b_1 = w_1, \dots, b_m = w_m\} \geq 5. \end{aligned}$$

This lets us capture the truth value of the body of the weight rule with the rule

$$t_7 \leftarrow t_3, \text{not } t_4, t_5, \text{not } t_6.$$

Hence, we can encode the original rule as

$$\begin{aligned} \{h_1, \dots, h_k\} &\leftarrow t_7 \\ &\leftarrow t_7, \text{not } t_1 \\ &\leftarrow t_7, t_2 . \end{aligned}$$

It is of course important that the atoms t_1, \dots, t_7 are not used anywhere else in the program.

Second Phase In the second phase a Davis-Putnam like procedure computes stable models of a set of primitive rules. This procedure is a backtracking search procedure that finds the stable models of a program by assigning truth values to the atoms of the program. Moreover, it uses the properties of the stable model semantics to infer and propagate additional truth values. The propagation is based on the well-founded semantics of normal programs but extends it, e.g., by handling the new primitive rules, i.e., primitive choice, constraint, and weight rules, and by back-propagation and lookahead techniques. The procedure employs a dynamic application-independent search heuristic where the idea is to assign truth values to the atoms in an order attempting to minimize the remaining search space. More details on the procedure can be found in (Simons, 1999a).

Since the procedure is in effect traversing a binary search tree, the number of nodes in the search space is in the worst case on the order of 2^n , where n can be taken to be the number of atoms that appear in a constraint in a head of a rule or that appear as a negative literal in a recursive loop of the program. This means, for example, that if we want to find a satisfying assignment of a general propositional formula, then the search space of the encoding given in Section 3.1 is restricted to the atoms of the original formula.

5 EXPERIMENTS

In this section we demonstrate the current level of performance of the **Smodels** system. We compare it with the **DLV** system, which is another advanced implementation of the stable model semantics, and with state-of-the-art propositional satisfiability checkers: two complete checkers **sato** (Zhang, 1997) and **satz** (Li and Anbulagan, 1997) together with **walksat** (Selman et al., 1994) based on local search. These satisfiability checkers provide interesting reference systems because they have been employed as basic inference engines in advanced reasoning systems, e.g., **sato** has been used in **CCALC** (McCain, 1999) and the other two checkers in the planning system **BlackBox** (Kautz and Selman, 1999).

For the tests we use random 3-SAT problems and Hamiltonian cycle problems. The 3-SAT problems are chosen such that the clause to atom ratio is $4.258 + 58.26a^{-5/3}$, where a is the number of atoms, as this particular ratio determines a region of hard satisfiability problems (Crawford and Auton, 1996). For the Hamiltonian cycle problem we use planar graphs created by the **plane** function found in the Stanford GraphBase (Knuth, 1993) which builds a

```

 $inCycle(V, U) \vee outCycle(V, U) \leftarrow edge(V, U)$ 
 $r(V) \leftarrow initialvertex(V)$ 
 $r(V) \leftarrow edge(U, V), r(U), inCycle(U, V)$ 
 $r(V) \leftarrow edge(V, U), r(U), inCycle(V, U)$ 
 $\leftarrow vertex(V), \text{not } r(V)$ 
 $n(V, U) \leftarrow edge(V, U), inCycle(V, U)$ 
 $n(V, U) \leftarrow edge(U, V), inCycle(U, V)$ 
 $\leftarrow vertex(V), arc(V, U_1), arc(V, U_2), arc(V, U_3),$ 
 $n(V, U_1), n(V, U_2), n(V, U_3), U_1 \neq U_2, U_1 \neq U_3, U_2 \neq U_3$ 
 $twoNs(V) \leftarrow vertex(V), arc(V, U_1), arc(V, U_2),$ 
 $n(V, U_1), n(V, U_2), U_1 \neq U_2$ 
 $\leftarrow vertex(V), \text{not } twoNs(V)$ 

```

Figure 21.1 Encoding of the Hamiltonian cycle problem for DLV

planar graph by inserting a given number of random points on a plane and by constructing the Delaunay triangulation of the points.

The 3-SAT problems are encoded as logic programs using the method in Section 3.1. For **DLV** the generator part includes for each atom a in the set of clauses a rule $a \vee a' \leftarrow$ where a' is a new atom not used anywhere else in the program.

The Hamiltonian cycle problems are encoded for **Smodels** as described in Section 3.2 using cardinality constraints. Since **DLV** does not support cardinality constraints, we have modified the encoding and used the translation given in Figure 21.1. Here the idea is that a Hamiltonian cycle is represented by facts $inCycle(V, U)$. The first five rules guarantee that in a stable model every vertex is reachable from the initial vertex through the chosen edges $inCycle(V, U)$ in the model. The next three rules exclude stable models where a vertex has three neighbors in the cycle. Here we use the predicate $arc(V, U)$, which is the symmetric closure of $edge(V, U)$, in order to achieve a compact encoding. The last two rules exclude models where there is a vertex which does not have two neighbors.

It is challenging to devise a compact encoding of Hamiltonian cycles in propositional logic. We translate a Hamiltonian cycle problem into a set of clauses following (Papadimitriou, 1995) using a quadratic number of atoms w.r.t. the number of vertices in the graph. The translation encodes a total order on the vertices of the graph in such a way that there is an edge between any pair of vertices that are adjacent in the order. Given the vertices v_1, \dots, v_n ,

we use the atom $v_{i,j}$ to encode that vertex v_i is in position j using the clauses:

$$\begin{array}{ll} v_{i,1} \vee \dots \vee v_{i,n} & v_i \text{ is in some position,} \\ \neg v_{i,j} \vee \neg v_{i,k} & j \neq k \quad v_i \text{ is in at most one position,} \\ v_{1,j} \vee \dots \vee v_{n,j} & \text{some vertex is in position } j, \text{ and} \\ \neg v_{i,k} \vee \neg v_{j,k} & i \neq j \quad v_i \text{ and } v_j \text{ are not in the same position.} \end{array}$$

Then we deny orders that do not correspond to Hamiltonian cycles:

$$\neg v_{i,k} \vee \neg v_{j,k+1 \bmod n} \quad \{v_i, v_j\} \text{ is not an edge in the graph.}$$

For each problem size we generate ten sets of rules/clauses. Each set is randomly shuffled and run ten times. We shuffle the input since a particular ordering might help the algorithms to avoid backtracking, thereby giving a skewed picture of their behavior. That is, we try to lessen the impact of lucky choices. The durations of the tests are given in seconds and they represent the time to find a solution or the time to decide that there are no solutions. All tests were run under Linux 2.2.12 on 450 MHz Pentium III computers with 256 MB of memory.

We tested each system using standard settings, i.e., without any tuning, using the following versions: **Smodels** with **smodels** 2.25 and **lparse** 0.99.45, **DLV** release 1999-11-24, **sato** 3.2, **satz41**, and **walksat** 35. For **walksat** which is based on local search, some parameters must be chosen and we have used the settings suggested in (Kautz and Selman, 1999): '-best -noise 50 100 -cutoff 100000000'.

The results of the 3-SAT tests are shown in Figure 21.2 giving the maximum, average and minimum running times. We did not test **walksat** on 3-SAT problems because **walksat** is an incomplete procedure and not all 3-SAT test cases had a solution. However, all Hamiltonian cycle test cases had a solution and in this case **walksat** could be used. The results of the Hamiltonian cycles tests are given in Figure 21.3. Here the running times do not include the preprocessing time to translate a graph to a set of propositional clauses or ground rules. The clausal translation was done using a special purpose program but for the rule translation the **lparse** program was used. This means that the execution time for both **DLV** and **Smodels** was measured when given a ground program (produced by **lparse**) as input. In order to study the efficiency of the cardinality constraints, the **Smodels** system was run also using an encoding without cardinality constraints which is obtained from the **DLV** encoding (Figure 21.1) by replacing the first rule by

$$\{inCycle(V, U)\} \leftarrow edge(V, U).$$

The results for this encoding are denoted by 'smodeles-n' in Figure 21.3.

The **Smodels** system scales in the same way as the satisfiability checkers on the random 3-SAT problems, but it is on average forty to fifty times slower than the fastest checker. When it comes to the Hamiltonian cycle problems the situation is reversed. The **Smodels** system scales better and is much faster

than the satisfiability checkers. The difference can be attributed to the more compact logic program encoding of the Hamiltonian cycle problem and to the search heuristic used in the `smodels` procedure.

6 CONCLUSIONS

The `Smodels` system has been extended with cardinality and weight constraints in order to enable more realistic applications. Such constructs are important, e.g., in product configuration, resource-bounded planning, and constraint satisfaction to represent different kinds of cardinality, cost and resource constraints. They often enable compact, natural and easy to maintain representations of involved problems. The extension has been done in such a way that basic computational complexity has not been affected, i.e., deciding whether a ground weight program has a stable model remains NP-complete as for normal programs. However, corresponding reasoning tasks for closely related semantics of disjunctive programs are typically complete problems on the second level of the polynomial hierarchy (Eiter and Gottlob, 1995).

We have also developed a two phase implementation technique for the new constraints. In the first phase, rules with variables and built-in functions are compiled to ground weight rules and then to a set of primitive rules and in the second phase the answers, the stable models, are computed as the stable models of the primitive rules. Our implementation handles a subclass of rules, domain-restricted ones, which enable an efficient compilation method based on database techniques. The second phase is based on a Davis-Putnam like backtracking search procedure for which efficient pruning techniques and search heuristics have been developed. The comparisons with state-of-the-art satisfiability checkers show that our implementation techniques are quite competitive.

Acknowledgments

The support of the Academy of Finland (Project 43963) is gratefully acknowledged. The work of the second author has also been supported by the Helsinki Graduate School in Computer Science and Engineering. The authors would like to thank Tommi Syrjänen for developing and implementing the `lparse` procedure.

References

- Brewka, G. and Eiter, T. (1998). Preferred answer sets for extended logic programs. In Cohn, A., Schubert, L., and Shapiro, S., editors, *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, pages 86–97, Trento, Italy. Morgan Kaufmann Publishers.
- Buccafurri, F., Leone, N., and Rullo, P. (1997). Strong and weak constraints in disjunctive Datalog. In Dix, J., Furbach, U., and Nerode, A., editors, *Proceedings of the Fourth International Conference on Logic Programming and Non-Monotonic Reasoning*, pages 2–17, Dagstuhl Castle, Germany. Springer-Verlag.

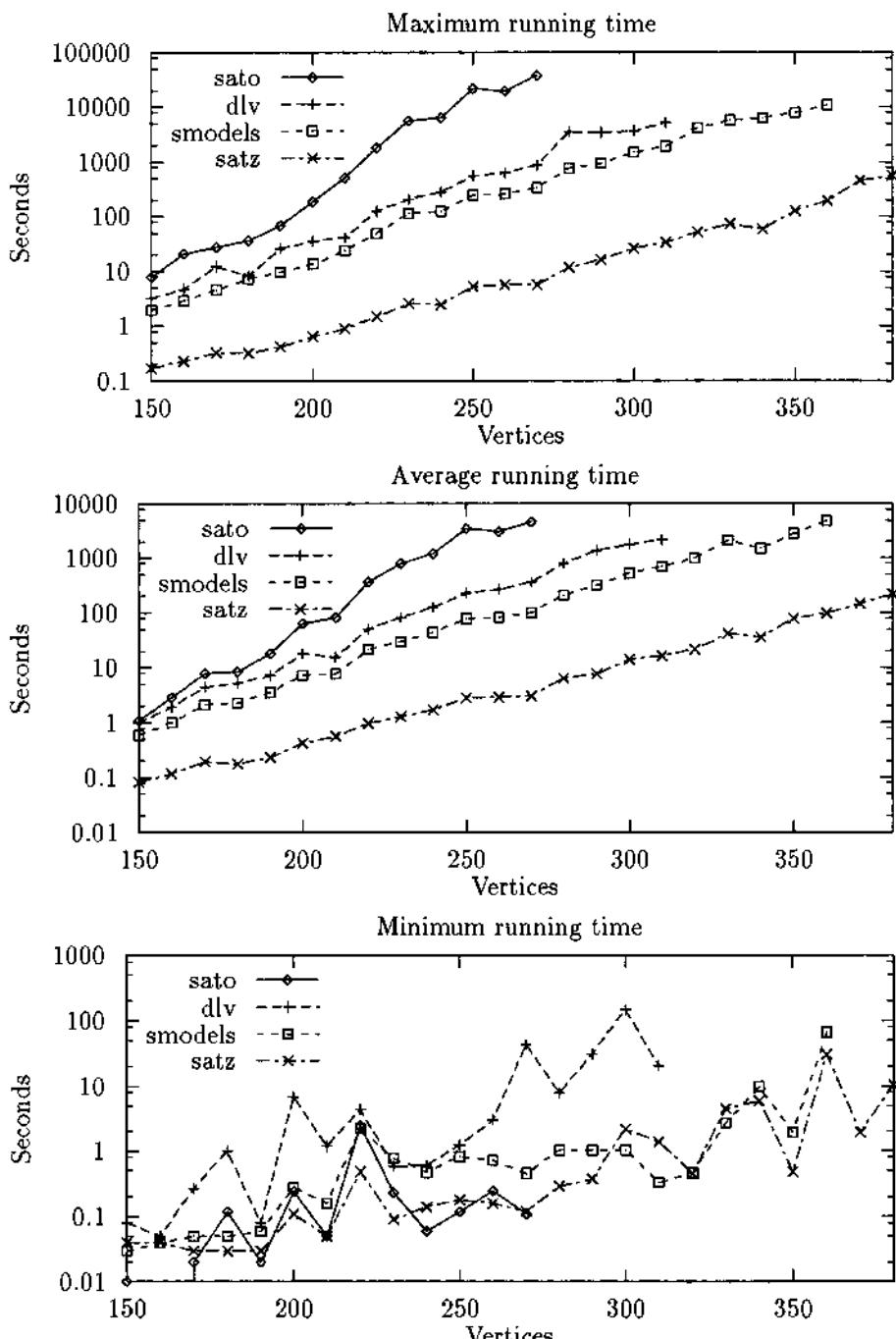


Figure 21.2 Experimental results on 3-SAT problems

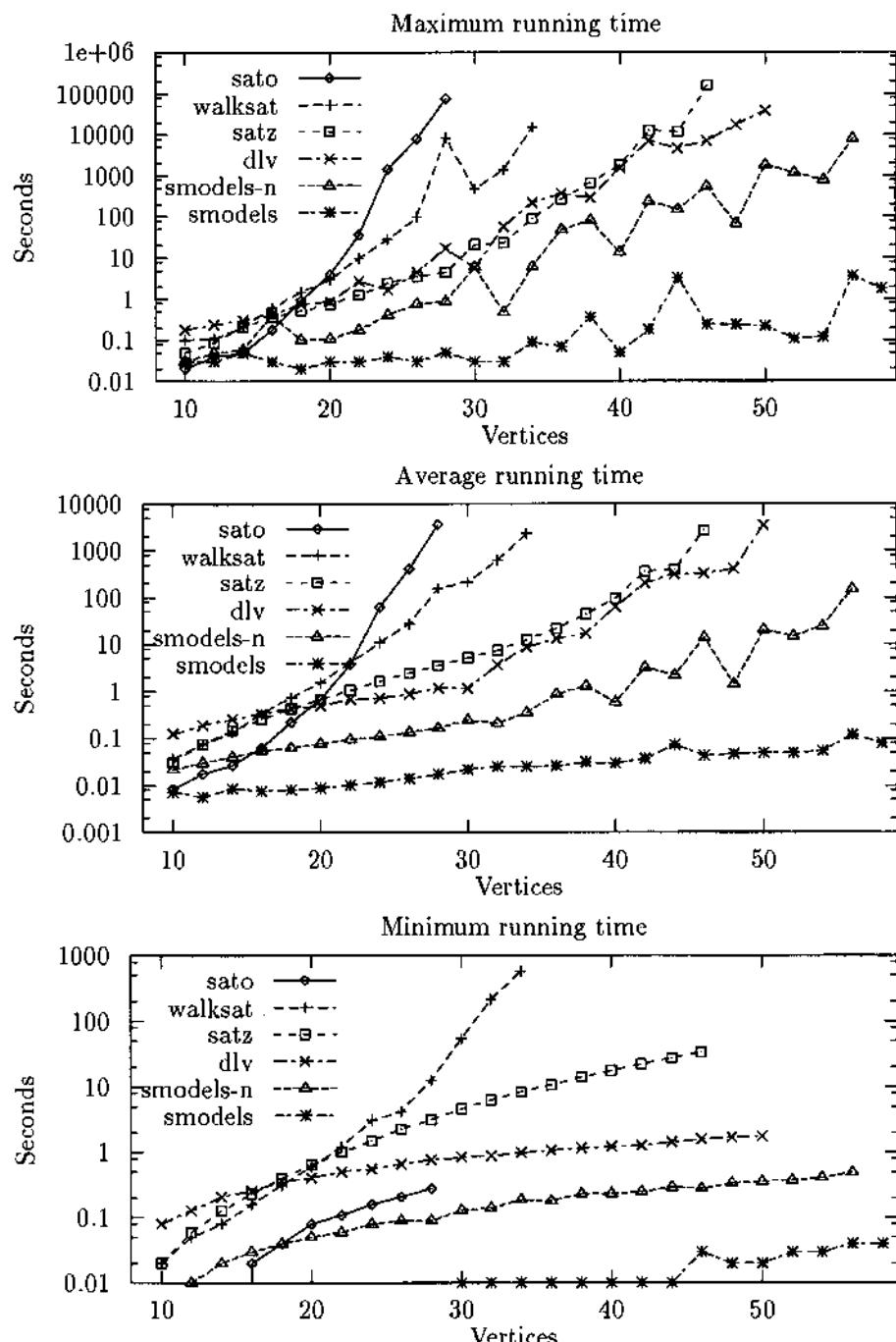


Figure 21.3 Experimental results on Hamiltonian cycle problems

- Cadoli, M., Palopoli, L., Schaerf, A., and Vasile, D. (1999). NP-SPEC: An executable specification language for solving all problems in NP. In Gupta, G., editor, *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages*, pages 16–30, San Antonio, Texas. Springer-Verlag.
- Crawford, J. and Auton, L. (1996). Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, 81(1):31–57.
- Eiter, T. and Gottlob, G. (1995). On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15:289–323.
- Eiter, T., Gottlob, G., and Mannila, H. (1997). Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418.
- Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarnello, F. (1998). The KR system dlv: Progress report, comparisons and benchmarks. In Cohn, A., Schubert, L., and Shapiro, S., editors, *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, pages 406–417, Trento, Italy. Morgan Kaufmann Publishers.
- Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Seattle, USA. The MIT Press.
- Gelfond, M. and Lifschitz, V. (1990). Logic programs with classical negation. In *Proceedings of the Seventh International Conference on Logic Programming*, pages 579–597, Jerusalem, Israel. The MIT Press.
- Greco, S. (1999). Dynamic programming in Datalog with aggregates. *IEEE Transactions on Knowledge and Data Engineering*, 11(2):265–283.
- Jaffar, J. and Lassez, J.-L. (1987). Constraint logic programming. In O'Donnell, M. J., editor, *Conference Record of the 14th Annual ACM Symposium on Principles of Programming Languages*, pages 111–119, Munich, FRG. ACM Press.
- Kautz, H. and Selman, B. (1999). Unifying sat-based and graph-based planning. In Dean, T., editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 318–325, Stockholm, Sweden. Morgan Kaufmann Publishers.
- Knuth, D. (1993). The Stanford GraphBase. <http://labrea.stanford.edu/pub/sgb/>.
- Leone, N. et al. (1999). Dlv, a disjunctive Datalog system. <http://www.dbai.tuwien.ac.at/proj/dlv/>.
- Li, C. and Anbulagan (1997). Look-ahead versus look-back for satisfiability problems. In Smolka, G., editor, *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming*, pages 341–355, Linz, Austria. Springer-Verlag.
- Lifschitz, V. (1999). Answer set planning. In De Schreye, D., editor, *Proceedings of the 16th International Conference on Logic Programming*, pages 25–37, Las Cruces, New Mexico. The MIT Press.
- Lu, J., Nerode, A., and Subrahmanian, V. (1996). Hybrid knowledge bases. *IEEE Trans. on Knowledge and Data Engineering*, 8(5):773–785.

- Marek, W. and Truszczyński, M. (1999a). Logic programming with costs. Manuscript which is available at <http://www.cs.engr.uky.edu/~mirek/papers.html>.
- Marek, W. and Truszczyński, M. (1999b). Stable models and an alternative logic programming paradigm. In Apt, K., Marek, V., Truszczyński, M., and Warren, D., editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag.
- McCain, N. (1999). The causal calculator. <http://www.cs.utexas.edu/users/mccain/cc/>.
- Ng, R. and Subrahmanian, V. (1994). Stable semantics for probabilistic deductive databases. *Information and Computation*, 110:42–83.
- Niemelä, I. (1999). Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273.
- Niemelä, I., Simons, P., and Soininen, T. (1999). Stable model semantics of weight constraint rules. In Gelfond, M., Leone, N., and Pfeifer, G., editors, *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 317–331, El Paso, Texas, USA. Springer-Verlag.
- Papadimitriou, C. (1995). *Computational Complexity*. Addison-Wesley Publishing Company.
- Sakama, C. and Inoue, K. (1994). An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of Automated Reasoning*, 13:145–172.
- Selman, B., Kautz, H., and Cohen, B. (1994). Noise strategies for improving local search. In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 337–343, Seattle, WA. AAAI Press.
- Simons, P. (1999a). Extending the stable model semantics with more expressive rules. In Gelfond, M., Leone, N., and Pfeifer, G., editors, *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 305–316, El Paso, Texas, USA. Springer-Verlag.
- Simons, P. (1999b). smodels, a procedure for computing stable models of ground programs. <http://www.tcs.hut.fi/Software/smodels/>.
- Soininen, T. and Niemelä, I. (1999). Developing a declarative rule language for applications in product configuration. In Gupta, G., editor, *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages*, pages 305–319, San Antonio, Texas. Springer-Verlag.
- Syrjänen, T. (1999). lpars, a procedure for grounding domain-restricted logic programs. <http://www.tcs.hut.fi/Software/smodels/lparse/>.
- Truszczyński, M. et al. (1999). DeReS, a default reasoning system. <http://www.cs.engr.uky.edu/ai/deres.html>.
- Tsang, E. (1993). *Foundations of Constraint Satisfaction*. Academic Press, London.
- Warren, D. et al. (1999). The XSB programming system. <http://www.cs.sunysb.edu/sbprolog/xsb-page.html>.
- Zaniolo, C. et al. (1999). LDL⁺⁺, a second-generation deductive database system. <http://www.cs.ucla.edu/ldl/>.

- Zhang, H. (1997). SATO: An efficient propositional prover. In McCune, W., editor, *Proceedings of the 14th International Conference on Automated Deduction*, pages 272–275, Townsville, North Queensland, Australia. Springer-Verlag.

Chapter 22

NONMONOTONIC REASONING IN *LDL⁺⁺*

Haixun Wang,
Computer Science Department
University of California at Los Angeles
hxwang@cs.ucla.edu

Carlo Zaniolo
Computer Science Department
University of California at Los Angeles
zaniolo@cs.ucla.edu

Abstract Deductive database systems have made major advances on efficient support for nonmonotonic reasoning. A first generation of deductive database systems supported the notion of stratification for programs with negation and set aggregates. Stratification is simple to understand and efficient to implement but it is too restrictive; therefore, a second generation of systems seeks efficient support for more powerful semantics based on notions such as well-founded models and stable models. In this respect, a particularly powerful set of constructs is provided by the recently enhanced *LDL⁺⁺* system that supports (i) monotonic user-defined aggregates, (ii) XY-stratified programs, and (iii) the nondeterministic choice constructs under stable model semantics. This integrated set of primitives supports a terse formulation and efficient implementation for complex computations, such as greedy algorithms and data mining functions, yielding levels of expressive power unmatched by other deductive database systems.

Keywords: Deductive Databases, Nonmonotonic Reasoning, Stratification, Monotonic Aggregation

1 INTRODUCTION

The *LDL* system (Chimenti et al., 1990) developed at MCC in the 80's is representative of a generation of Deductive Database Systems (Minker 1996) that sought to integrate logic-based rule languages into relational databases,

in order to provide efficient and scalable support for sophisticated queries on large data sets and knowledge bases. Therefore, most of these systems (Ramakrishnan and Ullman 1995) followed a (bottom-up) fixpoint approach for both abstract and operational semantics, rather than the (top-down) SLD-resolution of Prolog, since the former was considered more conducive to mapping into relational algebra and secondary-store implementation techniques of database systems than the stack-oriented implementation techniques of Prolog (Zaniolo et al., 1997). Indeed, this line of work led to implementation techniques, such as differential fixpoint and magic sets that are now included in most commercial database systems and SQL3 standards (Filkenstein et al., 1996); also it leads to the ability of the $\mathcal{L}\mathcal{DL}/\mathcal{L}\mathcal{DL}++$ system to compile rules and predicates into SQL when these are supportable on external databases. These techniques have already been documented extensively (Chimenti et al., 1990, Ramakrishnan et al., 1993, Abiteboul et al., 1995, Ramakrishnan and Ullman 1995, Zaniolo et al., 1997) and will not be further discussed here.

In this paper, we will instead discuss the nonmonotonic reasoning techniques that were developed later, largely in response to the experience of deploying these systems in real life applications. Indeed, it was found that the stratification condition obeyed by $\mathcal{L}\mathcal{DL}$ and most systems of this generation was too restrictive and prevents the efficient expression of quintessential database queries, such as Bill of Materials (BoM) queries and the optimized transitive-closure queries needed to compute efficiently least-cost paths in a graph.

Many new notions and techniques were therefore proposed and added to deductive database systems to support negation and set aggregates in nonstratified logic programs (Ramakrishnan and Ullman 1995). In particular, $\mathcal{L}\mathcal{DL}$ was followed by $\mathcal{L}\mathcal{DL}++$ which, with the recent extensions implemented at UCLA (Zaniolo et al. 1998), provides a powerful and comprehensive set of nonmonotonic constructs; these extensions have been used in advanced applications such as AI planning (Brogi et al., 1997) and data mining (Bonchi et al., 1999).

The issues posed by nonmonotonicity can be illustrated by the notion of stratification (Przymusinski 1988) which, while lacking in expressive power, is rich in the other important qualities that are desirable in nonmonotonic constructs. Indeed, stratification (i) has a formal declarative semantics, (ii) has good usability, since it is easy for users to understand, (iii) is amenable to efficient implementation (using an iterated fixpoint procedure (Przymusinski 1988, Zaniolo et al., 1997)), and (iv) is applicable to set-aggregates as well as negation.

The problem of generalizing stratified programs without compromising the four properties described above proved to be a real challenge. The notions of well-founded models (Van Gelder et al., 1990), and stable models (Gelfond and Lifschitz) provide a sound basis for formal semantics; but then, there remain the issues of performance (since finding stable models is \mathcal{NP} -hard (Schlipf et al., 1992)), and also the issue of usability. The usability issue follows from the complexity of the notions involved, and the fact that the semantic well-formedness of a program cannot be checked at compile time, since the existence

of stable models or well-founded models for a given program depends on the database content besides the rules¹.

Some Deductive Database Systems (Ramakrishnan and Ullman 1995) provide support for modular stratification (Ross 1994), which is more powerful than stratification but fail to solve the usability issue. On the other hand, the notion of XY-stratification supported in $\mathcal{L}\mathcal{D}\mathcal{L}^{++}$ solves both the issue of expressive power and that of usability (Zaniolo et al., 1993).

A second line of attack, rather than attempting to solve the nonmonotonicity conundrum, tries to circumvent it by observing that in many applications of interest, e.g., greedy algorithms (Ganguly et al., 1995), aggregates are used in a monotonic way. This idea of monotone aggregation was pursued in a general setting in (Ross et al., 1997), where several interesting examples were studied in which aggregates are monotonic in lattices different from the standard set-containment lattice used for Horn clauses. Unfortunately, the lack of viable techniques to detect these special lattices (Van Gelder, 1993) prevented the implementation of this idea in actual DDB systems. This situation has changed recently with the introduction in $\mathcal{L}\mathcal{D}\mathcal{L}^{++}$ of user-defined aggregates (UDAs) that support *early returns*. In fact, we discovered (Zaniolo and Wang, 1999) that UDAs that only use early returns (no final returns) are monotonic in the standard lattice of set containment; thus, they can be used freely in recursive $\mathcal{L}\mathcal{D}\mathcal{L}^{++}$ programs, or SQL queries—unlike aggregates with final returns that are nonmonotonic and can only be used in XY-stratified programs.

Our presentation is organized as follows. In the next section, we introduce the construct of choice, and focus on programs with negation that have nondeterministic but monotonic stable-model semantics. Then, in Section 3, we use choice to define the semantics of aggregates and provide syntactic characterization for monotonic and nonmonotonic user-defined aggregates. Finally, in Section 4, we describe the notion of XY-stratification for recursive rules containing negation and nonmonotonic aggregates.

2 MONOTONIC NONDETERMINISM

Say that our database contains relations `student(Name, Major, Year)`, and `professor(Name, Major)`. In fact, let us take a toy example that only has the following facts:

<code>student('JimBlack', ee, senior).</code>	<code>professor(ohm, ee).</code>
	<code>professor(bell, ee).</code>

Now, the rule is that the major of a student must match his/her advisor's major area of specialization. Then, eligible advisors can be computed as follows:

¹Unlike Prolog where the programmer sees both facts and rules, only the rules are normally available to the programmer of a deductive database system (along with the database schema that contains type information for the tables but not their content).

```
elig_adv(S,P) ← student(S,Majr,Year), professor(P,Majr).
```

This yields

```
elig_adv('JimBlack',ohm).
elig_adv('JimBlack',bell).
```

But, since a student can only have one advisor, the goal `choice((S),(P))` must be added to our rule to force the selection of a unique advisor for each student:

Example 1 Computation of unique advisors by a choice rule

```
actual_adv(S,P) ← student(S,Majr,Yr), professor(P,Majr),
choice((S),(P)).
```

The goal `choice((S),(P))` can also be viewed as enforcing a *functional dependency* (FD) $S \rightarrow P$ on the results produced by the rule; thus, in `actual_adv`, the second column (professor name) is functionally dependent on the first one (student name). As in FDs, S and P will be called the left side and the right side of this choice goal. The right side of a choice goal cannot be empty, but its left side can be empty, denoting that all tuples produced must share the same values for the right side attributes.

The result of this rule is *nondeterministic*: it can either return a singleton relation containing the tuple ('JimBlack',ohm), or one containing the tuple ('JimBlack',bell).

A program where the rules contain choice goals is called a *choice program*. The semantics of a choice program P can be defined by transforming P into a program with negation, $f\circ e(P)$, called the *first order equivalent* of P . Now, $f\circ e(P)$ exhibits a multiplicity of stable models, each obeying the FDs defined by the choice goals; each such stable model corresponds to an alternative set of answers for P and is called a *choice model* for P . The first order equivalent of Example 1 is as follows:

Example 2 The first order equivalent for Example 1

```
actual_adv(S,P) ← student(S,Majr,Yr), professor(P,Majr),
chosen(S,P).

chosen(S,P) ← student(S,Majr,Yr), professor(P,Majr),
¬diffChoice(S,P).

diffChoice(S,P) ← chosen(S,P'), P ≠ P'.
```

This can be read as a statement that a professor will be assigned to a student whenever a different professor has not been assigned to the same student. In general, $foe(P)$ is defined as follows:

Definition 1 Let P denote a program with choice rules: its first order equivalent $foe(P)$ is obtained by the following transformation. Consider a choice rule r in P :

$$r : A \leftarrow B(Z), \text{choice}((X_1), (Y_1)), \dots, \text{choice}((X_k), (Y_k)).$$

where,

- (i) $B(Z)$ denotes the conjunction of all the goals of r that are not choice goals, and
- (ii) $X_i, Y_i, Z, 1 \leq i \leq k$, denote vectors of variables occurring in the body of r such that $X_i \cap Y_i = \emptyset$ and $X_i, Y_i \subseteq Z$.

Then, $foe(P)$ is constructed P as follows:

1. Replace r with a rule r' obtained by substituting the choice goals with the atom $chosen_r(W)$:

$$r' : A \leftarrow B(Z), chosen_r(W).$$

where $W \subseteq Z$ is the list of all variables appearing in choice goals, i.e., $W = \bigcup_{1 \leq j \leq k} X_j \cup Y_j$.

2. Add the new rule

$$chosen_r(W) \leftarrow B(Z), \neg diffChoice_r(W).$$

3. For each choice atom $choice((X_i), (Y_i))$ ($1 \leq i \leq k$), add the new rule

$$diffChoice_r(W) \leftarrow chosen_r(W'), Y_i \neq Y'_i.$$

where (i) the list of variables W' is derived from W by replacing each $A \notin X_i$ with a new variable A' (i.e., by priming those variables), and (ii) $Y_i \neq Y'_i$ is true if $A \neq A'$, for some variable $A \in Y_i$ and its primed counterpart $A' \in Y'_i$.

Properties of Choice Models

Theorem 1 Let P be a positive program with choice rules. Then the following properties hold (Giannotti et al., 1991):

- $foe(P)$ has one or more total stable models.
- The $chosen$ atoms in each stable model of $foe(P)$ obey the FDs defined by the choice goals.

Observe that the $\text{foe}(P)$ of a program with choice does not have total well-founded models; in fact, for our Example 1, the well-founded model yields undefined values for advisors. Therefore, the choice can express nondeterministic semantics, which can be also expressed by stable models but not well-founded models. On the other hand, the choice model avoids the exponential complexity which is normally encountered in programs with nondeterministic semantics, such as disjunctive logic programs (Eiter et al., 2000). Indeed, the computation of stable models is \mathcal{NP} -hard (Schilp et al., 1992), but the computation of choice models for positive programs can be performed in polynomial time with respect to the size of the database. This, basically, is due to the monotonic nature of the choice construct that yields a simple fixpoint computation for programs with choice (Giannotti et al., 2000). We show next that the use of choice rules in positive programs preserve their monotonic properties. A program P can be viewed as consisting of two separate components: an extensional component (i.e., the database facts), denoted $\text{edb}(P)$, and an intensional one (i.e., the rules), denoted $\text{idb}(P)$. Then, a positive choice program defines a monotonic multi-valued mapping from $\text{edb}(P)$ to $\text{idb}(P)$, as per the following theorem proven in (Giannotti et al., 2000):

Theorem 2 Let P and P' be two positive choice programs where $\text{idb}(P') = \text{idb}(P)$ and $\text{edb}(P') \supseteq \text{edb}(P)$. Then, if M is a choice model for P , then, there exists a choice model M' for P' such that $M' \supseteq M$.

Thus positive choice programs represent a class of logic programs that are very well-behaved from both semantic and computational aspects. The same can be said for choice programs with stratified negation that are defined next.

Definition 2 Let P be a choice program with negated goals. Then, P is said to be stratified when the program obtained from P by removing its choice goals is stratified.

The stable models for a stratified choice program P can be computed using an *iterated choice fixpoint* procedure that directly extends the iterated fixpoint procedure for programs with stratified negation (Przymusinski 1988, Zaniolo et al., 1997); this is summarized next. Let P_i denote the rules of P (whose head is) in stratum i , and let P_i^* be the union of P_j , $j \leq i$. Now, if M_i is a stable model for P_i^* , then every stable model for $M_i \cup P_{i+1}$ is a stable model for the program P_{i+1}^* . Therefore, the stable models of stratified choice programs can be computed by modifying the iterated fixpoint procedure used for stratified programs so that choice models (rather than the least models) are computed for strata containing choice rules (Giannotti et al., 1999).

Programs with Choice

Choice significantly extends the power of Datalog by allowing *nondeterministic* queries; moreover using choice in Datalog we can express important *deterministic* queries that cannot be expressed in stratified Datalog (Giannotti et al., 2000). This can be illustrated by the following choice program that

places the elements of a relation $d(Y)$ into a chain, thus establishing a random total order on these elements.

Example 3 Linear sequencing of the elements of a set. The elements of the set are stored by means of facts of the form $d(Y)$.

```
chain(nil, nil).
chain(X, Y) ←      chain(_, X), d(Y),
                     choice((X), (Y)), choice((Y), (X)).
```

Here `chain(nil, nil)` is the root of a chain linking all the elements of $d(Y)$ —thus inducing a total order on elements of d . Because of the ability of choice programs to order the elements of a set, Datalog with choice and stratified negation is P-time complete and can, for instance, express the parity query that determines if a relation has an even number of elements (Abiteboul et al., 1995). This query cannot be expressed in Datalog with stratified negation unless we assume that the underlying universe is totally ordered—an assumption that violates the data independence principle of *genericity* (Abiteboul et al., 1995).

The expressive power of choice was studied in (Giannotti et al., 2000) where it was shown that it is more powerful than other nondeterministic constructs, such as the witness operator (Abiteboul et al., 1995) and the original version of choice proposed in (Krishnamurthy and Naqvi, 1988); e.g., these other operators can neither order domains nor express the parity query (Giannotti et al., 1991).

Example 4 Rooted spanning tree. We are given an undirected graph where an edge joining two nodes, say x and y , is represented by the pair $g(x, y)$ and $g(y, x)$. Then, a spanning tree in this graph, starting from the source node a , can be constructed by the following program:

```
st(root, a).
st(X, Y) ←      st(_, X), g(X, Y), Y ≠ a, Y ≠ X,
                  choice((Y), (X)).
```

To illustrate the presence of multiple total choice models for this program, take a simple graph consisting of the following arcs:

```
g(a, b).  g(b, a).
g(b, c).  g(c, b).
g(a, c).  g(c, a).
```

After the exit rule adds `st(root, a)`, the recursive rule could add `st(a, b)` and `st(a, c)` along with the two tuples `chosen(a, b)` and `chosen(a, c)` in the `chosen` table. No further arc can be added after those, since the addition of `st(b, c)`

or $st(c, b)$ would violate the FD that follows from $\text{choice}((Y), (X))$ enforced through the chosen table. However, since $st(\text{root}, a)$ was produced by the first rule (the exit rule), rather than the second rule (the recursive choice rule), the table chosen contains no tuple with second argument equal to the source node a . Therefore, to avoid the addition of $st(c, a)$ or $st(b, a)$, the goal $Y \neq a$ was added to the recursive rule.

By examining all possible solutions, we conclude that this program has three different choice models, for which we list only the st -atoms, below:

1. $st(a, b), st(b, c).$
2. $st(a, b), st(a, c).$
3. $st(a, c), st(c, b).$

3 AGGREGATES IN LOGIC

The expressive power of choice can be put to use to provide an inductive definition of aggregates in logic. Say for instance that we want to define the aggregate avg that returns the average of Y -values that satisfy $d(Y)$. Using the notation first used in *CDL* (Chimenti et al., 1990), and then in *CORAL* (Ramakrishnan et al., 1993) and *CDL++*, this computation can be specified by the following rule:

$$p(\text{avg}(Y)) \leftarrow d(Y).$$

A logic-based equivalent for this rule is

$$p(Y) \leftarrow \text{results}(\text{avg}, Y).$$

where $\text{results}(\text{avg}, Y)$ is derived from $d(Y)$ by (i) the chain rules, (ii) the $cagr$ rules and (iii) the return rules. The chain rules are those of Example 3, that place the elements of $d(Y)$ into an order-inducing chain. Then, the $cagr$ rules perform the inductive computation by calling the single and multi rules as follows:

```
cagr(AgName, Y, New) ← chain(nil, Y), Y ≠ nil, single(avg, Y, New).
cagr(AgName, Y2, New) ← chain(Y1, Y2), cagr(AgName, Y1, Old),
                           multi(AgName, Y2, Old, New).
```

Thus, the $cagr$ rules are used to memorize the previous results, and to apply (i) single to the first element of $d(Y)$ (i.e., for the pattern $\text{chain}(\text{nil}, Y)$) and (ii) multi to the successive elements. The return rules are as follows:

```

results(AgName, Yield) ← chain(Y1, Y2), cagr(AgName, Y1, Old),
                           ereturn(AgName, Y2, Old, Yield).
results(AgName, Yield) ← chain(X, Y), ¬chain(Y, _),
                           cagr(AgName, Y, Old),
                           freturn(AgName, Y, Old, Yield).

```

Therefore we first compute `chain`, and then `cagr` that applies the `single` and `multi` to every element in the chain. Concurrently, the first `results` rule produces all the results that can be generated by the application of `ereturn` to each element of the chain. The final returns are however computed by the second `results` rule by the application of `freturn` once the last element in the chain (i.e., the element without successors) is detected. This is the only rule using negation; in the absence of `freturn` this rule can be removed yielding a positive choice program that is monotonic by Theorem 2. Thus every aggregate with only early returns is *monotonic with respect to set-containment* and can be used freely in recursive rules.

To define a new aggregate, the user must write the `single`, `multi`, `ereturn` and `freturn` rules; the remaining rules are built in the system. For instance, the computation of `avg` requires the accumulation of the count and the sum of the elements visited so far. Thus `single(avg, ...)` initializes the count to 1 and the sum to the value of the actual element `Y`, while `multi(avg, ...)` increases the count by one and the sum by the new value. Therefore, for `avg` the user can write the following rules:

```

single(avg, Y, cs(1, Y)).
multi(avg, Y, cs(Cnt, Sum), cs(Cnt1, Sum1)) ←
    Cnt1 = Cnt + 1, Sum1 = Sum + Y.

```

Then, the user writes the `freturn` rule that upon visiting the final element in `d(Y)` produces the ratio of sum over count, as follows:

```

freturn(avg, Y, cs(Cnt, Sum), Val) ← Val = Sum/Cnt.

```

To compute the average salary of employees grouped by `Dno`, the user can write:

```

avgsal(Dno, avg(Sal)) ← emp(Eno, Sal, Dno).

```

Each aggregate is defined by its own `single`, `multi`, `ereturn` and `freturn` rules, where the unique aggregate name is used as the first argument in the head of these rules to eliminate interference between the rules defining different aggregates. Also, for the general situation where there are group-by attributes, such as `Dno` in the rule above, then these must be added as additional arguments to the predicates of the equivalent rules (Zaniolo and Wang, 1999).

Applications of User Defined Aggregates (UDAs)

From the above discussion it follows that traditional SQL aggregates can be characterized as shorthand for logic programs with negation under stable model semantics. As shown below, the same is true for new UDAs. For instance, say that from a set of pairs such as $(\text{Name}, \text{YearOfBirth})$ as input, we want to return the **Name** of the youngest person (i.e., the person born in the latest year). This computation, which cannot be expressed directly as an aggregate in SQL, is easy to express by the UDA **youngest**, below (in $\mathcal{L}\mathcal{D}\mathcal{L}++$, a vector of n arguments (x_1, \dots, x_n) is basically treated as an n -argument function with a default name).

```

single(youngest, (N, Y), (N, Y)).
multi(youngest, (N, Y), (N1, Y1), (N, Y)) ←      Y ≥ Y1.
multi(youngest, (N, Y), (N1, Y1), (N1, Y1)) ←   Y ≤ Y1.
freturn(youngest, (N, Y), (N1, Y1), N1).

```

User-defined aggregates provide a simple solution to a number of complex problems in deductive databases; due to space limitations, we consider here few simple examples, and more examples can be found in (Zaniolo et al. 1998).

The power of UDAs is significantly enhanced by *early returns*. A first important application of early returns is online aggregation (Hellerstein et al., 1997). This can be used to provide good estimates for the value of an aggregate long before the whole data set is visited. For instance, say that we want to see the average value obtained so far every 100 elements. Then, we can add the following **ereturn**:

```

ereturn(avg, X, (Sum, Count), Avg) ←
    Count mod 100 = 0, Avg = Sum/Count.

```

As second example, let us consider the well-known problem of coalescing after temporal projection in temporal databases (Zaniolo et al., 1997). For instance in Example 5, below, after projecting out from the employee relation the salary column, we might have a situation where the same **Eno** appears in tuples where their valid-time intervals overlap; then these intervals must be coalesced. Here, we use closed intervals represented by the pair (From, To) where **From** is the start-time, and **To** is the end-time. Under the assumption that tuples are sorted by increasing start-time, then we can use a special **coales** aggregate to perform the task in one pass through the data.

Example 5 *Coalescing overlapping intervals sorted by start time.*

```

empProj(Eno, coales((From, To))) ←  emp(Eno, -, -, (From, To)).

```

```

single(coales,(Frm,To),(Frm,To)).
multi(coales,(Nfr,Nto),(Cfr,Cto),(Cfr,Cto)) ←
    Nfr <= Cto, Nto > Cto.
multi(coales,(Nfr,Nto),(Cfr,Cto),(Cfr,Cto)) ← Nto <= Cto.
multi(coales,(Nfr,Nto),(Cfr,Cto),(Cfr,Cto)) ← Cto < Nfr.

ereturn(coales,(Nfr,Nto),(Cfr,Cto),(Cfr,Cto)) ← Cto < Nfr.
freturn(coales,_,LastInt,LastInt).

```

In the multi rules, when the new interval (Nfr, Nto) overlaps the current interval (Cfr, Cto) (i.e., when $Nfr \leq Cto$ given that $Cfr \leq Nfr$ by the assumption that input intervals are ordered by their start time), then we coalesce the two into an interval that begins in Cfr and ends with the larger of Nto and Cto . Otherwise, the current interval is returned while the new interval becomes the current one.

A rule r whose head contains aggregates is called an *aggregate rule*. A program P is said to be stratified w.r.t. aggregates when for each aggregate rule r in P , the stratum of its head predicate is strictly higher than the stratum of each body predicate. The previous program is stratified w.r.t. `coales` which is nonmonotonic since it uses both early returns and final returns.

Therefore, nonmonotonic UDAs in stratified programs are very useful in many applications. However, stratified programs are too restrictive for many advanced applications. A first solution to this problem consists in using a form of locally stratified programs called XY-stratified programs, which will be discussed in Section 3. The second solution consists in using monotonic aggregates that eliminate the need for stratification all together. As discussed in the next section, monotonic aggregates can be used freely in recursion, to express many important algorithms (Wang and Zaniolo, 2000).

Monotone Aggregation

We next define a continuous count that returns the current count after each new element (thus, it does not have a `freturn` since that would be redundant).

```

single(mcount,Y,1).
multi(mcount,Y,Old,New) ← New = Old + 1.
ereturn(mcount,Y,Old,New) ← New = Old + 1.

```

The next two examples are derived from (Ross et al., 1997).

Join the Party. Some people will come to the party no matter what, and their names are stored in a `sure(Person)` relation. But others will join only after they know that at least $K = 3$ of their friends will be there. Here, `friend(P,F)` denotes that F is P 's friend.

```

willcome(P) ←           sure(P).
willcome(P) ←           c_friends(P, K), K ≥ 3.
c_friends(P, mcount(F)) ← willcome(F), friend(P, F).

```

Consider now a computation of these rules on the following database.

```

friend(jerry, mark).    sure(mark).
friend(penny, mark).   sure(tom).
friend(jerry, jane).   sure(jane).
friend(penny, jane).
friend(jerry, penny).
friend(penny, tom).

```

Then, the basic semi-naïve computation yields:

```

willcome(mark), willcome(tom), willcome(jane),
c_friends(jerry, 1), c_friends(penny, 1), c_friends(jerry, 2),
c_friends(penny, 2), c_friends(penny, 3), willcome(penny),
c_friends(jerry, 3), willcome(jerry).

```

This example illustrates how the standard semi-naïve computation can be applied to queries containing monotone user-defined aggregates. Another interesting example is transitive ownership and control of corporations.

Company Control. Say that `owns(C1, C2, Per)` denotes the percentage of shares that corporation `C1` owns of corporation `C2`. Then, `C1` controls `C2` if it owns more than, say, 50% of its shares. In general, to decide whether `C1` controls `C3` we must also add the shares owned by corporations such as `C2` that are controlled by `C1`. This yields the transitive control rules defined with the help of a continuous sum aggregate that returns the partial sum for each new element:

```

control(C, C) ←           owns(C, _, _).
control(Onr, C) ←           towns(Onr, C, Per), Per > 50.
towns(Onr, C2, msum(Per)) ← control(Onr, C1), owns(C1, C2, Per).

single(msum, Y, Y).
multi(msum, Y, Old, New) ←   New = Old + Y.
ereturn(msum, Y, Old, New) ← New = Old + Y.

```

Thus, every company controls itself, and a company C_1 that has transitive ownership of more than 50% of C_2 's shares controls C_2 . In the last rule, `towns` computes transitive ownership with the help of `msum` that adds up the shares of controlling companies. Observe that any pair (Onr, C_2) is added at most once to `control`, thus the contribution of C_1 to Onr 's transitive ownership of C_2 is only accounted once.

Bill-of-Materials (BoM) Applications. BoM applications represent an important application area that requires aggregates in recursive rules. Say, for instance that `assembly(P1, P2, QT)` denotes that P_1 contains part P_2 in quantity QT . We also have elementary parts described by the relation `basic_part(Part, Price)`. Then, the following program computes the cost of a part as the sum of the cost of the basic parts it contains.

```
part_cost(Part, 0, Cst) ←      basic_part(Part, Cst).
part_cost(Part, mcount(Sb), msum(MCst)) ←
    part_cost(Sb, ChC, Cst), prolfc(Sb, ChC),
    assembly(Part, Sb, Mult), MCst = Cst * Mult.
```

Thus, the key condition in the body of the second rule is that a subpart S_b is counted in `part_cost` only when all S_b 's children have been counted. This occurs when the number of S_b 's children counted so far by `mcount` is equal to the out-degree of this node in the graph representing `assembly`. This number is kept in the prolificacy table, `prolfc(Part, ChC)`, which can be computed as follows:

```
prolfc(P1, count(P2)) ←  assembly(P1, P2, -).
prolfc(P1, 0) ←            basic_part(P1, -).
```

4 BEYOND STRATIFICATION

Several deductive database systems have addressed the need to go beyond stratification by supporting the notion of modular stratification (Ross 1994). Unfortunately, this approach suffers from poor usability, since the existence of a modular stratification for a program can depend on its extensional information (i.e., its fact base) and, in general, cannot be checked without executing the program. This should be contrasted with the concept of basic stratification, which instead provides a simple criterion for the programmer to follow and for the compiler to use when validating the program and optimizing its execution. Therefore, $\mathcal{L}\mathcal{D}\mathcal{L}^{++}$ uses the notion of XY-stratified programs that preserves the compilability and usability benefits of stratified programs while achieving the expressive power of well-founded models (Kemp et al., 1995). XY-stratified programs are locally stratified explicitly by a temporal arguments: i.e., they can be viewed as Datalog_{IS} programs, which are known to provide a powerful tool for temporal reasoning (Baudinet et al.,

1994, Zaniolo et al., 1997), or Statalog programs that were used to model active databases (Lausen et al., 1998). The deductive database system Aditi (Vaghanin et al., 1994, Kemp and Ramamohanarao, 1998) also supports the closely related concept of explicitly locally stratified programs, which were shown to be as powerful as well-founded models since they can express their alternating fixpoint computation (Kemp et al., 1995).

For instance, the ancestors of `marc`, with the number of generations that separates them from `marc`, can be computed using the following program which models the differential fixpoint computation:

Example 6 *Computing ancestors of Marc and their remoteness from Marc using differential fixpoint.*

```
r1 : delta_anc(0, marc).
r2 : delta_anc(J + 1, Y) ← delta_anc(J, X), parent(Y, X),
                           ~all_anc(J, Y).
r3 : all_anc(J + 1, X) ← all_anc(J, X).
r4 : all_anc(J, X) ← delta_anc(J, X).
```

This program is locally stratified by the first arguments in `delta_anc` and `all_anc` that serve as temporal arguments (thus `+1` is a postfix successor function symbol, much in the same way as $s(J)$ denotes the successor of J in Datalog_{IS} (Zaniolo et al., 1997)). The zero stratum consists of atoms of nonrecursive predicates such as `parent` and of atoms that unify with `all_anc(0, X)` or `delta_anc(0, X)`, where `X` can be any constant in the universe. The k^{th} stratum consists of atoms of the form `all_anc(k, X)`, `delta_anc(k, X)`. This program is locally stratified (Przymusinski 1988) since the heads of recursive rules belong to strata that are one above those of their goals. Alternatively, we can view this program as a compact representation for the stratified programs obtained by instantiating the temporal argument to integers that are then viewed as part of the predicate names.

Also observe that the temporal arguments in every rule are either the same as those in the head of the rule or one less. Then, there are two kinds of rules in our example: (i) an `X`-rule (i.e., a horizontal rule) where the temporal argument in each goal is the same as that in the head, (ii) a `Y`-rule (i.e., a vertical rule) where the temporal argument in some goal is one less than that in the head. Formally, let P be a set of rules defining mutually recursive predicates, where each recursive predicate has a distinguished temporal argument and every rule in P is either an `X`-rule or a `Y`-rule. Then, P will be said to be an XY-program. For instance, the program in Example 6 is an XY-program, where r_4 and r_1 are `X`-rules, while r_2 and r_3 are `Y`-rules.

A simple test can now be used to decide whether an XY-program P is locally stratified. The test begins by labeling all the head predicates in P with the prefix ‘new’. Then, the body predicates with the same temporal argument as the head are labeled with the prefix ‘new’, and the others are labeled with the prefix ‘old’. Finally, the temporal arguments are dropped from the program. The resulting program is called the *bistate version* of P and is denoted P_{bis} .

Example 7 The bistate version of the program in Example 6

```

new_delta_anc(marc).
new_delta_anc(Y) ← old_delta_anc(X), parent(Y, X),
                  ¬old_all_anc(Y).
new_all_anc(X) ← new_delta_anc(X).
new_all_anc(X) ← old_all_anc(X).

```

Now we have that (Zaniolo et al., 1993, Zaniolo et al., 1997):

Definition 3 Let P be an XY-program. P is said to be XY-stratified when P_{bis} is a stratified program.

Theorem 3 Let P be an XY-stratified program. Then P is locally stratified.

The program of Example 7 is stratified with the following strata: $S_0 = \{\text{parent}, \text{old_all_anc}, \text{old_delta_anc}\}$, $S_1 = \{\text{new_delta_anc}\}$, and $S_2 = \{\text{new_all_anc}\}$. Thus, the program in Example 6 is locally stratified.

For an XY-stratified program P , the iterated fixpoint of Algorithm (Zaniolo et al., 1993) becomes quite simple; basically it reduces to a repeated computation over the stratified program P_{bis} . However, since the temporal arguments have been removed from this program, we need to

1. store the temporal argument as an external fact `counter(T)`,
2. add a new goal `counter(Ir)` to each exit rule r in P_{bis} , where I_r is the variable from the temporal arguments of the original rule r , and
3. For each recursive predicate q add the rule:

```
q(J, X) ← new_q(X), counter(J).
```

The program so constructed will be called the *synchronized* bistate version of P , denoted $syncbi(P)$. For instance, to obtain the synchronized version of the program in Example 7, we need to change the first rule to

```
new_delta_anc(marc) ← counter(0).
```

since the temporal argument in the original exit rule was the constant 0. Then, we must add the following rules:

```

delta_anc(J, X) ← new_delta_anc(X), counter(J).
all_anc(J, X) ← new_all_anc(X), counter(J).

```

Then, the iterated fixpoint computation for an XY-stratified program can be implemented by the following procedure:

Procedure 4 Computing a stable model of an XY-stratified program P . Add the fact `counter(0)`. Then forever repeat the following two steps;

1. Compute the stable model of $\text{syncbi}(P)$.

2. For each recursive predicate `q`, replace `old_q` with `new_q`, computed in the previous step. Then, increase the value of `counter` by one.

Since $\text{syncbi}(P)$ is stratified, we can then use the iterated fixpoint computation to compute its stable model.

XY-stratified programs have unique computational advantages that are exploited in the $\mathcal{LDC}++$ implementation. For instance, the replacement of `old_q` with `new_q` described in the last step of the procedure becomes a zero-cost operation when it is properly implemented (e.g., by switching the pointers to the relations). A second improvement concerns *copy rules*, such as the last rule in Example 6. For instance r_3 in Example 6 is a copy rule that copies the new values of `all_anc` from its old values. Observe that the body and the head of this rule are identical, except for the prefixes `new` or `old`, in its bistate version (Example 7). Thus, in order to compute `new_all_anc`, we first execute the copy rule by simply setting the pointer to `new_all_anc` to point to `old_all_anc`—a zero-cost operation. Rule r_4 that adds tuples to `new_all_anc` must be executed after r_3 .

In writing XY-stratified programs, the user must also be concerned with termination conditions, since e.g., rule such as r_3 in Example 6 will keep generating the same results even after `delta` becomes empty. One solution to this problem is to add the goal `delta_anc(J, _)` to rule r_3 ; once no new arcs are found, then because of the negated goal in the second rule, there is no `new_delta_anc(J, _)` atom and the computation stops. Alternatively, these recursive rules might be called by a goal such as `delta_anc(J, Y)` in which case the $\mathcal{LDC}++$ system will stop as soon as no result is produced for some J value.

The next example solves the coalescing problem of Example 5 without assuming that tuples are sorted by their start-time. Therefore, we use the XY-stratified program of Example 8, which iterates over two basic computation steps. The first step is defined by the `overlap` rule. This determines pairs of distinct overlapping intervals, where the first interval contains the start of the second interval. The second step consists of deriving a new interval that begins at the start of the first interval, and ends at the later of the two endpoints. Finally, a rule `final_e_hist` returns the intervals that do not overlap other intervals (after eliminating the temporal argument).

Example 8 Coalescing overlapping periods into maximal periods after a projection

```

e_hist(0, Eno, Frm, To) ← emp_dep_sal(0, Eno, _, _, Frm, To).
overlap(J + 1, Eno, Frm1, To1, Frm2, To2) ←
    e_hist(J, Eno, Frm1, To1),
    e_hist(J, Eno, Frm2, To2),
    Frm1 ≤ Frm2, Frm2 ≤ To1,
    distinct(Frm1, To1, Frm2, To2).
e_hist(J, Eno, Frm1, To) ← overlap(J, Eno, Frm1, To1, Frm2, To2),
    select_larger(To1, To2, To).

final_e_hist(J + 1, Eno, Frm, To) ← e_hist(J, Eno, Frm, To),
    ¬overlap(J + 1, Eno, Frm, To, _, _).

distinct(Frm1, To1, Frm2, To2) ← To1 ≠ To2.
distinct(Frm1, To1, Frm2, To2) ← Frm1 ≠ Frm2.
select_larger(X, Y, X) ← X ≥ Y.
select_larger(X, Y, Y) ← Y > X.

```

As demonstrated by these examples, XY-stratified programs allow an efficient logic-based expression of procedural algorithms—in fact the alternating fixpoint used in the computation of well-founded models can also be expressed using these programs (Kemp et al., 1995). Also observe that, for the examples used here, the bistrate program is nonrecursive, as in the original definition of XY-stratification (Zaniolo et al., 1993). In general, by making the computation of the recursive predicate explicit as it was done for the `anc` example, it is possible to rewrite a program with a recursive bistrate graph into one where the bistrate program is nonrecursive.

Aggregates and Choice in XY-stratified Programs

The notion of XY-stratification can be extended easily to programs with choice and aggregates (Zaniolo et al., 1997).

Let P be a choice program with negated goals. Then P is said to be *XY-stratified* w.r.t. negation when the following two conditions hold:

- The program obtained from P by removing its choice goals is XY-stratified w.r.t. negation, and
- If r is a recursive choice rule in P , then some choice goal of r contains the variable from r 's temporal arguments in its left side.

Choice programs stratified w.r.t. negation always have (total) stable models (Giannotti et al., 1999) and there is a simple procedure for computing a stable model for such a program. The procedure begins with constructing the bistrate version P_{bis} of a program P by (i) dropping the temporal variable from the

choice goals, and (ii) applying the standard transformation to the rest of the rule. The synchronized version $\text{syncbi}(P)$ is constructed as previously described. Finally, Procedure 4 can be applied without modification.

XY-stratified programs with choice provide a very powerful modeling tool that has, e.g., been used to model the AI planning problem (Brogi et al., 1997) and active database rules (Zaniolo, 1997).

Let us consider now the situation of a recursive program P with aggregates. There is no restriction on P if it contains only monotonic aggregates and no negation. But recursive programs with nonmonotonic aggregates, or with negated goals and monotonic aggregates must satisfy the XY-stratification requirements discussed next. Say that given a program P with aggregates, we transform it into an equivalent choice program P' by the transformation described in Section 2. For P' to be an XY-stratified choice program, P must satisfy the following requirements:

- The bistratified version of P must be stratified w.r.t. negation and nonmonotonic aggregates, and
- For each recursive aggregate rule, the temporal variable must be contained in the group-by attributes.

After checking these simple conditions, the $LDC++$ compiler proceeds with the usual computation of $\text{syncbi}(P)$ as previously described.

Floyd's algorithm to compute the least-cost path between pairs of nodes in a graph can be expressed by the following XY-stratified program, where $g(X, Y, C)$ denotes an arc from X to Y of cost C :

Example 9 Floyd's least-cost paths between all node pairs.

```

delta(0, X, Y, C) ←      g(X, Y, C).
new(J + 1, X, Z, C) ←    delta(J, X, Y, C1), all(J, Y, Z, C2), C = C1 + C2.
new(J + 1, X, Z, C) ←    all(J, X, Y, C1), delta(J, Y, Z, C2), C = C1 + C2.
newmin(J, X, Z, min < C >) ←  new(J, X, Z, C).
discard(J, X, Z, C) ←   newmin(J, X, Z, C1), all(J, X, Z, C2), C1 ≥ C2.
delta(J, X, Z, C) ←     newmin(J, X, Z, C), ¬discard(J, X, Z, _).
all(J + 1, X, Z, C) ←   all(J, X, Z, C), ¬delta(J + 1, X, Z, _).
all(J, X, Z, C) ←       delta(J, X, Z, C).

```

This example uses a nonmonotonic min aggregate to select the least cost pairs among those just generated (observe that the temporal variable J is among the group-by attributes in this rule). The next rule derives the new `delta` pairs by discarding from `new` those that are larger than existing pairs in `all`. The new `delta` is then used to update `all` and compute new pairs using the new rules.

XY-stratified programs can naturally model AI planning problems, since preconditions can simply be expressed by rules, choice can be used to select among applicable actions, and frame axioms can be expressed by XY-stratified

rules that describe changes from the old state to the new state (Brogi et al., 1997).

5 CONCLUSION

The constructs described in this paper are fully operational in the version of \mathcal{LDC}^{++} implemented at UCLA (Zaniolo et al. 1998) and can express and support effectively complex queries and knowledge-based applications. This claim is supported by the applications developed in \mathcal{LDC}^{++} (Tsur, 1991), which include knowledge-based applications, such as ontology-based cooperation of heterogeneous agents in heterogeneous information systems (Tsur, 1991), and datamining applications (Shen et al., 1996, Bonchi et al., 1999). Furthermore, these constructs have a declarative formal semantics based on the concept of total stable models and can therefore be used to characterize other computational notions, such as AI planning (Brogi et al., 1997) and active database rules (Zaniolo, 1997). Finally, using these constructs, many elegant greedy algorithms that were not easily expressed in Prolog can be expressed in \mathcal{LDC}^{++} , using rules that provide a natural logic-based abstraction for these algorithms (Greco and Zaniolo, 1998).

The concepts of choice, XY-stratification and aggregates can be viewed as syntactic devices to harness the power of stable models in order to achieve user-friendliness and efficient computation. XY-stratification is used to express *deterministic* computations, including the computation of well-founded models. Nondeterministic computations can instead be expressed by the choice construct, which also provides a simple characterization of monotonic aggregates (defined using only early return rules) and nonmonotonic aggregates (those using final return rules). All these constructs can be freely mixed in programs, provided that negation and nonmonotonic aggregates are used in stratified or XY-stratified programs. Programs that obey this condition always have total stable models computable in polynomial time, but also have clear intuitive meaning that can be easily mastered by users, who need not understand the subtleties of stable models.

Many of these advantages are unique to \mathcal{LDC}^{++} ; in particular, the well-founded model semantics used in several systems (Ramakrishnan et al., 1993, Rao et al., 1997) is not conducive to nondeterministic computations or user-defined aggregates. On the other hand, other form of nondeterminism, such as disjunctive logic programs, are prone to computational problems, whereas \mathcal{LDC}^{++} programs assure efficient polynomial-time execution. In fact, we have implemented user-defined aggregates in a commercial database system (including monotonic aggregates in recursive queries) and achieved performance levels comparable to those of the aggregates built in by the vendor (Wang and Zaniolo, 2000).

Acknowledgements

The authors would like to thank the referees for the improvements they have suggested.

References

- Abiteboul, S., Hull, R. and Vianu, V. (1995). *Foundations of Databases*. Addison-Wesley, Reading, MA, 1995.
- Baudinet, M., Chomicki, J., and Wolper, P. (1994). *Temporal Deductive Databases*, Chapter 13 of Temporal Databases: Theory, Design, and Implementation, A. Tansel et al. (eds), pp. 294-320, Benjamin/Cummings, 1994.
- Brogi, A., Subrahmanian, V. S. and Zaniolo, C. (1997). The Logic of Totally and Partially Ordered Plans: A Deductive Database Approach, *Annals of Mathematics and Artificial Intelligence* 19(1-2): 27-58 (1997).
- Eiter, T. et al. (2000). Declarative Problem-Solving Using the DLV System, In Minker, J., editor, *Logic-Based Artificial Intelligence*, pages 79-103, Kluwer Academic Publishers, Norwell, Massachusetts, 02061.
- Bonchi, F. et al. (1999). "Applications of $\mathcal{CDL}++$ to Datamining: A Classification-Based Methodology for Planning Audit Strategies in Fraud Detection", *Proc. Fifth ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining, KDD'99* 175-184, ACM, 1999.
- Chimenti, D. et al. (1990). The LDL System Prototype. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), 76-90 (1990).
- Finkelstein, S. J., et al.(1996) Expressing Recursive Queries in SQL, ISO WG3 report X3H2-96-075, March 1996.
- Hellerstein, J. M., Haas, P.J. and Wang, H.J. (1997). "Online Aggregation". *Proc. ACM SIGMOD Int. Conference on Management of Data*, 171-182, ACM, 1997.
- Ganguly, S., Greco, S. and Zaniolo, C. (1995). "Extrema Predicates in Deductive Databases," *JCSS* 51(2): 244-259 (1995)
- Gelfond, M. and Lifschitz, V. (1988). The Stable Model Semantics for Logic Programming. *Proc. Joint International Conference and Symposium on Logic Programming*, R. A. Kowalski and K. A. Bowen, eds., pp. 1070-1080, MIT Press, 1988.
- Giannotti, F. et al. (1981). Non-Determinism in Deductive Databases. *DOOD'91*, C. Delobel, M. Kifer, Y. Masunaga (Eds.), pp. 129-146, Springer, 1991.
- Giannotti, F., et al. (1999). On the Effective Semantics of Nondeterministic, Nonmonotonic, Temporal Logic Databases, *Proc. 12th Int. Workshop, Computer Science Logic*, pp. 58-72, LNCS Vol. 1584, Springer, 1999.
- Giannotti, F., Pedreschi, D. and Zaniolo, C. (2000). "Semantics and Expressive Power of Non-Deterministic Constructs in Deductive Databases," *Journal of Computer and System Sciences*, to appear.
- Greco, S. and Zaniolo, C. (1998). Greedy Algorithms in Datalog with Choice and Negation, *Proc. 1998 Joint Int. Conference & Symposium on Logic Programming, JCLSP'98*, pp. 294-309, MIT Press, 1998.
- Kemp, D., Ramamohanarao, K., and Stuckey, P. (1995). ELS Programs and the Efficient Evaluation of Non-Stratified Programs by Transformation to ELS. In *Proc. Int. Conf. on Deductive and Object-Oriented Databases: DOOD'95*, T. W. Ling, A. O. Mendelzon, L. Vieille (Eds.): pp. 91-108, Springer, 1995.

- Kemp, D. and Ramamohanarao, K. (1998). Efficient Recursive Aggregation and Negation in Deductive Databases. *TKDE* 10(5): 727-745 (1998).
- Krishnamurthy, R., Naqvi, S. (1988) Non-Deterministic Choice in Datalog. *Proc. 3rd Int. Conf. on Data and Knowledge Bases*, pp. 416-424, Morgan Kaufmann, Los Altos (1988).
- Minker, J. (1996). Logic and Databases: A 20 Year Retrospective. *Proc. International Workshop on Logic in Databases (LID'96)*, D. Pedreschi and C. Zaniolo (eds.), pp. 5-52, Springer-Verlag, 1996.
- Przymusinski, T.C. (1998). On the Declarative and Procedural Semantics of Stratified Deductive Databases. In J. Minker (ed.), *Foundations of Deductive Databases and Logic Programming*, pp. 193-216, Morgan Kaufman, San Francisco, CA, 1988.
- Lausen, G., Ludaescher, B. and May, W. (1998). On Logical Foundations of Active Databases, *In Logics for Databases and Information Systems*, J. Chomicki and G. Saake (Eds.), pp. 389-422 Kluwer Academic Publishers, pp. 375-398, 1998.
- Rao, P., et al. (1997). XSB: A System for Efficiently Computing WFS. *Proc. Fourth Int. Conference on Logic Programming and Non-Monotonic Reasoning, LPNMR'97*, J. Dix, U. Furbach, A. Nerode (Eds.), pp. 431-441, Springer 1997
- Ramakrishnan, R., et al. (1993). Implementation of the CORAL Deductive Database System. *Proc. International ACM SIGMOD Conference on Management of Data*, pp. 167-176, 1993.
- Ramakrishnan, R., and Ullman J.D. (1995). A survey of deductive database systems. *JLP*, 23(2): 125-149 (1995)
- Ross, K.A. (1994). Modular Stratification and Magic Sets for Datalog Programs with Negation. *Journal of ACM* 41(6):1216-1266, 1994.
- Ross, K.A. and Sagiv, Y. (1997). "Monotonic Aggregation in Deductive Database", *JCSS*, 54(1), 79-97 (1997).
- Shen, W., et al. (1996). Metaqueries for Data Mining, Chapter 15 of *Advances in Knowledge Discovery and Data Mining*, U. M. Fayyad et al (eds.), pp. 201-217, MIT Press, 1996.
- Schlipf, J.S. (1992). A Survey of Complexity and Undecidability Results in Logic Programming, *Proc. Workshop on Structural Complexity and Recursion-Theoretic Methods in Logic Programming*, 1993, pp.143-164
- Vaghani, J., et .al (1994) The Aditi Deductive Database System. *The VLDB Journal*, 3(2), pp. 245-288 (1994).
- Van Gelder, A., Ross, K.A. and Schlipf, J.S. (1990). The Well-Founded Semantics for General Logic Programs. *Journal of ACM* 38:620-650, 1991.
- Van Gelder, A. (1990). Foundations of Aggregations in Deductive Databases *Proc. of Int. Conf. On Deductive and Object-Oriented databases, DOOD'93*, S. Ceri, K. Tanaka, S. Tsur (Eds.), pp. 13-34, Springer, 1993.
- Tsur, S. (1991). Deductive Databases in Action, *Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages*, pp. 142-154, 1991.

- Zaniolo, C., Arni, N. Ong, K. (1993). Negation and Aggregates in Recursive Rules: the $\mathcal{L}\mathcal{DL}++$ Approach. *DOOD'93*, S. Ceri, K. Tanaka, S. Tsur (Eds.), pp. 204-221, Springer, 1993.
- Wang, H. and Zaniolo, C. (2000). User-Defined Aggregates in Object-Relational Database Systems. *International Conference on Database Engineering*, pp. 111-121, IEEE Press, 2000.
- Zaniolo, C. and Wang, H. (1999). Logic-Based User-Defined Aggregates for the Next Generation of Database Systems. In *The Logic Programming Paradigm: Current Trends and Future Directions*. K.R. Apt, V. Marek, M. Truszczyński, D.S. Warren (eds.), Springer Verlag, pp. 401-424, 1999.
- Zaniolo, et al. (1997) *Advanced Database Systems*, Morgan Kaufmann Publishers, 1997.
- Zaniolo, C. (1997). The Nonmonotonic Semantics of Active Rules in Deductive Databases. *DOOD 1997*, F. Bry, R. Ramakrishnan, K. Ramamohanarao (Eds.) pp. 265-282, Springer, 1997.
- Zaniolo, C. et al. (1998) $\mathcal{L}\mathcal{DL}++$ Documentation and Web Demo, 1998: <http://www.cs.ucla.edu/ldl>

xv

**APPLICATIONS OF
THEOREM PROVING AND
LOGIC PROGRAMMING**

Chapter 23

TOWARDS A MECHANICALLY CHECKED THEORY OF COMPUTATION

The ACL2 Project

J Strother Moore

*Department of Computer Sciences,
University of Texas, Austin, TX 78712
moore@cs.utexas.edu.*

Abstract

Formal mathematical logic is ideally suited to describing computational processes. We discuss the use of one particular mechanized mathematical logic, namely ACL2 (A Computational Logic for Applicative Common Lisp) to model computational problems and to prove theorems about such models. After a few elementary examples, we explain how computational artifacts are formalized in ACL2 and we summarize the main industrial applications of ACL2 as of 1999.

Keywords: Automatic theorem proving, hardware verification, software verification

1 PRELUDE

In 1961, John McCarthy first presented the seminal paper “A Basis for a Mathematical Theory of Computation” (see (McCarthy, 1963)). In that paper McCarthy defined the class of computable functions on some set of base functions. He discussed other fundamental issues, such as the role of non-computable functions, quantification, functionals and what we now call abstract data types. He introduced the notion of “recursion induction.” He used his formal system to prove many now classic elementary theorems in his emerging theory of computation, including distributivity of Peano multiplication over Peano addition and the associativity of the list concatenation function. He also clearly laid the basis for the formal establishment that two computational processes are equivalent or that they stand in some other precisely defined relationship.

In a companion paper, (McCarthy, 1962), written a year later, McCarthy wrote

Instead of debugging a program, one should prove that it meets its specifications, and this proof should be checked by a computer program. For this to be possible, formal systems are required in which it is easy to write proofs. There is a good prospect of doing this, because we can require the computer to do much more work in checking each step than a human is willing to do. Therefore, the steps can be bigger than with present formal systems.

In the three and a half decades since these papers were published, McCarthy's vision has inspired many of us. In 1971, Bob Boyer and I wrote a computer program, called the Pure Lisp Theorem Prover (see (Boyer and Moore, 1975)), capable of automatically finding many of the example proofs exhibited in (McCarthy, 1963). That early computer program evolved over the next decade to become the Boyer-Moore theorem prover, NQTHM (Boyer and Moore, 1979; Boyer and Moore, 1997). Then Matt Kaufmann and I, with the advice of Boyer, collaborated to produce ACL2 (Kaufmann and Moore, 1997; Kaufmann and Moore, 1999a).

While still undeniably a research tool, ACL2 finds commercial applications in the mechanical checking of formally stated properties of microprocessors, microcode, and other such artifacts. While not nearly as ambitious as McCarthy's 1961 vision of a theory of computation, ACL2 is a reification of it. In this paper I discuss ACL2 and its use as a mechanized theory of computation.

2 INTRODUCTION

Microprocessors are changing our world. They are becoming ubiquitous, popping up in places both expected (e.g., computers, airplanes, missiles, televisions, automobiles) and unexpected (e.g., toothbrushes, bicycles, and greeting cards).

In a very important sense, microprocessors are just state machines: at every tick of the clock the machine changes from one state to the next according to a precisely specified mathematical "state transition" function. States are discrete. They are typically composed of "atomic" objects such as integers and words, glued together into more complex structures such as sequences, tables, trees, etc. For the simplest machines, the state transition function takes just one argument, the current state, and produces the next state. In general, the function may take additional arguments representing some kind of "input" impinging on the machine "from outside" and may produce, in addition to a new state, some "output." For our purposes, machines may have either a finite or an infinite number of distinct states.

State machines are well-studied mathematical abstractions. There are many different kinds: finite state automata, Moore machines, Mealy machines, Turing machines, etc. Such machines may be described formally by describing the legal states and the state transition function. Such descriptions are typically written in some logical language. Indeed, mathematical logic, especially a logic supporting recursive definition and inductive construction, is almost ideally suited to describing state machines formally. While simple machines might be

described by relatively simple languages, as used in automata theory, the state machines implemented by microprocessors can, for all practical purposes, only be described in a general-purpose mathematical logic.

This places us at an extremely interesting moment in the history of mathematical logic: applications of great practical importance are plentiful. Most of mathematics has long enjoyed – or suffered from – this convergence of theory and practice; but until the invention of the digital computer, mathematical logic was almost exclusively a tool for the study of logic itself or of philosophy. This study led to a deep understanding of the power (and limitations) of formalism. Indeed, computing science grew, in part, out of the study of formal systems. But classical logic is too “black and white” to serve as a descriptive and predictive framework for “real world” phenomena. This, of course, motivated many interesting developments in logic, e.g., non-monotonic logic and fuzzy logic. But, compared to say linear algebra or partial differential equations, classical logic remained a tool without many direct, commercially interesting applications. The complexity and ubiquity of computing in general, and microprocessors in particular, has changed that. Microprocessors are intended to be discrete state machines. They realize this intention so faithfully and so directly that mathematical logic is the language of choice for writing down their descriptions. Indeed, until one descends in the design hierarchy to the point at which gates and wires are laid out, microprocessors are so accurately described by logical formulas that there is often confusion over whether by “microprocessor” one means the mathematical abstraction or the physical artifact. In this paper, I take the view that a microprocessor is a physical artifact that is modeled by a set of logical formulas. But quite often in microprocessor design there are levels of abstraction above that low-level model, i.e., at the instruction-set level, at which there is no distinction between the machine and the set of formulas describing it: the artifact of interest is the abstract state machine, not some imperfectly realized physical object.

That microprocessors can be accurately described by a classical formal logic is not the end of the story. The complexity of today’s microprocessor designs is such that mathematical proof is the only way to gain confidence that a design enjoys some desired property.

Therefore, there is a growing community of applied logicians: people trained in creating formal models of microprocessors and producing formal proofs of theorems about those models. Typical models consist of many definitions of “new” concepts, such as “the registers upon which instruction *i* depends” and whether “processor *i* has the exclusive write permission for address *a*. These concepts are combined to produce descriptions of machines and their desired properties. Machine models can require hundreds of pages to write down fully.

Proving theorems about these models can be a daunting task. For this reason, many applied logicians prefer to use mechanized tools (Gordon and Melham, 1993; Owre et al., 1992; McCune, 1994) to develop and/or check their proofs. ACL2 (Kaufmann et al., 2000) is one such tool.

This paper is addressed primarily to the student of formal methods. The main message is: mechanically checked proofs about industrial designs are possible. But there are several other messages. One is that we prove *formulas* in

some formal system and we are inevitably faced with expressing our models and their alleged properties as formulas. Another is that proofs are not automatic; the user must provide some form of guidance. This paper illustrates how these issues are dealt with in the ACL2 system. The paper is not intended as an introduction to formal methods nor as a tutorial introduction to the use of the ACL2 system.

3 THE ACL2 LOGIC

"ACL2" is both the name of a mathematical logic and the mechanical theorem proving system supporting it. It stands for A Computational Logic for Applicative Common Lisp. The ACL2 logic is a first-order essentially quantifier-free logic of recursive functions and inductively defined objects, developed by Matt Kaufmann and the author. It is modeled closely on the Boyer-Moore NQTHM logic (Boyer and Moore, 1997) but describes an applicative subset of Common Lisp (Steele, 1990) rather than a "home-grown" Lisp.

Our style of formalizing ACL2 logically follows that of (Shoenfield, 1967). We introduce quantifier-free first order logic with equality between terms. The syntax of terms is that of Common Lisp. We then add axioms describing five Common Lisp data types: complex rationals (with the rationals and the integers as subsets), characters, strings, symbols (with packages), and conses. We add axioms defining about 170 Common Lisp functions, including such basic list processing functions as `if`, `equal`, `cons`, `car`, `cdr`, `consp`, and `atom`, as well as the primitives for manipulating the other data types, such as `+`, `-`, `*`, and `/`; `code-char` and `char-code` (mapping between naturals and characters); `coerce` (mapping between lists of characters and strings); and `intern` and `symbol-name` (mapping between strings and symbols). We define the function `acl2-count` to be a measure of the size of an object. The size of a `cons`-tree is the number of `cons` nodes in it plus the sizes of the leaves.

Following (Goodstein, 1964), we identify a certain set of objects, constructed from conses and natural numbers, with the ordinals below $\varepsilon_0 = \omega^{\omega^{\omega}}$. These objects are recognized by the function `e0-ordinalp`. We define `e0-ord-<` to test when one ordinal is less than another. In a suitable set theory one could prove that `e0-ord-<` is well-founded. This theorem cannot be stated in the weak logic of ACL2, but its validity is fundamental to the soundness of the formal logic.

We allow the definitional extension of the logic by the addition of new recursion equations, under certain syntactic conditions, provided certain syntactically specified "measure conjectures" can be proved. These conjectures establish that some ordinal measure of the arguments to the function decreases according to `e0-ord-<` in the recursive calls. Consider

```
(defun append (x y)
  (if (consp x)
      (cons (car x) (append (cdr x) y))
      y)).
```

This definition is admitted using the measure (`(acl2-count x)`). There are two measure conjectures.

```
(e0-ordinalp (acl2-count x))

(implies (consp x)
         (e0-ord-< (acl2-count (cdr x))
                     (acl2-count x)))
```

Both are theorems.

Only terminating recursions are allowed under the definitional principle. This preserves the soundness of the logic, which otherwise would be imperiled by “definitions” such as (`(defun f (x) (+ 1 (f x)))`). In our simple first-order logical setting, nonterminating “definitions” can give rise not just to “ambiguous” functions (to use the terminology of (McCarthy, 1963)) but to inconsistency.

We also allow inductions up to ϵ_0 in a manner exactly dual to the definitional principle. The definitional principle and the induction rule of inference are taken virtually unchanged from NQTHM (Boyer and Moore, 1997).

The logic is somewhat richer than described above. The syntax can be extended by the use of “macros.” The logic includes the “encapsulation” mechanism, which allows the introduction of undefined function symbols axiomatized to satisfy certain witnessed constraints. A derived rule of inference, called “functional instantiation,” allows theorems about constrained functions to be instantiated in a second-order way with function symbols known to satisfy the constraints (Boyer et al., 1991; Kaufmann and Moore, 1999b). The logic also permits the introduction of “Skolem functions,” each of which returns an unspecified object satisfying a given formula, provided such an object exists. See `defchoose` in (Kaufmann and Moore, 1999a). This is used to provide the power of full first-order quantification (without providing the expressive convenience or implementation challenges of the quantifiers themselves); see `defun-sk` in (Kaufmann and Moore, 1999a).

A subset of ACL2 is executable in the sense that “most” ground expressions may be reduced to primitive constants by the systematic use of instantiation and substitution of equals for equals. (The subset excludes constrained functions.) Furthermore, ACL2 is consistent with Common Lisp in that ground instances of certain expressions may be computed by simply evaluating the instances in any compliant Common Lisp (subject to resource limitation). To make this precise we introduce the notion of the “intended domain” of Common Lisp functions. The Common Lisp standard (Steele, 1990) does not specify the behavior of functions outside their intended domains (but ACL2 does). We provide a means by which one can establish that an expression’s evaluation stays within the intended domains of the functions involved. This is the so-called “guard” mechanism of (Kaufmann and Moore, 1997). If a function’s “guard conjectures” are theorems, appropriate calls of the function can be evaluated by direct appeal to Common Lisp.

Except for the maxim that the devil is in the details, we do not regard ACL2 as particularly remarkable, either as a mathematical logic or as a functional programming language.

4 THE ACL2 THEOREM PROVER

Like its logic, ACL2's theorem prover was inspired by NQTHM's but has been considerably improved. Like NQTHM, ACL2 supports fast type reasoning, complete propositional calculus and equality, a built-in linear arithmetic decision procedure, backwards chaining, conditional rewriting, verified meta-theoretic simplifiers, a variety of heuristics designed to generalize a conjecture before resorting to mathematical induction, and sophisticated heuristics for inventing often-appropriate inductive schemas. All of these proof techniques are closely integrated and driven by a database of previously proved theorems. By the careful selection of lemmas to prove first, the informed user of either NQTHM or ACL2 can "program" the system to find a given proof.

In addition, ACL2 has integrated many other proof techniques, including faster propositional calculus procedures than NQTHM's, the mixing of rewriting with the binary decision diagrams of (Bryant, 1992) for canonicalizing propositional expressions (see (Moore, 1994) and the documentation of the BDD hint in (Kaufmann and Moore, 1999a)), forward chaining, many devices for controlling the application of previously proved lemmas, including meta-theoretic simplifiers, and, perhaps most importantly, the implementation of congruence-based rewriting (see *congruence* in (Kaufmann and Moore, 1999a)). The last permits the user to introduce new equivalence relations and cause ACL2 to use theorems about those relations as rewrite rules in appropriate contexts defined by congruences established by other lemmas.

This is especially useful in a logic such as ours where abstract objects are represented concretely. Consider finite sets represented as lists and let `(union a b)` be defined to be the union operator. Because of duplication and ordering, it is not in general the case that `(union a b)` is equal to `(union b a)`. Therefore, `union` expressions cannot in general be commuted. One can define the relation `set-equal`, testing whether two lists have the same elements while ignoring duplications and order. One can then prove `(set-equal (union a b) (union b a))`. However, this lemma is not very useful, by itself, because it cannot be used as a rewrite rule to simplify expressions: it is not an equality! But ACL2 takes special notice of such rules if `set-equal` has been proved to be an equivalence relation. Furthermore, congruence lemmas can be proved, establishing, for example, that `set-equal` is a congruence (modulo iff) for `mem`, the set-membership relation: `(implies (set-equal x y) (iff (mem e x) (mem e y)))`. ACL2 can be commanded to prove this congruence lemma by issuing the command `(defcong set-equal iff (mem e x) 2)`. If the proof is successful, the "new" theorem informs the ACL2 rewriter as follows. Suppose the rewriter encounters an expression of the form `(mem δ (union α β))` in a context in which the rewriter is trying to maintain propositional equivalence (`iff`). That is, suppose the `mem` expression occurs as a hypothesis or conclusion of a formula to be proved, or as the test of an `if`. The congruence lemma tells

us that the propositional value of the `mem` expression is unchanged even if we replace the second argument by some `set-equal` value. Therefore, when rewriting the second argument of the `mem` expression, $(\text{union } \alpha \beta)$, we may use `use (set-equal (union a b) (union b a))` as a rewrite rule, as though it were $(\text{union } a b) = (\text{union } b a)$. (Heuristics determine whether ACL2 will use such a rule to permute the arguments of a function.) We illustrate congruence-based rewriting here.

Readers wishing more background on the mechanization of ACL2 are urged to see (Boyer and Moore, 1979) and (Boyer and Moore, 1997), where the earlier NQTHM is described, plus the extensive ACL2 user's manual. The last is available in hypertext format on the internet, see (Kaufmann and Moore, 1999a).

5 ELEMENTARY EXAMPLES

We now turn to the main topic of this paper, which is the demonstration of ACL2 as a mechanized theory of computation. We do not show ACL2's proofs, but rather what the user must say to ACL2 to lead it to those proofs. We start with several examples from (McCarthy, 1963). Our basic position is that a mechanized logic of computation ought allow the user simply to observe that elementary formulas are theorems; the underlying theorem prover should do the computation necessary to fill in the proof. The presentation below contrasts McCarthy's observations with ACL2's ability to follow them. Unless otherwise indicated, ACL2 fills in the proofs without help.

McCarthy defines the list concatenation function as the binary operator “`*`” with the notation:

$$x * y = [\text{null}[x] \rightarrow y; T \rightarrow \text{cons}[\text{car}[x]; \text{cdr}[x] * y]].$$

Since “`*`” is already defined in our system (to be multiplication), we change the name to “`jmc::*`,” i.e., to the symbol with the name “`*`” in a new package named `jmc`. This symbol can be written simply as `*` if we select `jmc` the “current package.” By importing into that package the other Lisp symbols we need, we can then write:

```
(defun * (x y)
  (if (null x)
      y
      (cons (car x) (* (cdr x) y))))
```

As written above, this definition is not admitted by ACL2. The reason is that no measure is supplied and so the system guesses the measure (`acl2-count x`). But that measure does not decrease. For example, if `x` is `0`, then `(null x)` is not true and hence the definition recurs to `(cdr 0)`. In ACL2, `cdr` returns `nil` when applied to non-`cons` objects. But both `0` and `nil` are of size 0, as measured by `acl2-count`. Hence the guessed measure does not decrease. (McCarthy deals with problem by implicitly limiting the function to lists that terminate in `nil`.)

ACL2 can admit the function above, but the user must supply a measure that gives nil a smaller size than all other objects.

```
(defun * (x y)
  (declare (xargs :measure
                  (if (null x) 0 (+ 1 (acl2-count x)))))
  (if (null x)
      y
      (cons (car x) (* (cdr x) y))))
```

With this measure, the definition is admitted. Note that $(* '(1 2 3) '(4 5 6))$ is $'(1 2 3 4 5 6)$, as expected. Perhaps unexpectedly, $(* 0 '(4 5 6))$ is $'(nil 4 5 6)$, because 0 is not null and has car and cdr nil.

McCarthy next proves $[x * y] * z = x * [y * z]$ by recursion induction. ACL2 proves this without help from the user, in response to the command

```
(defthm associativity-of-
  (equal (* (* x y) z) (* x (* y z))))
```

The proof is by induction on x. When the proof is completed, ACL2 builds this lemma into its database of rewrite rules. Henceforth, the simplifier will right-associate *-nests.

McCarthy then proves $\text{NIL} * x = x$ and $x * \text{NIL} = x$, the latter by induction. In ACL2, the former is trivially proved and the latter is not valid (consider $x = 0$). But the following is a theorem

```
(defthm *-left-id
  (implies (true-listp x)
            (equal (* x nil) x)))
```

which ACL2 can prove without assistance from the user. The predicate (true-listp x) is true if x is a list terminated by nil. The proof of *-left-id is by induction on x.

Next, McCarthy defines

$$\text{reverse}[x] = [\text{null}[x] \rightarrow \text{NIL}; T \rightarrow \text{reverse}[\text{cdr}[x]] * \text{cons}[\text{car}[x]; \text{NIL}]].$$

The analogous

```
(defun reverse (x)
  (declare (xargs :measure
                  (if (null x) 0 (+ 1 (acl2-count x)))))
  (if (null x)
      nil
      (* (reverse (cdr x)) (cons (car x) nil))))
```

is easily admitted by ACL2, but note that the user must supply the same special measure justifying this recursive scheme.

Finally, McCarthy states without proof two theorems which are “not difficult to prove by recursion induction.”

$$\text{reverse}[x * y] = \text{reverse}[y] * \text{reverse}[x],$$

and

$$\text{reverse}[\text{reverse}[x]] = x.$$

The McCarthy-style proof of the first is by recursion induction with a “base case” of $x = \text{NIL}$. The conjecture simplifies to $\text{reverse}[y] = \text{reverse}[y] * \text{NIL}$, which is proved by the left-identity lemma above. However, when ACL2 tries to construct this proof, it fails to apply `*-left-id` because it cannot relieve the hypothesis (`true-listp (reverse y)`) of our version of `*-left-id`. This fact requires induction to prove and ACL2 does not use induction to relieve hypotheses during backchaining. Therefore the user must help ACL2 by proving

```
(defthm true-listp-reverse
  (true-listp (reverse x)))
```

first. Then ACL2 “automatically” proves

```
(defthm reverse-*
  (equal (reverse (* x y)) (* (reverse y) (reverse x)))).
```

McCarthy’s $[\text{reverse}[\text{reverse}[x]] = x]$ is not valid in our system; the restriction to lists ending in `nil` must be made explicit. However, the following is proved automatically

```
(defthm reverse-reverse
  (implies (true-listp x)
            (equal (reverse (reverse x)) x)))
```

and uses `reverse-*` in the proof. It is perhaps worth noting that `reverse-reverse` can also be proved by ACL2 immediately after the introduction of the definition of `reverse`, i.e., without any user-suggested lemmas about `reverse`. The prover’s heuristics lead it to the conjecture that `(reverse (* x (cons a nil)))` is `(cons a (reverse x))`, instead of the more general `reverse-*`. It proves this conjecture by a second induction.

McCarthy’s classic paper also deals with proofs in Peano arithmetic. These theorems can also be proved by ACL2.

ACL2’s performance on problems of this scale is essentially the same as NQTHM’s, which in turn is essentially the same as the Pure Lisp Theorem Prover of 1973. (It should be noted that the earliest version of our prover

did not do termination proofs of function definitions and thus could be made inconsistent by user-added “definitional” axioms.)

6 MERGE SORT

Here is a problem with which ACL2 has little difficulty but NQTHM has much more. The ACL2 solution exploits ACL2’s support for equivalence relations and congruence based rewriting. We will define a merge sort algorithm and prove that it returns an ordered permutation of its input. The algorithm is defined as follows:

```
(defun merge-sort (x)
  (if (consp x)
      (if (consp (cdr x))
          (merge (merge-sort (evens x))
                 (merge-sort (odds x)))
          x)
      x))
```

where we will define (`evens x`) and (`odds x`) to return those elements of the linear list `x` in even-numbered or odd-numbered positions, respectively, and we will define `merge` so that it merges two ordered lists.

Our goal is to prove both (`orderedp (merge-sort x)`) and (`perm (merge-sort x) x`). That is, the output of `merge-sort` is ordered and is a permutation of the input. With an ideal theorem prover we should only have to define the concepts involved and state the two theorems. With ACL2 we must do more.

6.1 ORDERED PERMUTATIONS

Here is the definition of `orderedp`.

```
(defun orderedp (x)
  (if (consp x)
      (if (consp (cdr x))
          (and (<= (car x) (cadr x))
               (orderedp (cdr x)))
          t)
      t))
```

For example, (`orderedp '(1 2 2 7 13)`) is `t` but (`orderedp '(1 2 7 2 13)`) is `nil`. ACL2 admits this definition without help since its standard measure, (`acl2-count x`), decreases.

Below is the definition of when one list is a permutation of another. The basic idea is that two lists are permutations if and only if we can successively find and remove equal elements from the two lists until both lists are empty.

```
(defun perm (x y)
  (if (consp x)
      (and (mem (car x) y)
            (perm (cdr x) (del (car x) y)))
      (not (consp y))))
```

where (`mem e x`) determines whether `e` is a member of `x` and (`del e x`) deletes the first occurrence of `e` from `x`. Their easily admitted definitions are:

```
(defun mem (e x)
  (if (consp x)
      (or (equal e (car x))
          (mem e (cdr x)))
      nil))

(defun del (e x)
  (if (consp x)
      (if (equal e (car x))
          (cdr x)
          (cons (car x) (del e (cdr x))))
      nil))
```

`Perm` is admitted easily too, using (`acl2-count x`) as the measure. ACL2 can be made to prove that `perm` is an equivalence relation . That is

```
(defthm perm-is-an-equivalence
  (and (perm x x)
       (implies (perm x y) (perm y x))
       (implies (and (perm x y) (perm y z)) (perm x z)))
  :rule-classes :equivalence).
```

This may be abbreviated as (`defequiv perm`) using a macro that expands as above. We do not show the lemmas here. NQTHM can, of course, be made to prove this theorem, but NQTHM does not subsequently make any special use of it.

Once `perm` is known to be an equivalence relation, ACL2 can easily prove the following congruence rule for `cons`: (`implies (perm x y) (perm (cons e x) (cons e y))`). This may be abbreviated as the command:

```
(defcong perm perm (cons x y) 2)
```

and can be thought of as telling ACL2 that if one is rewriting a `cons` expression while maintaining the equivalence relation `perm`, it is permitted to rewrite the second argument while maintaining `perm`. That is, theorems about `perm` may be used as rewrite rules in that context. ACL2 can also easily prove

```
(defcong perm perm (append x y) 1)
(defcong perm perm (append x y) 2)
```

which mean that when rewriting `append` expressions while maintaining the `perm` equivalence relation, it is permitted to rewrite both arguments maintaining `perm`. Finally, ACL2 can easily prove

```
(defthm append-alt-def
  (perm (append x y)
    (if (consp y)
        (cons (car y) (append x (cdr y)))
        x))

:rule-classes
(:definition :controller-alist
  ((acl2::binary-append nil t))))
```

which says that `(append x y)` could be defined symmetrically if just `perm` equivalence is required. The `:rule-classes` hint tells ACL2 to use this “equation” as an alternative definition of `append`.

The definitions and lemmas mentioned in this section may be collected together into a “book” and incorporated into subsequent ACL2 sessions by the user. The ability to construct the database of relevant theorems incrementally is a very powerful feature of ACL2 because it allows users to build on the work of others and encourages the development of general collections of rules. We will call the book representing the work of this section “`orderedp-and-perm`”. It contains four definitions (`orderedp`, `mem`, `del`, and `perm`) and the five theorems noted above. The proof of `(defequiv perm)` requires the proof of seven lemmas which are not “exported” from the book. To construct all these proofs, ACL2 requires about 2.5 seconds (on an IBM Thinkpad laptop with a 330 MHz Pentium II and 256 MB of RAM). Henceforth we assume we have the material in the “`orderedp-and-perm`” book. We will now solve the merge sort problem while showing every lemma and hint the user must give to ACL2.

6.2 THE DEFINITION OF MERGE-SORT

We define the even and odd elements of a list as follows:

```
(defun evens (x)
  (if (consp x)
      (cons (car x) (evens (cddr x)))
      nil))

(defun odds (x) (evens (cdr x)))
```

Their admission is trivial. To merge two lists we define

```
(defun merge (x y)
  (declare (xargs :measure
                  (+ (acl2-count x) (acl2-count y))))
  (if (consp x)
      (if (consp y)
          (if (< (car x) (car y))
              (cons (car x) (merge (cdr x) y))
              (cons (car y) (merge x (cdr y))))
          x)
      y))
```

Note that it is necessary to tell ACL2 a suitable measure.

Our next goal is to admit the definition of `merge-sort`. It recurs on the `evens` and `odds` of its argument, when that argument is a list of at least two elements. We must prove that the sizes of these two lists are smaller. That can be done with the following lemma.

```
(defthm acl2-count-evens
  (and (implies (and (consp x)
                      (consp (cdr x)))
                 (< (acl2-count (evens x))
                     (acl2-count x)))
               (<= (acl2-count (evens x))
                   (acl2-count x)))
  :rule-classes :linear)
```

This lemma, which is really two interesting mathematical facts, says that the size of `(evens x)` is weakly smaller than of `x` and the inequality is strict when `x` has more than one element. These facts are proved without further ado. But the `defthm` command above has a third interesting aspect. It directs the system to build these facts into the linear arithmetic package. They are used automatically during the admission of `merge-sort` (above).

6.3 THE CORRECTNESS PROOF

The system is now able to prove

```
(defthm ordereddp-merge-sort
  (ordereddp (merge-sort x)))
```

without further help from the user. The system's heuristics lead it to conjecture that `(merge x y)` is ordered if `x` and `y` are ordered, and to prove this conjecture with a secondary induction. The reader is invited to construct the proof of the `merge` lemma by hand. Most readers who try will probably feel that their hand proof is messy and boring, with various cases having to be considered. The fact

that ACL2 does the proof automatically illustrates McCarthy's hope that we can require the computer to do more work than a person is willing to check.

The last goal is that `merge-sort` produces a permutation of its input. We contribute to the proof in a major way by making two key observations. First, `merge` is `append` (modulo the `perm` sense of equivalence).

```
(defthm perm-merge
  (perm (merge x y) (append x y))

  :hints (( "Goal" :induct (merge x y))))
```

Note that we have to tell ACL2 to induct the way `merge` recurses (instead of the way `append` recurses). Once proved, this theorem allows ACL2 to replace instances of `(merge x y)` by instances of `(append x y)` in contexts in which `perm` equivalence is to be maintained.

The second key observation is that appending the `evens` and `odds` of `x` produces `x` (modulo `perm`).

```
(defthm perm-evens-odds
  (perm (append (evens x) (odds (cdr x)))
        x))
```

This too is used as a rewrite rule in appropriate contexts.

With these two observations in the database, the final goal is trivial by induction on `x`.

```
(defthm perm-merge-sort
  (perm (merge-sort x) x))
```

The time required to process all the events in this section (on the abovementioned IBM laptop) is approximately 2.5 seconds. Hence, the entire `merge-sort` problem is checked in about 5 seconds.

Note that we do not present ACL2 with a formal proof to check. Instead, we present it with something more analogous to the kind of proofs exchanged by humans: a series of observations with occasional hints.

Not counting the development of the lemmas about `orderedp` and `perm` (which are independent of the sorting algorithm to be analyzed), the ideal prover would have required us to make four definitions (`evens`, `odds`, `merge` and `merge-sort`) and then state two theorems (`orderedp-merge-sort` and `perm-merge-sort`). ACL2 required, in addition, that we

- exhibit the measure that decreases in `merge`,
- observe that the size of `evens` decreases, sometimes strictly,
- recommend that these facts be used as linear rules,
- observe that `merge` is `append` (modulo `perm`), and

- observe that `append` of the `evens` and `odds` is the identity (modulo `perm`).

We do not find these requirements onerous. The first three could probably be lifted with some additional heuristics. The last two seem to require genuine mathematical insight. At least ACL2 allows the user to communicate the insight and can make appropriate use of it. Of course, this aspect of ACL2 should remind readers of McCarthy's "Advice Taker" described in (McCarthy, 1959) proposal in which he says "In order for a program to be capable of learning something it must first be capable of being told it."

7 COMPUTING MACHINES

Most serious users of ACL2 use it to model microprocessors and other computing machines. In this section we sketch how this is typically done in ACL2.

7.1 JAVA BYTE CODE

Here is a Java program "implementing" the factorial function.

```
public static int fact(int n){
    if (n>0)
        {return n*fact(n-1);}
    else return 1;
}
```

To be precise, it implements the factorial function for values of `n` between 0 and 12. $13!$ is 6227020800 but the program above returns 1932053504 because the `int` type in Java is allocated only 32 bits. We will ignore this and other important fine points of the semantics of Java in this discussion but will return to it later.

Java is implemented by compiling it into "byte code" for the Java Virtual Machine or JVM (Lindholm and Yellin, 1996). The JVM is a stack based machine. In this section we show a formal model of a "toy JVM", TJVM, that is suggestive of the JVM but small enough to present in some detail. Below we show the compilation of the `fact` program above. In the left column is the byte code for our TJVM. On the right is the JVM code generated by Sun Microsystems' Java compiler.

<code>("fact" (n)</code>	
<code>(load n)</code>	; 0 iload_0
<code>(ifle 8)</code>	; 1 ifle 12
<code>(load n)</code>	; 2 iload_0
<code>(load n)</code>	; 3 iload_0
<code>(push 1)</code>	; 4 iconst_1
<code>(sub)</code>	; 5 isub
<code>(invokestatic "Math" "fact" 1)</code>	; 6 invokestatic ...
<code>(mul)</code>	; 7 imul

```
(xreturn) ; 8 ireturn
(push 1) ; 9 iconst_1
(xreturn) ; 10 ireturn
```

The left-hand column is actually a list constant containing thirteen elements. The first element is the string "fact"; the second is a list containing the single symbol n, the third is a list containing the symbol load followed by the symbol n, etc. This constant will be our representation of a "method declaration" on the TJVM. This method declaration for the "fact" method might be found in the "Math" class on the TJVM. The shallow correspondence between our TJVM and the JVM is obvious. Some of the discrepancies are explained as follows. On the TJVM, methods refer to their local variables by name; on the JVM methods refer to their locals by position. The "8" in the TJVM ifle instruction is an instruction offset by which the program counter is incremented. The corresponding offset on the JVM counts bytes rather than instructions and some JVM instructions take more than one byte. The JVM has typed instructions, e.g., the JVM's iload loads an integer-valued variable on the stack while the TJVM's load loads any value.

Roughly speaking, (load n) pushes the value of n on the stack; (ifle 8) pops one item off the stack and increments the program counter by 8, if the item is less than or equal to 0, and by 1, otherwise; (push 1) pushes the constant 1; (sub) pops two items off the stack and pushes their difference; and (invokestatic "Math" "fact" 1) invokes the method named "fact" in the "Math" class on the topmost object on the stack.

A mechanized theory of computation must allow us to formalize such machines and reason about their behavior mechanically.

7.2 THE FORMAL MODEL

We do this by formalizing the notion of the state of the TJVM. Our TJVM state contains several components, including a call stack of frames, a heap, and a class table listing classes and their method declarations. Here, a frame is itself a structure associated with each method invocation. A frame contains the current program counter, the stack of intermediate results, the invocation's variable bindings, and the byte code of the invoked method. We then define the semantics of each TJVM instruction as a function that takes the instruction, *inst*, and the current TJVM state, *s*, and returns a new state. Below is the semantic function for the TJVM add instruction. We use a macro named *modify* to denote the state obtained from *s* by replacing the indicated components and leaving all other components unchanged.

```
(defun execute-add (inst s)
  (modify s
    :pc (+ 1 (pc (top-frame s)))
    :stack (push (+ (top (pop (stack (top-frame s))))
                    (top (stack (top-frame s)))))
               (pop (pop (stack (top-frame s)))))))
```

The `add` instruction increments the program counter (in the topmost frame of the call stack of `s`) and pops two items off the stack (in the topmost frame of the call stack of `s`) and pushes their sum. In this model, the sum is computed with the ACL2 function `+`, which is true mathematical addition, not the 32-bit 2's complement sum operator used by the JVM.

After defining the semantic function of each TJVM byte code instruction we define the “big switch” `do-inst` which executes a given instruction on a state, we use it to define `step`, which executes the next instruction (i.e., the one at the program counter on the topmost frame of the call stack), on the current state, and finally define `tjvm`, which iteratively steps the current state `n` times.

```
(defun do-inst (inst s)
  (case (op-code inst)
    (add          (execute-add inst s))
    ...
    (invokevirtual (execute-invokevirtual inst s))
    (invokestatic  (execute-invokestatic inst s))
    (getfield      (execute-getfield inst s))
    (putfield      (execute-putfield inst s))
    ...
    (otherwise s)))

(defun step (s)
  (do-inst (next-inst s) s))

(defun tjvm (s n)
  (if (zp n)
      s
      (tjvm (step s) (- n 1))))
```

The function `tjvm` is an ACL2 version of a state machine. This machine is not finite, since we can use it to create arbitrary integers by repeated adds.

7.3 THEOREMS ABOUT TJVM

As befitting a mechanized theory of computation, we can prove theorems mechanically about such state machines. A very useful general theorem about `tjvm` is the following:

```
(defthm tjvm-+
  (implies (and (natp i) (natp j))
            (equal (tjvm s (+ i j))
                   (tjvm (tjvm s i) j))))
```

which allows us to decompose a long run into two shorter runs. This theorem is proved automatically by induction on i . The theorem is quite general, putting no restrictions on (the state) s .

At the opposite extreme is a theorem that can be proved simply by execution. Consider the state to which $tjvm$ below is applied. It is a constant: a state with one frame, poised to push 13 on the empty stack and call our "fact". Suppose $tjvm$ is used to take 124 steps from that state and then consider the topmost item on the stack in the topmost frame of the call stack of the resulting state. The theorem says we will find 6227020800. Here the constant (**Math-class**) is defined to be a class declaration containing the "fact" method declaration shown earlier.

```
(defthm example13
  (equal
   (top
    (stack
     (top-frame
      (tjvm
       (make-state
        (push (make-frame 0 nil nil
                           '((push 13)
                             (invokestatic "Math" "fact" 1)
                             (halt)))
               nil)
        nil (list (Math-class)))
      124))))
   6227020800))
```

This illustrates the executability of ACL2. It is easy to see that this might be a useful feature when dealing with situations like this. We use test executions to experiment with our formal models. An important aspect of this experimentation is to *corroborate* a formal model against an informal "specification" such as the expectations of the design team, a document written in English or some other informal language, or the behavior of some artifact such as a simulator or prototype created by the designers. In addition, if an expression such as that above arises during the course of a more difficult proof, it is convenient to be able to compute its value, just as one might wish to compute, rather than derive by primitive proof steps, the values of $(+ 2 2)$ or $(expt 2 32)$.

Finally, here is another theorem we can prove about $tjvm$. It states that if $s0$ is a TJVM state poised to execute a call of our "fact" method on a natural number n , then the state obtained by executing $s0$ a certain number of steps, namely $(fact-clock n)$, can alternatively be obtained by advancing the program counter of $s0$ by one and pushing $(fact n)$ in place of n on the stack.

```
(defthm fact-is-correct
  (implies
    (and (equal (next-inst s0)
                '(invokestatic "Math" "fact" 1))
         (equal (assoc-equal "Math" (class-table s0))
                (Math-class))
         (equal n (top (stack (top-frame s0))))
         (natp n))
    (equal
      (tjvm s0 (fact-clock n))
      (modify s0
        :pc (+ 1 (pc (top-frame s0)))
        :stack (push (fact n)
                      (pop (stack (top-frame s0))))))))
```

Here, `fact` is defined as

```
(defun fact (n)
  (if (zp n)
      1
      (* n (fact (- n 1))))))
```

The theorem above establishes that the TJVM byte code for our "`fact`" method is correct in the sense that it computes the same thing as the factorial function in ACL2. Furthermore, as a rewrite rule it is extremely useful because it allows us to run `tjvm` past an invocation of "`fact`" in a single logical step, just as though `(invokestatic "Math" "fact" 1)` were a primitive instruction that computed the factorial of the top of the stack, except costing `(fact-clock n)` primitive steps.

For details about the TJVM model, see (Moore, 1999b). For details about how ACL2 is configured to prove such theorems, see (Boyer and Moore, 1996).

8 NONDETERMINISM AND DISTRIBUTED COMPUTATION

Within this basic framework it is possible to formalize machines that are nondeterministic by incorporating into the model an "oracle," typically formalized as a list, that provides arbitrary input for each step. Such input is often used to select between various possible behaviors. If the oracle is unconstrained in some theorem, then nondeterminism is modeled. But hypotheses are sometimes added to constrain the oracle to be "fair" or "responsive" or to have other qualities.

In (Moore, 1999a), a model is given of a set of n machines, each more or less comparable to the TJVM, but operating on a shared memory via a "compare and swap" instruction. Parallelism is modeled by interleaved atomic execution of individual instructions. An oracle determines which processor steps next.

To illustrate the efficacy of the model, we prove a theorem about the execution of a certain non-blocking counter algorithm. The details of the algorithm are unimportant here.

But the proof involved formalizing a different view of the system: that of a single processor in the system. From that perspective, only one processor exists, but the shared part of memory changes "chaotically." Formalizing this required another oracle.

The proof strategy involved proving an invariant about the shared memory in the multiprocessor system, translating that invariant to a monotonicity restriction on the oracle in the uniprocessor view, and then doing a classical uniprocessor code proof (such as that for "fact" above) under the restricted oracle. Of course, the two models of computation have to be formally related if conclusions reached in the uniprocessor view are to be reflected in the multiprocessor view in which the goal theorem was stated.

Relating the two models involved showing how a computation on the multiprocessor system could be carried out on the uniprocessor system from the perspective of any selected processor, pid. The required theorem states that for all states and oracles of the multiprocessor system there exist a state and oracle of the uniprocessor system such that the two systems yield "the same" answers on the respective states and oracles. (The notion of equivalence is not equality since the notions of "state" in the two systems are different.) Instead of merely claiming existence however, we construct suitable objects. That is, we define functions that map a state and oracle of one system to a state and oracle of the other and use these functions in the statement of the theorem in place of existentially quantified variables. This is not as distracting as it may seem. Often, when one proves an existential theorem, the proof encodes such a construction anyway.

We focus here on the transformation of the oracle. The goal is to produce from a multiprocessor state and oracle, the sequence of memories seen each time a selected process, pid, is stepped by the uniprocessor model. We first partition the oracle of the multiprocessor system, slicing it at each clock tick at which pid is stepped. The multiprocessor system is then run on each successive partition, starting each run in the appropriate multiprocessor state. The resulting shared memories are then collected as the uniprocessor oracle. This rather messy and ambiguous English description can be succinctly written in ACL2 as:

```
(defun new-oracle (pid s partitions)
  (if (endp partitions)
      nil
      (cons (mem (mrun s (car partitions)))
            (new-oracle pid
                        (mstep pid
                               (mrun s (car partitions)))
                        (cdr partitions)))))
```

where `partitions` is the multiprocessor oracle after slicing it at each `pid`, and `mstep` and `mrun` are the multiprocessor single- and iterated step transformations.

The modeling methods sketched here can be extended to deal with much more complex systems. For example, (Sawada and Hunt, 1998) describes an ACL2 model of a pipelined microprocessor with multiple instruction issue and speculative execution. The model includes interrupts and exceptions. ACL2 is used to prove that the modeled processor implements a sequential instruction set architecture. The combination of the pipeline and external interrupts makes this an especially difficult task, requiring very clear thinking about "when" an interrupt is "seen" by the higher level machine. It is unlikely that a suitable explanation would be given without the aid of mechanized proof.

There are other ways of dealing with nondeterminism in ACL2, including the use of constrained functions or the definition of state machines in terms of a state transition relation rather than a step function.

In (Manolios et al., 1999), a method is shown for using ACL2 to prove that two state machines are bisimilar up to stuttering. States are stuttering bisimilar if they have the same infinite paths up to stuttering; this implies that such states satisfy the same $CTL^*\backslash X$ properties. The method, based on well-founded orderings, allows the proof to focus on single state transitions rather than on infinite execution paths. The method is demonstrated by verifying the alternating bit protocol. The protocol, as initially defined, has an infinite state space. But it is shown to be stuttering bisimilar to a small finite-state system, which is then model-checked.

9 INDUSTRIAL APPLICATIONS

ACL2 finds its most important applications in industry, where it is used to prove properties of models of hardware and low-level software.

For example, ACL2 has been used to prove theorems about formal models of

- the Motorola CAP digital signal processor (Brock et al., 1996),
- the floating-point units of the AMD K5 and K7 (Athlon) microprocessors (Moore et al., 1998; Russinoff, 1998).
- the Rockwell-Collins JEM1 microprocessor (the world's first silicon Java Virtual Machine) (Greve, 1998; Greve and Wilding, 1998), and
- the IBM 4758 secure coprocessor (Smith and Austel, 1998),

With the exception of the IBM 4758 (which involved a security model) the models concerned are executable and both bit- and cycle-accurate. That is, the models completely specify the value of every bit in the machine, on every cycle. Thus, these models are extremely large and complex. Often, however, they follow very closely the style of formalization shown in the TJVM examples above.

In the Motorola CAP work, ACL2 was used to prove the correspondence between two such models. One modeled the pipelined microarchitectural design

of a digital signal processor. The other modeled the sequential instruction set level. The theorem proved by Bishop Brock (described in (Brock et al., 1996)) established that the two models were "equivalent" in a suitable sense, provided that the microcode being executed did not expose certain pipeline "hazards." A predicate was defined to capture this notion of whether microcode contained pipeline hazards. One can view the major contribution of this work as being the identification and certification of this predicate. The theorem just cited can be thought of as a correctness result for the predicate: the designers' model of the hardware and the programmers' model are equivalent for code satisfying the predicate. The predicate is executable. So it was possible to ascertain, without further proof, whether microcode written for the CAP was hazard free. To this end, Brock extracted from the CAP ROM about 50 digital signal processing applications, written by Motorola microcode programmers, and simply executed his predicate on the microcode. Several hazard violations were found; they were eventually recognized as errors and fixed. Whether these errors would have been found by conventional testing is, of course, unknown. But these kinds of errors can be difficult to find: the low level machine is working as the designers expect, the microcode is correctly written for the high level machine, and the problem is that the low level machine does not necessarily implement the high level machine because of the pipeline hazards.

In related work, similar to the proof of the TJVM "fact" program, (Brock and Moore, 1999) shows that the microcode for a systolic peak finding algorithm is correct.

The initial AMD work proved that the microcode for the AMD K5 correctly implements floating point division according to the IEEE 754 floating-point standard (Standards Committee of the IEEE Computer Society, 1985). That standard says that the result computed shall be the floating point number that would be obtained by computing the actual quotient to infinite precision and then rounding it according to the rounding mode specified by the user. The microcode was formalized by hand-translating it to an ACL2 function. Here is a sketch of the function:

```
(defun divide (p d mode)
  (let*
    ((sd0 (eround (lookup d)           '(exact 17 8)))
     (dr  (eround d                   '(away 17 32)))
     (sdd0 (eround (* sd0 dr)         '(away 17 32)))
     (sd1 (eround (* sd0 (comp sdd0 32)) '(trunc 17 32)))
     (sdd1 (eround (* sd1 dr)         '(away 17 32)))
     (sd2 (eround (* sd1 (comp sdd1 32)) '(trunc 17 32)))
     ...
     (qq2 (eround (+ q2 q3)           '(sticky 17 64)))
     (qq1 (eround (+ qq2 q1)           '(sticky 17 64)))
     (divide
      (round   (+ qq1 q0)             mode)))
    (or (first-error sd0 dr sdd0 ... divide)
        divide)))
```

The function takes three inputs, *p*, *d*, and *mode*. The first two are floating-point numbers and the last is the desired rounding mode for the quotient. Roughly speaking, the function sequentially computes values for the variables *sdd0*, *dr*, *sdd0*, etc., and then either returns the first error that was detected or the value of the last variable, *divide*. Errors are modeled by non-numeric values, like strings, and we prove that no error occurs. The variables here represent floating-point registers in the microcode. Consider the line above at which *sdd1* is given a value. Its value is the (infinitely precise) product of *sdd1* and *dr*, rounded according to the mode (*away 17 32*), which is to say, to 32 bits of precision and 17 bits of exponent. The *eround* function signals an error if the product is not representable in 17 bits of exponent. Given the IEEE standard, this line is just the formal way of saying that *sdd1* is the IEEE product of *sdd1* and *dr*, rounding according to (*away 17 32*) and represents one line of the microcode. To formalize *divide*, above, required formalizing that part of the IEEE standard that deals with elementary operations and rounding.

Divide is executable. For example, (*divide 1 3 '(trunc 15 48 mask-uv)*) divides 1 by 3 and truncates it to 48 bits of precision and 15 bits of exponent. The result computed by ACL2 is the rational $93824992236885/281474976710656$. The binary representation of this rational is 0.01. This is the same result given by AMD's simulator (and by the K5 after it was fabricated).

The following theorem was proved about *divide*.

```
(defthm divide-divides
  (implies (and (floating-point-numberp p 15 64)
                (floating-point-numberp d 15 64)
                (not (equal d 0))
                (rounding-modep mode))
            (equal (divide p d mode)
                  (round (/ p d) mode))))
```

Less formally, if *p* and *d* are floating point numbers, *d* $\neq 0$, and *mode* is a rounding mode, then the *divide* function shown above produces the result of rounding the infinitely precise *p/d* to a floating point number as specified by *mode*. This work was done in collaboration with Matt Kaufmann and Tom Lynch, the leader of the AMD K5 floating point design team and is described in (Moore et al., 1998).

To lead ACL2 to this theorem required proving about 1000 lemmas. Many of them are general facts about the relationships between the elementary operations and the various rounding styles.

ACL2 work at AMD continued on AMD K7 microprocessor, better known by its marketing name, the AMD Athlon.

David Russinoff used ACL2 to analyze the hardware designs for parts of the Athlon. With Art Flatau, Russinoff produced a mechanical translator from the RTL design language used at AMD to ACL2. Russinoff then verified mechanically the correctness of the Athlon hardware designs for the floating-point operations of addition, subtraction, multiplication, division, and square

root. The translation from RTL to ACL2 was corroborated by running the ACL2 model of square root on 80 million test vectors, getting precisely the same results as AMD's RTL simulator. While attempting to construct and mechanically check correctness proofs Russinoff found several bugs in these well-tested designs. The bugs were fixed by changes to the RTL designs and the new designs were mechanically verified. See (Russinoff, 1998). One can only speculate as to whether the bugs would have been found by traditional testing before the Athlon was fabricated and shipped. The first Intel Pentium chip contained a bug in the floating point division operation. The bug is reputed to have cost Intel \$500 million to fix, after the chip was shipped to customers.

Researchers at Rockwell-Collins Avionics have produced the world's first silicon Java Virtual Machine and used ACL2 to model it formally. Unlike our TJVM, this model of the JVM is intended to be faithful to the actual design. Proofs were done on some minor parts of the design, primarily to demonstrate that proofs were possible. The main thrust of the work was to produce a formal model that was an efficient execution engine. The ACL2 model can execute about 250,000 JEM1 instructions per second, exploiting ACL2 single-threaded objects from (Boyer and Moore, 1999). With some behind-the-scenes work, the Rockwell engineers have improved this to over 2 million (see (Greve, 1998) and (Greve and Wilding, 1998)). Their model is integrated into a simulation environment so that other Rockwell engineers use it to run compiled Java byte code without knowing they are using a formal model.

In related work, (Cohen, 1997) formalizes (a subset of) Sun's informal specification of the JVM. Cohen's model, called the "defensive JVM" or dJVM, signals errors that are supposed to be avoided in the actual JVM by virtue of the checks made by Sun's byte code verifier. It is, in principle, possible to use the dJVM to investigate whether the verifier successfully guards against these errors.

Matt Kaufmann has used ACL2 to verify that certain transformations on COBOL date processing code produces Y2K compliant results. This work was done in support of commercial Y2K "remediation tools" with which programmers at EDS, Inc., attempt to find and correct COBOL code that makes now outdated assumptions about the century. Kaufmann's basic idea was to formalize the actual date represented by certain common data types, e.g., six-digit year-month-day format, and prove that certain extractions and comparisons produced results consistent with the intended interpretation of "earlier," "difference", etc. See (Kaufmann, 1998).

An emerging application of formal methods is e-commerce, an area where security and reliability are of great importance. At IBM Yorktown, ACL2 was used for the FIPS level 4 evaluation of the IBM 4758 secure coprocessor. This is described by (Smith and Austel, 1998). The IBM 4758 was the first device ever to achieve the highest certification level (4) for cryptographic devices, according to Federal Information Processing Standards (FIPS) 140-1; it is (to date) the only device to have achieved level 4 for both hardware and software. The 4758, about the size of a book, can be used to store confidential data unguarded in a hostile environment. This and other characteristics make the 4758 especially attractive for e-commerce applications. The ACL2 work involved formalizing

a “security model” for the IBM 4758 and proving certain high level theorems about that model.

10 CONCLUSIONS

ACL2 is not the general-purpose theory of computation envisioned by McCarthy. The logic supported by ACL2 is much weaker than that used by McCarthy. For example, he freely uses quantifiers, higher order functions and set theory and can thus investigate questions that fall outside the realm of ACL2.

Why is the ACL2 logic so weak? It is certainly possible to construct mechanical proof checkers for more expressive logics, e.g., HOL (Gordon and Melham, 1993), PVS (Owre et al., 1992), or Otter (McCune, 1994). These systems are used for applications similar to ours. But two good arguments can be made for ACL2’s expressive weakness. The first is that ACL2 was designed with mechanical proof in mind and hence represents a compromise between the two extremes of propositional calculus – where little of interest can be said but everything valid can be proved automatically – and heavy duty set theory – where everything of interest can be said but little can be proved automatically. The second argument is that executability is important for the applications we have in mind. Indeed, the main reason I abandoned almost twenty years of work on the NQTHM logic and started with ACL2 was the conviction that to construct and corroborate models of realistic microprocessors it was necessary to make the logic execute much more efficiently.

As illustrated here, the ACL2 logic is applicable to a wide variety of problems, despite its expressive weakness. This may account for its appeal: it is simple but often just powerful enough to get the job done. In addition, no matter what heavy duty features a logic may have, it will be necessary to reason with grace and power about elementary recursive functions of classic data structures like integers, lists and trees. Hence, I personally see the NQTHM and ACL2 projects as carrying out an exploration that is essential to the eventual success of formal methods, regardless of the power of the logical systems eventually adopted.

11 ACKNOWLEDGMENTS

I am indebted to Bob Boyer and Matt Kaufmann for their years of collaboration on both NQTHM and ACL2. In addition, I would like to thank the many ACL2 users whose work has been cited here, including Vernon Austel, Bishop Brock, Rich Cohen, John Cowles, Art Flatau, David Greve, David Hardin, Warren Hunt, Tom Lynch, Pete Manolios, Kedar Namjoshi, David Russinoff, Jun Sawada, Rob Sumners, and Matt Wilding. I am also indebted to the other ACL2 and NQTHM users who, for almost three decades, have fearlessly applied the systems to problems well beyond those for which they were designed and in so doing showed us what is necessary in a mechanized theory of computation. No project as ambitious as the construction and adoption of a mechanized theory of computation can be carried out without the enthusiastic support and hard work of such risk-taking innovators.

References

- Boyer, R., Goldschlag, D., Kaufmann, M., and Moore, J. S. (1991). Functional instantiation in first-order logic. In Lifschitz, V., editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 7–26. Academic Press.
- Boyer, R. S. and Moore, J. S. (1975). Proving theorems about pure lisp functions. *JACM*, 22(1):129–144.
- Boyer, R. S. and Moore, J. S. (1979). *A Computational Logic*. Academic Press, New York.
- Boyer, R. S. and Moore, J. S. (1996). Mechanized formal reasoning about programs and computing machines. In Veroff, R., editor, *Automated Reasoning and Its Applications: Essays in Honor of Larry Wos*, pages 147–176. MIT Press.
- Boyer, R. S. and Moore, J. S. (1997). *A Computational Logic Handbook, Second Edition*. Academic Press, New York.
- Boyer, R. S. and Moore, J. S. (1999). Single-threaded objects in ACL2. (*submitted for publication*).
- Brock, B., Kaufmann, M., and Moore, J. S. (1996). ACL2 theorems about commercial microprocessors. In Srivastava, M. and Camilleri, A., editors, *Formal Methods in Computer-Aided Design (FMCAD'96)*, pages 275–293. Springer-Verlag, LNCS 1166.
- Brock, B. and Moore, J. S. (1999). A mechanically checked proof of a comparator sort algorithm. (*submitted for publication*).
- Bryant, R. (1992). Symbolic boolean manipulation with ordered binary decision diagrams. Technical Report CMU-CS-92-160, School of Computer Science, Carnegie Mellon University.
- Cohen, R. M. (1997). The defensive Java Virtual Machine specification, version 0.53. Technical report, Electronic Data Systems Corp, Austin Technical Services Center, 98 San Jacinto Blvd, Suite 500, Austin, TX 78701.
- Goodstein, R. L. (1964). *Recursive Number Theory*. North-Holland Publishing Company, Amsterdam.
- Gordon, M. and Melham, T. (1993). *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press.
- Greve, D. A. (1998). Symbolic simulation of the JEM1 microprocessor. Technical Report (submitted), Advanced Technology Center, Rockwell Collins, Inc., Cedar Rapids, IA 52498.
- Greve, D. A. and Wilding, M. M. (Jan. 12, 1998). Stack-based Java a back-to-future step. *Electronic Engineering Times*, page 92.
- Kaufmann, M. (1998). ACL2 support for verification projects. In Kirchner, C. and Kirchner, H., editors, *Proceedings 15th International Conference on Automated Deduction*, pages 220–238. LNAI 1421, Springer-Verlag.
- Kaufmann, M., Manolios, P., and Moore, J. S. (2000). *ACL2: A Tutorial Introduction*. Kluwer.
- Kaufmann, M. and Moore, J. S. (1997). An industrial strength theorem prover for a logic based on Common Lisp. *IEEE Transactions on Software Engineering*, 23(4):203–213.

- Kaufmann, M. and Moore, J. S. (1999a). *The ACL2 User's Manual*. <http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html>.
- Kaufmann, M. and Moore, J. S. (1999b). Structured theory development for a mechanized logic. (*submitted for publication*).
- Lindholm, T. and Yellin, F. (1996). *The Java Virtual Machine Specification*. Addison-Wesley.
- Manolios, P., Namjoshi, K., and Sumners, R. (1999). Linking theorem proving and model-checking with well-founded bisimulation. In *Computed Aided Verification, CAV '99*, pages 369–379. Springer-Verlag LNCS 1633.
- McCarthy, J. (1959). Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91. H.M. Stationery Office.
- McCarthy, J. (1962). Towards a mathematical science of computation. In *Proceedings of IFIP Congress*, pages 21–28. North-Holland.
- McCarthy, J. (1963). A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*, pages 33–70. North-Holland Publishing Company, Amsterdam, The Netherlands.
- McCune, W. (1994). Otter 3.0 Reference Manual and Guide. Tech. Report ANL-94/6, Argonne National Laboratory, Argonne, IL. See also URL <http://www.mcs.anl.gov/AR/otter/>.
- Moore, J. S. (1994). Introduction to the OBDD algorithm for the ATP community. *Journal of Automated Reasoning*, 12(1):33–45.
- Moore, J. S. (1999a). A mechanically checked proof of a multiprocessor result via a uniprocessor view. *Formal Methods in System Design*, 14(2):213–228.
- Moore, J. S. (1999b). Proving theorems about Java-like byte code. In Olderog, E.-R. and Steffen, B., editors, *Correct System Design – Recent Insights and Advances*, pages 139–162. LNCS 1710.
- Moore, J. S., Lynch, T., and Kaufmann, M. (1998). A mechanically checked proof of the correctness of the kernel of the AMD5K86 floating point division algorithm. *IEEE Transactions on Computers*, 47(9):913–926.
- Owre, S., Rushby, J., and Shankar, N. (1992). PVS: A prototype verification system. In Kapur, D., editor, *11th International Conference on Automated Deduction (CADE)*, pages 748–752. Lecture Notes in Artificial Intelligence, Vol 607, Springer-Verlag.
- Russinoff, D. M. (1998). A mechanically checked proof of IEEE compliance of the floating point multiplication, division, and square root algorithms of the AMD-K7 processor. <http://www.onr.com/user/russ/david/k7-div-sqrt.html>.
- Sawada, J. and Hunt, W. (1998). Processor verification with precise exceptions and speculative execution. In *Computed Aided Verification, CAV '98*, pages 135–146. Springer-Verlag LNCS 1427.
- Shoenfield, J. R. (1967). *Mathematical Logic*. Addison-Wesley, Reading, MA.
- Smith, S. W. and Austel, V. (1998). Trusting trusted hardware: Towards a formal model for programmable secure coprocessors. In *Proceedings of the Third USENIX Workshop on Electronic Commerce*, pages 83–98.
- Standards Committee of the IEEE Computer Society (1985). IEEE standard for binary floating-point arithmetic. Technical Report IEEE Std. 754-1985, IEEE, 345 East 47th Street, New York, NY 10017.

Steele, Jr., G. L. (1990). *Common Lisp The Language, Second Edition*. Digital Press, 30 North Avenue, Burlington, MA 01803.

Chapter 24

LOGIC-BASED TECHNIQUES IN DATA INTEGRATION

Alon Y. Levy

*Department of Computer Science and Engineering
University of Washington
Seattle, WA, 98195
alon@cs.washington.edu*

Abstract The data integration problem is to provide uniform access to multiple heterogeneous information sources available online (e.g., databases on the WWW). This problem has recently received considerable attention from researchers in the fields of Artificial Intelligence and Database Systems. The data integration problem is complicated by the facts that (1) sources contain closely related and overlapping data, (2) data is stored in multiple data models and schemas, and (3) data sources have differing query processing capabilities.

A key element in a data integration system is the language used to describe the contents and capabilities of the data sources. While such a language needs to be as expressive as possible, it should also enable to efficiently address the main inference problem that arises in this context: to translate a user query that is formulated over a mediated schema into a query on the local schemas. This paper describes several languages for describing contents of data sources, the tradeoffs between them, and the associated reformulation algorithms.

Keywords: Data integration, description logics, views.

1 INTRODUCTION

The goal of a data integration system is to provide a *uniform* interface to a multitude of data sources. As an example, consider the task of providing information about movies from data sources on the World-Wide Web (WWW). There are numerous sources on the WWW concerning movies, such as the Internet Movie Database (providing comprehensive listings of movies, their casts, directors, genres, etc.), MovieLink (providing playing times of movies in US cities), and several sites providing reviews of selected movies. Suppose we want to find which movies, directed by Woody Allen, are playing tonight in

Seattle, and their respective reviews. None of these data sources *in isolation* can answer this query. However, by combining data from multiple sources, we can answer queries like this one, and even more complex ones. To answer our query, we would first search the Internet Movie Database for the list of movies directed by Woody Allen, and then feed the result into the MovieLink database to check which ones are playing in Seattle. Finally, we would find reviews for the relevant movies using any of the movie review sites.

The most important advantage of a data integration system is that it enables users to focus on specifying *what* they want, rather than thinking about *how* to obtain the answers. As a result, it frees the users from the tedious tasks of finding the relevant data sources, interacting with each source in isolation using a particular interface, and combining data from multiple sources.

The main characteristic distinguishing data integration systems from distributed and parallel database systems is that the data sources underlying the system are *autonomous*. In particular, a data integration system provides access to *pre-existing* sources, which were created independently. Unlike multidatabase systems (see (Litwin et al., 1990) for a survey) a data integration system must deal with a large and constantly changing set of data sources. These characteristics raise the need for richer mechanisms for describing our data, and hence the opportunity to apply techniques from knowledge representation. In particular, a data integration system requires a flexible mechanism for describing contents of sources that may have overlapping contents, whose contents are described by complex constraints, and sources that may be incomplete or only partially complete.

This paper describes the main languages considered for describing data sources in data integration systems (Section 4), which are based on extensions of database query languages. We then discuss the novel challenges arising in designing the appropriate reasoning algorithms (Section 5). Finally, we consider the advantages and challenges of accompanying these languages with a richer knowledge representation formalism based on Description Logics (Section 6).

This paper is not meant to be a comprehensive survey of the work on data integration or even on languages for describing source descriptions. My goal is simply to give a flavor of the main issues that arise and of the solutions that have been proposed. Pointers to more detailed papers are provided throughout.

2 NOTATION

Our discussion will use the terminology of relational databases. A *schema* is a set of relations. Columns of relations are called attributes, and their names are part of the schema (traditionally, the type of each attribute is also part of the schema but we will ignore typing here).

Queries can be specified in a variety of languages. For simplicity, we consider the language of conjunctive queries (i.e., single Horn rule queries), and several variants on it. A conjunctive query has the form:

$$q(\bar{X}) \vdash e_1(\bar{X}_1), \dots, e_n(\bar{X}_n),$$

where e_1, \dots, e_n are database relations, and $\bar{X}_1, \dots, \bar{X}_n$ are tuples of variables or constants. The atom $q(\bar{X})$ is the head of the query, and the result of the

query is a set of tuples, each giving a binding for every variable in \bar{X} . We assume the query is safe, i.e., $\bar{X} \subseteq \bar{X}_1 \cup \dots \cup \bar{X}_n$. Interpreted predicates such as $<$, \leq , \neq are sometimes used in the query. Queries with unions are expressed by multiple rules with the same head predicate. A *view* refers to a named query, and it is said to be *materialized* if its results are stored in the database.

The notions of query containment and query equivalence are important in order to enable comparison between different formulations of queries.

Definition 1 Query containment and equivalence. A query Q_1 is said to be contained in a query Q_2 , denoted by $Q_1 \sqsubseteq Q_2$, if for any database D , $Q_1(D) \subseteq Q_2(D)$. The two queries are said to be equivalent if $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$.

The problems of query containment and equivalence have been studied extensively in the literature. Some of the cases which are most relevant to our discussion include: containment of conjunctive queries and unions thereof (Chandra and Merlin, 1977; Sagiv and Yannakakis, 1981), conjunctive queries with built-in comparison predicates (Klug, 1988), and Datalog queries (Shmueli, 1993; Sagiv, 1988; Levy and Sagiv, 1993; Chaudhuri and Vardi, 1993; Chaudhuri and Vardi, 1994).

3 CHALLENGES IN DATA INTEGRATION

As described in the introduction, the task of a data integration system is to provide a uniform interface to a collection of data sources. The data sources can either be full-fledged database systems (of various flavors: relational, object-oriented, etc.), legacy systems, or structured files hidden behind some interface program. For the purposes of our discussion we model data sources as containing relations. In this paper (as in most of the research) we only consider data integration systems whose goal is to query the data, and not to perform updates on the sources.

Even though this paper focuses on the problem of modeling data sources and query reformulation, it is worthwhile to first highlight some of the challenges involved in building data integration systems. To do so, we briefly compare data integration systems with traditional database management systems. Figure 24.1 illustrates the different stages in processing a query in a data integration system.

Data modeling: in a traditional database application one begins by modeling the requirements of the application, and designing a database schema that appropriately supports the application. As noted earlier, a data integration application begins from a set of pre-existing data sources. Hence, the first step of the application designer is to develop a *mediated schema* (often referred to as a *global schema*) that describes the data that exists in the sources, and exposes the aspects of this data that may be of interest to users. Note that the mediated schema does not necessarily contain all the relations and attributes modeled in each of the sources. Users pose queries in terms of the mediated schema, rather than directly in terms of the source schemas. As such, the mediated schema is a

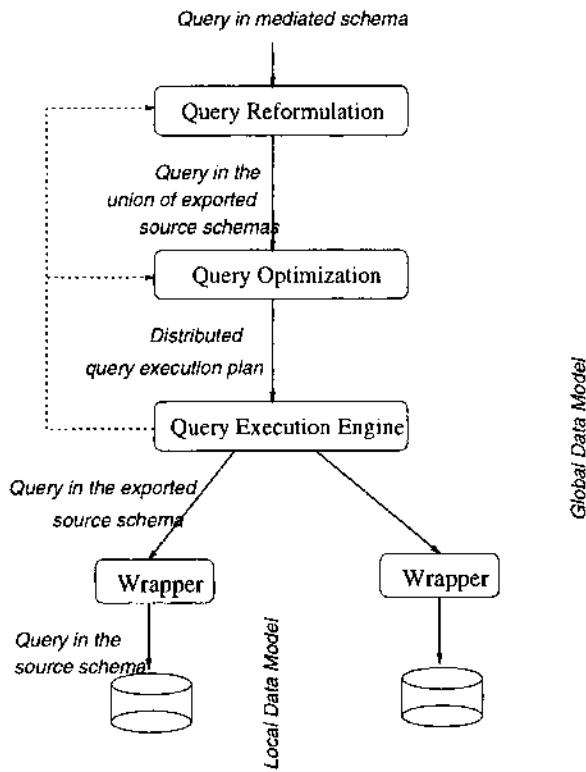


Figure 24.1 Prototypical architecture of a data integration system

set of *virtual* relations, in the sense that they are not actually stored anywhere. For example, in the movie domain, the mediated schema may contain the relation MOVIE(TITLE, YEAR, DIRECTOR, GENRE) describing the different properties of a movie, the relation MOVIEACTOR(TITLE, NAME), representing the cast of a movie, MOVIE REVIEW(TITLE, REVIEW) representing reviews of movies, and AMERICAN(DIRECTOR) storing which directors are American.

Along with the mediated schema, the application designer needs to supply *descriptions* of the data sources. The descriptions specify the relationship between the relations in the mediated schema and those in the local schemas at the sources. The description of a data source specifies its contents (e.g., contains movies), attributes (e.g., genre, cast), constraints on its contents (e.g., contains only American movies), completeness and reliability, and finally, its query processing capabilities (e.g., can perform selections, or can answer arbitrary SQL queries).

The fact that data sources are pre-existing requires that we be able to handle the following characteristics in the language for describing the sources:

1. Overlapping and even contradictory data among different sources.

2. Semantic mismatches among sources: since each of the data sources has been designed by a different organization for different purposes, the data is modeled in different ways. For example, one source may store a relational database in which all the attributes of a particular movie are stored in one table, while another source may spread the attributes across several relations. Furthermore, the names of the attributes and of the tables will be different from one source to another, as will the choice of what should be a table and what should be an attribute.
3. Different naming conventions for data values: sources use different names or formats to refer to the same object. Simple examples include various conventions for specifying addresses or dates. Cases in which persons are named differently in the sources are harder to deal with (e.g., one source contains the full name, while another contains only the initials of the first name). A recent elegant treatment of this problem is presented in (Cohen, 1998).

Query reformulation: a user of a data integration system poses queries in terms of the mediated schema, rather than directly in the schema in which the data is stored. As a consequence, a data integration system must contain a module that uses the source descriptions in order to *reformulate* a user query into a query that refers directly to the schemas of the sources. Such a reformulation step does not exist in traditional database systems. Clearly, as the language for describing data sources becomes more expressive, the reformulation step becomes harder. Clearly, we would like the reformulation to be sound, (i.e., the answers to the reformulated query should all be correct answers to the input query), and complete (i.e., all the answers that can be extracted from the data sources should be in the result of applying the reformulated query). In addition, we want the reformulation to produce an efficient query, e.g., to ensure that we do not access irrelevant sources (i.e., sources that cannot contribute any answer or partial answer to the query).

Wrappers: the other layer of a data integration system that does not exist in a traditional system is the wrapper layer. Unlike a traditional query execution engine that communicates with a local storage manager to fetch the data, the query execution plan in a data integration system must obtain data from remote sources. A wrapper is a program which is specific to a data source, whose task is to translate data from the source to a form that is usable by the query processor of the system. For example, if the data source is a web site, the task of the wrapper is to translate the query to the source's interface, and when the answer is returned as an HTML document, it needs to extract a set of tuples from that document. (Clearly, the emergence of XML as a standard for data exchange on the WWW will alleviate much of the wrapper building problem).

Query optimization and execution: a traditional relational database system accepts a *declarative* SQL query. The query is first parsed and then

passed to the *query optimizer*. The role of the optimizer is to produce an efficient *query execution plan*, which is an imperative program that specifies exactly how to evaluate the query. In particular, the plan specifies the *order* in which to perform the different operations in the query (join, selection, projection), a specific algorithm to use for each operation (e.g., sort-merge join, hash-join), and the scheduling of the different operators. Typically, the optimizer selects a query execution plan by searching a space of possible plans, and comparing their estimated cost. To evaluate the cost of a query execution plan the optimizer relies on extensive statistics about the underlying data, such as sizes of relations, sizes of domains and the selectivity of predicates. Finally, the query execution plan is passed to the query execution engine which evaluates the query.

The main differences between the traditional database context and that of data integration are the following:

- Since the sources are autonomous, the optimizer may have no statistics about the sources, or unreliable ones. Hence, the optimizer cannot compare between different plans, because their costs cannot be estimated.
- Since the data sources are not necessarily database systems, the sources may appear to have different processing capabilities. For example, one data source may be a web interface to a legacy information system, while another may be a program that scans data stored in a structured file (e.g., bibliography entries). Hence, the query optimizer needs to consider the possibility of exploiting the query processing capabilities of a data source. Note that query optimizers in distributed database systems also evaluate where parts of the query should be executed, but in a context where the different processors have identical capabilities.
- Finally, in a traditional system, the optimizer can reliably estimate the time to transfer data from the disc to main memory. But in a data integration system, data is often transferred over a wide-area network, and hence delays may occur for a multitude of reasons. Therefore, even a plan that appears to be the best based on cost estimates may turn out to be inefficient if there are unexpected delays in transferring data from one of the sources accessed early on in the plan.

4 MODELING DATA SOURCES AND QUERY REFORMULATION

As described in the previous section, one of the main differences between a data integration system and a traditional database system is that users pose queries in terms of a mediated schema. The data, however, is stored in the data sources, organized under local schemas. Hence, in order for the data integration system to answer queries, there must be some description of the relationship between the source relations and the mediated schema. The query processor of the integration system must be able to reformulate a query posed on the mediated schema into a query against the source schemas.

In principle, one could use arbitrary formulas in first-order logic to describe the data sources. But in such a case, sound and complete reformulation would be practically impossible. Hence, several approaches have been explored in which restricted forms of first-order formulas have been used in source descriptions, and effective accompanying reformulation algorithms have been presented. We describe two such approaches: the *Global as view* approach (GAV) (Garcia-Molina et al., 1997; Papakonstantinou et al., 1996; Adali et al., 1996; Florescu et al., 1996), and the *Local as view* approach (LAV) (Levy et al., 1996b; Kwok and Weld, 1996; Duschka and Genesereth, 1997a; Duschka and Genesereth, 1997b; Friedman and Weld, 1997; Ives et al., 1999).

Global As View: In the GAV approach, for each relation R in the mediated schema, we write a query over the source relations specifying how to obtain R 's tuples from the sources.

For example, suppose we have two sources DB_1 and DB_2 containing titles, actors and years of movies. We can describe the relationship between the sources and the mediated schema relation MOVIEACTOR as follows:

$$DB_1(id, title, actor, year) \Rightarrow \text{MOVIEACTOR}(title, actor)$$

$$DB_2(id, title, actor, year) \Rightarrow \text{MOVIEACTOR}(title, actor).$$

If we have a third source that shares movie identifiers with DB_1 and provides movie reviews, the following sentence describes how to obtain tuples for the MOVIEREVIEW relation:

$$DB_1(id, title, actor, year) \wedge DB_3(id, review) \Rightarrow$$

$$\text{MOVIEREVIEW}(title, review)$$

In general, GAV descriptions are Horn rules that have a relation in the mediated schema in the consequent, and a conjunction of atoms over the source relations in the antecedent.

Query reformulation in GAV is relatively straightforward. Since the relations in the mediated schema are defined in terms of the source relations, we need only unfold the definitions of the mediated schema relations. For example, suppose our query is to find reviews for movies starring Marlon Brando:

$$q(title, review) : - \text{MOVIEACTOR}(title, 'Brando'),$$

$$\text{MOVIEREVIEW}(title, review).$$

Unfolding the descriptions of MOVIEACTOR and MOVIEREVIEW will yield the following queries over the source relations: (the second of which will obviously be deemed redundant)

$$q(title, review) : - DB_1(id, title, 'Brando', year), DB_3(id, review)$$

$$q(title, review) : - DB_1(id, title, 'Brando', year),$$

$$DB_2(id, 'Brando', year), DB_3(id, review)$$

Local As View: The LAV approach the descriptions of the sources are given in the opposite direction. That is, the contents of a data source are described as a query over the mediated schema relation.

Suppose we have two sources: (1) V_1 , containing titles, years and directors of American comedies produced after 1960, and (2) V_2 containing movie reviews produced after 1990. In LAV, we would describe these sources by the following

formulas (variables that appear only on the right hand sides are assumed to be existentially quantified):

$$S_1 : V_1(title, year, director) \Rightarrow \text{MOVIE}(title, year, director, genre) \wedge \\ \text{AMERICAN}(director) \wedge \\ \text{year} \geq 1960 \wedge \text{genre} = \text{Comedy}.$$

$$S_2 : V_2(title, review) \Rightarrow \text{MOVIE}(title, year, director, genre) \wedge \text{year} \geq 1990 \wedge \\ \text{MOVIEREVIEW}(title, review).$$

Query reformulation in LAV is more tricky than in GAV, because it is not possible to simply unfold the definitions of the relations in the mediated schema. For example, suppose our query asks for reviews for comedies produced after 1950:

$$q(title, review) : -\text{MOVIE}(title, year, director, Comedy), \text{year} \geq 1950, \\ \text{MOVIEREVIEW}(title, review).$$

The reformulated query on the sources would be:

$$q'(title, review) : -V_1(title, year, director), V_2(title, review).$$

Note that in this case, the reformulated query is not equivalent to the original query, because it only returns movies that were produced after 1990. However, given that we only have the sources S_1 and S_2 , this is the best reformulation possible. In the next section we define precisely the reformulation problem in LAV and present several algorithms for solving it.

A Comparison of the Approaches:. The main advantage of the GAV approach is that query reformulation is very simple, because it reduces to rule unfolding. However, adding sources to the data integration system is non-trivial. In particular, given a new source, we need to figure out all the ways in which it can be used to obtain tuples for each of the relations in the mediated schema. Therefore, we need to consider the possible interaction of the new source with each of the existing sources, and this limits the ability of the GAV approach to scale to a large collection of sources.

In contrast, in the LAV approach each source is described in isolation. It is the system's task to figure out (at query time) how the sources interact and how their data can be combined to answer the query. The downside, however, is that query reformulation is harder, and sometimes requires recursive queries over the sources. An additional advantage of the LAV approach is that it is easier to specify rich constraints on the contents of a source (simply by specifying more conditions in the source descriptions). Specifying complex constraints on sources is essential if the data integration system is to distinguish between sources with closely related and overlapping data. Finally, (Friedman et al., 1999) described the GLAV language that combines the expressive power of GAV and LAV, and query reformulation complexity is the same as for LAV.

5 ANSWERING QUERIES USING VIEWS

The reformulation problem for LAV can be intuitively explained as follows. Because of the form of the LAV descriptions, each of the sources can be viewed as containing an *answer* to a query over the mediated schema (an answer to

the query expressed by the right hand side of the source description). Hence, sources represent materialized answers to queries over the virtual mediated schema. A user query is also posed over the mediated schema. The problem is therefore to find a way of answering the user query using only the answers to the queries describing the sources.

The problem of answering a query using a set of previously materialized views has received significant attention because of its relevance to other database problems, such as query optimization (Chaudhuri et al., 1995), maintaining physical data independence (Yang and Larson, 1987; Tsatalos et al., 1996), and data warehouse design.

Formally, suppose we are given a query Q and a set of view definitions V_1, \dots, V_m . A rewriting of the query using the views is a query expression Q' whose subgoals use *only* view predicates or interpreted predicates. We distinguish between two types of query rewritings: *equivalent rewritings* and *maximally-contained rewritings*.

Definition 2 (Equivalent rewritings). Let Q be a query and $\mathcal{V} = V_1, \dots, V_m$ be a set of view definitions. The query Q' is an equivalent rewriting of Q using \mathcal{V} if:

- Q' refers only to the views in \mathcal{V} , and
- Q' is equivalent to Q .

Definition 3 (Maximally-contained rewritings). Let Q be a query and $\mathcal{V} = V_1, \dots, V_m$ be a set of view definitions in a query language \mathcal{L} . The query Q' is a maximally-contained rewriting of Q using \mathcal{V} w.r.t. \mathcal{L} if:

- Q' refers only the views in \mathcal{V} ,
- Q' is contained in Q , and
- there is no rewriting Q_1 , such that $Q' \subseteq Q_1 \subseteq Q$ and Q_1 is not equivalent to Q' .

Equivalent rewritings of a query are needed when materialized views are used for query optimization and physical data independence. In the data integration context, we usually need to settle for maximally-contained rewritings, because, as we saw in the previous section, the sources do not necessarily contain all the information needed to provide an equivalent answer to the query.

Recent research has considered many variants of the problem of answering queries using views (see (Levy, 1999) for a survey). The problem has been shown to be NP-complete even when the queries describing the sources and the user query are conjunctive and don't contain interpreted predicates (Levy et al., 1995). In fact, (Levy et al., 1995) shows that in the case of conjunctive queries, we can limit the candidate rewritings that we consider to those that have at most the number of subgoals in the query. Importantly, the complexity of the problem is polynomial in the number of views (i.e., the number of data sources in the context of data integration).

In what follows we describe two algorithms for answering queries using views that were developed specifically for the context of data integration. These algorithms are the *bucket algorithm* developed in the context of the Information Manifold system (Levy et al., 1996b), and the *inverse-rules algorithm* (Qian, 1996; Duschka and Genesereth, 1997b; Duschka et al., 1999) which was implemented in the InfoMaster system (Duschka and Genesereth, 1997b).

5.1 THE BUCKET ALGORITHM

The goal of the bucket algorithm is to reformulate a user query that is posed on a mediated (virtual) schema into a query that refers directly to the available data sources. Both the query and the sources are described by select-project-join queries that may include atoms of arithmetic comparison predicates (hereafter referred to simply as predicates). The bucket algorithm returns the maximally contained rewriting of the query using the views.

The main idea underlying the bucket algorithm is that the (possibly exponential) number of query rewritings that need to be considered can be drastically reduced if we first consider each subgoal in the query in isolation, and determine which views may be relevant to each subgoal. Given a query Q , the bucket algorithm proceeds in two steps. In the first step, the algorithm creates a bucket for each subgoal in Q , containing the views (i.e., data sources) that are relevant to answering the particular subgoal. More formally, a view V is put in the bucket of a subgoal g in the query if the definition of V contains a subgoal g_1 such that

- g and g_1 can be unified, and
- after applying the unifier to the query and to the variables of the view that appear in the head, the predicates in Q and in V are mutually satisfiable.

The actual bucket contains the head of the view V after applying the unifier to the head of the view. Note that a subgoal g may unify with more than one subgoal in a view V , and in that case the bucket of g will contain multiple occurrences of V .

In the second step, the algorithm considers query rewritings that are conjunctive queries, each consisting of one conjunct from every bucket. Specifically, for each possible choice of element from each bucket, the algorithm checks whether the resulting conjunction is contained in the query Q , or whether it can be made to be contained if additional predicates are added to the rewriting. If so, the rewriting is added to the answer. Hence, the result of the bucket algorithm is a union of conjunctive rewritings.

Example 1 Consider an example including views over the following schema:

Enrolled(student, dept) Registered(student, course, year)

Course(course, number)

$V1(\text{student}, \text{number}, \text{year}) :- \text{Registered}(\text{student}, \text{course}, \text{year}),$
 $\quad \quad \quad \text{Course}(\text{course}, \text{number}),$
 $\quad \quad \quad \text{number} \geq 500, \text{year} \geq 1992.$

$V2(\text{student}, \text{dept}, \text{course}) :- \text{Registered}(\text{student}, \text{course}, \text{year}),$

Enrolled(student,dept)

V3(student,course) :- Registered(student,course,year), year \leq 1990.

V4(student,course,number) :- Registered(student,course,year),
Course(course,number),
Enrolled(student,dept), number \leq 100

Suppose our query is:

$q(S,D) :- \text{Enrolled}(S,D), \text{Registered}(S,C,Y), \text{Course}(C,N), N \geq 300, Y \geq 1995.$

In the first step of the algorithm we create a bucket for each of the relational subgoals in the query in turn. The resulting contents of the buckets are shown in Table 24.1. The bucket of $\text{Enrolled}(S,D)$ will include the views V2 and V4, since the following mapping maps the atom in the query into the corresponding Enrolled atom in the views:

{ $S \rightarrow \text{student}, D \rightarrow \text{dept}$ }.

Note that the view head in the bucket only includes the variables in the domain of the mapping, and fresh variables (primed) for the other head variables of the view.

The bucket of the subgoal $\text{Registered}(S,C,Y)$ will contain the views V1, V2 and V4 since the following mapping maps the atom in the query into the corresponding Registered atom in the views:

{ $S \rightarrow \text{student}, C \rightarrow \text{course}, Y \rightarrow \text{year}$ }.

$\text{Enrolled}(S,D)$	$\text{Registered}(S,C,Y)$	$\text{Course}(C,N)$
V2(S,D,C') V4(S,C',N')	V1(S,N',Y) V2(S,D',C) V4(S,C,N')	V1(S',N,Y')

Table 24.1 Contents of the buckets. The primed variables are those that are not in the domain of the unifying mapping.

The view V3 is not included in the bucket of $\text{Registered}(S,C,Y)$ because the predicates $Y \geq 1995$ and $\text{year} \leq 1990$ are mutually inconsistent. On the other hand, one may wonder why V4 is included in the bucket, since the predicates on the course number in the view and in the query are mutually inconsistent. However, the point to note is that the mapping from the query to the view does not map the variable D to the variable number in V4. Hence, the contradiction does not arise.¹

Next, consider the bucket of the subgoal $\text{Course}(C,N)$. The view V1 will be included in the bucket because of the mapping
{ $C \rightarrow \text{course}, N \rightarrow \text{number}$ }.

¹However, if we also knew that the course name functionally determines its number, then we could prune V4 from this bucket.

Note that in this case the view V4 is *not* included in the bucket because the unifier does map N to number, and hence the predicates on the course number would be mutually inconsistent.

In the second step of the algorithm, we combine elements from the buckets. In our example, the only combination that would yield a solution is joining V1 and V2 as follows:

$q'(S, D) :- V1(S, N, Y), V2(S, D, C), Y \geq 1995.$

The only other option that would not involve a redundant view subgoal in the rewriting, would involve a join between V1 and V4 and is dismissed because the two views contain disjoint numbers of courses (greater than 500 for V1 and less than 100 for V4). In this case, the views V1 and V4 are relevant to the query in *isolation*, but, if joined, produce the empty answer. Finally, the reader should also note that in this example, as usually happens in the data integration context, the algorithm produced a *maximally-contained* rewriting of the query using the views, and not an equivalent rewriting.

5.2 THE INVERSE-RULES ALGORITHM

Like the bucket algorithm, the inverse-rules algorithm was also developed in the context of a data integration system (Duschka and Genesereth, 1997b). The key idea underlying the algorithm is to construct a set of rules that *invert* the view definitions, i.e., rules that show how to compute tuples for the database relations from tuples of the views. We illustrate inverse rules with an example.

Suppose we have the following view:

$V3(\text{dept}, \text{c-name}) :- \text{Enrolled}(\text{s-name}, \text{dept}), \text{Registered}(\text{s-name}, \text{c-name}).$

We construct one inverse rule for every conjunct in the body of the view:

$\text{Enrolled}(f_1(\text{dept}, X), \text{dept}) :- V3(\text{dept}, X)$

$\text{Registered}(f_1(Y, \text{c-name}), \text{c-name}) :- V3(Y, \text{c-name})$

Intuitively, the inverse rules have the following meaning. A tuple of the form $(\text{dept}, \text{name})$ in the extension of the view V3 is a witness of tuples in the relations Enrolled and Registered. The witness provides only partial information, and hides the rest. In particular, it tells us two things:

- the relation Enrolled contains a tuple of the form (Z, dept) , for some value of Z.
- the relation Registered contains a tuple of the form (Z, name) , for the same value of Z.

In order to express the information that the unknown value of Z is the same in the two atoms, we refer to it using the functional term $f_1(\text{dept}, \text{name})$. Note that there may be several values of Z in the database that cause the tuple $(\text{dept}, \text{name})$ to be in the join of Enrolled and Registered, but all that matters is that there exists at least one such value.

In general, we create one function symbol for every existential variable that appears in the view definitions, and these function symbols are used in the heads of the inverse rules.

The rewriting of a query Q using the set of views V is the Datalog program that includes

- the inverse rules for V , and
- the query Q .

As shown in (Duschka and Genesereth, 1997a), the inverse-rules algorithm returns the maximally contained rewriting of Q using V . In fact, the algorithm returns the maximally contained query even if Q is an arbitrary Datalog program.

Example 2 Suppose our query asks for the departments in which the students of the "Database" course are enrolled,

$q(\text{dept}) \leftarrow \text{Enrolled}(\text{s-name}, \text{dept}), \text{Registered}(\text{s-name}, \text{"Database"})$

and the view V_3 includes the tuples:

$\{ (\text{CS}, \text{"Database"}), (\text{EE}, \text{"Database"}), (\text{CS}, \text{"AI"}) \}$

The inverse rules would compute the following tuples:

$\text{Registered: } \{ (f_1(\text{CS}, \text{"Database"}), \text{CS}), (f_1(\text{EE}, \text{"Database"}), \text{EE}), (f_1(\text{CS}, \text{"AI"}), \text{CS}) \}$

$\text{Enrolled: } \{ (f_1(\text{CS}, \text{"Database"}), \text{"Database"}),$

$(f_1(\text{EE}, \text{"Database"}), \text{"Database"}), (f_1(\text{CS}, \text{"AI"}), \text{"AI"}) \}$

Applying the query to these extensions would yield the answers CS and EE.

In this example we showed how functional terms are generated as part of the evaluation of the inverse rules. However, the resulting rewriting can actually be rewritten in such a way that no functional terms appear (Duschka and Genesereth, 1997a).

5.3 COMPARISON OF THE ALGORITHMS

There are several interesting similarities and differences between the bucket and inverse rules algorithms that are worth noting. In particular, the step of computing buckets is similar in spirit to that of computing the inverse rules, because both of them compute the views that are relevant to single atoms of the database relations. The difference is that the bucket algorithm computes the relevant views by taking into consideration the *context* in which the atom appears in the query, while the inverse rules algorithm does not. Hence, if the predicates in a view definition entail that the view cannot provide tuples relevant to a query (because they are mutually unsatisfiable with the predicates in the query), then the view will not end up in a bucket. In contrast, the query rewriting obtained by the inverse rules algorithm may result in containing views that are not relevant to the query. However, the inverse rules can be computed once, and be applicable to any query. In order to remove irrelevant views from the rewriting produced by the inverse-rules algorithm we need to apply a subsequent constraint propagation phase (as in (Levy et al., 1997; Srivastava and Ramakrishnan, 1992)).

The strength of the bucket algorithm is that it exploits the predicates in the query to prune significantly the number of candidate conjunctive rewritings that need to be considered. Checking whether a view should belong to a bucket can be done in time polynomial in the size of the query and view definitions when the predicates involved are arithmetic comparisons. Hence,

if the data sources (i.e., the views) are indeed distinguished by having different comparison predicates, then the resulting buckets will be relatively small. In the second step, the algorithm needs to perform a query containment test for every candidate rewriting. The testing problem is π_2^P -complete, but only in the size of the query and the view definition, and hence quite efficient in practice. The bucket algorithm also extends naturally to unions of conjunctive queries, and to other forms of predicates in the query such as class hierarchies. Finally, the bucket algorithm also makes it possible to identify opportunities for interleaving optimization and execution in a data integration system in cases where one of the buckets contains an especially large number of views.

The inverse-rules algorithm has the advantage of being modular. The inverse rules can be computed ahead of time, independent of a specific query. As shown in (Duschka and Genesereth, 1997a; Duschka and Levy, 1997), extending the algorithm to handle functional dependencies on the database schema, recursive queries or the existence of binding pattern limitations can be done by adding another set of rules to the resulting rewriting. Unfortunately, the algorithm does not handle arithmetic comparison predicates, and extending it to handle such predicates turns out to be quite subtle. The algorithm produces the maximally-contained rewriting in time that is polynomial in the size of the query and the views (but the complexity of removing irrelevant views has exponential time complexity (Levy et al., 1997)).

MiniCon Algorithm (Pottinger and Levy, 1999) builds on the ideas of both the bucket algorithm and the inverse rules algorithm. Extensive experiments described in (Pottinger and Levy, 1999) show that the MiniCon algorithm significantly outperforms both of its predecessors, and scales up gracefully to hundreds and even thousands of views.

5.4 THE NEED FOR RECURSIVE REWRITINGS

An interesting phenomenon in several variants of LAV descriptions is that the reformulated query may actually have to be *recursive* in order to provide the maximal answer. The most interesting of these variants is the common case in which data sources can only be accessed with particular patterns.

Consider the following example, where the first source provides papers (for simplicity, identified by their title) published in AAAI, the second source records citations among papers, and the third source stores papers that have won significant awards. The superscripts in the source descriptions depict the access patterns that are available to the sources. The superscripts contain strings over the alphabet $\{b, f\}$. If a b appears in the i 'th position, then the source requires a binding for the i 'th attribute in order to produce provide answers. If an f appears in the i 'th position, then the i 'th attribute may be either bound or not. From the first source we can obtain all the AAAI papers (no bindings required); to obtain data from the second source, we first must provide a binding for a paper and then receive the set of papers that it cites; with the third source we can only query whether a *given* paper won an award, but not ask for all the award winning papers.

$$\begin{aligned} AAAIdb^f(X) &\Rightarrow AAAIPapers(X) \\ CitationDB^{bf}(X, Y) &\Rightarrow Cites(X, Y) \\ AwardDB^b(X) &\Rightarrow AwardPaper(X) \end{aligned}$$

Suppose our query is to find all the award winning papers:

$$Q(X) : -AwardPaper(X)$$

As the following queries show, there is no finite number of conjunctive queries over the sources that is guaranteed to provide *all* the answers to the query. In each query, we can start from the AAAI database, follow citation chains of length n , and feed the results into the award database. Since we cannot limit the length of a citation chain we need to follow *a priori* (without examining the data), we cannot put a bound on the size of the reformulated query.

$$Q'(X) : -AAAIdb(X), AwardDB(X)$$

$$Q'(X) : -AAAIdb(V), CitationDB(V, X_1), \dots, CitationDB(X_n, X), \\ AwardDB(X).$$

However, if we consider recursive queries over the sources, we can obtain a finite concise query that provides all the answers, as follows (note that the newly invented relation *papers* is meant to represent the set of all papers reachable from the AAAI database):

$$papers(X) : -AAAIdb(X)$$

$$papers(X) : -papers(Y), CitationDB(Y, X)$$

$$Q'(X) : -papers(X), AwardDB(X).$$

Other cases in which recursion may be necessary are in the presence of functional dependencies on the mediated schema (Duschka and Levy, 1997), when the user query is recursive (Duschka and Genesereth, 1997a), and, as we show in the next section, when the descriptions of the sources are enriched by description logics (Beeri et al., 1997).

Finally, it turns out that slight changes to the form of source descriptions in LAV can cause the problem of answering queries to become NP-hard in the size of the data in the sources (Abiteboul and Duschka, 1998). One example of such a case is when the query contains the predicate \neq . Other examples include cases in which the sources are known to be *complete* w.r.t. their descriptions (i.e., where the \Rightarrow in the source description is replaced by \Leftrightarrow).

6 THE USE OF DESCRIPTION LOGICS

Thus far, we have described the mediated schema and the local schemas as flat sets of relations. However, in many applications we would like to present users with an interface that includes a rich model of the underlying domain, and be able to pose the integration queries over such a model. Furthermore, providing the mapping between source relations and a mediated schema is considerably facilitated when the the mediated schema is part of a rich domain model. Several works have considered the use of a richer representation based on description logics in the mediated schema (Arens et al., 1996; Catarcı and Lenzerini, 1993; Levy et al., 1996a; Lattes and Rousset, 1998).

Description logics are a family of logics for modeling complex hierarchical structures. In a description logic it is possible to intensionally define sets of objects, called *concepts*, using descriptions built from a set of constructors.

For example, the set $\text{Person} \sqcap (\leq 3 \text{ child}) \sqcap (\forall \text{ child smart})$ describes the set of objects belonging to the class `person` who have at most 3 children, and all of their children belong to the class `smart`. In addition to defining concepts, it is also possible to state that an object belongs to a concept. In fact, it is possible to say that `Fred` belongs to the above concept, *without* specifying who his children are, and how many there are.

As originally suggested in (Catarci and Lenzerini, 1993), when the source descriptions and mediated schema both use description logics, it is possible to use the reasoning service of the underlying logic to detect when a source is relevant to a query. However, description logics in themselves are not expressive enough to model arbitrary relational data. In particular, they are unable to express arbitrary joins of relations. Hence, several hybrid languages combining the expressive power of Horn rules and description logics have been investigated (Levy and Rousset, 1998; Donini et al., 1991; MacGregor, 1994; Cadoli et al., 1997) with associated reasoning algorithms.

Answering queries using views in the context of these hybrid languages is still largely an open problem. As the following example shows, in the presence the rewriting of a query using a set of views may need to be recursive.

Example 3 Consider the following two views:

$v_1(X, Y) : -\text{child}(X, Y), \text{child}(X, Z), (\leq 1 \text{ child})(Z)$

$v_2(X) : -(\geq 3 \text{ child})(X)$.

The atom $(\leq 1 \text{ child})(X)$ denotes the set of objects that have at most one child. Similarly, $(\geq 3 \text{ child})(X)$ denotes the objects with 3 children or more. Suppose the query Q is to find all individuals who have at least 2 children, i.e.:

$q(X) : -(\geq 2 \text{ child})(X)$.

For any n , the following conjunctive query is a rewriting that is contained in Q :

$$q'_n(X) : - v_1(X, Y_1), v_1(Y_1, Y_2), v_1(Y_n, U), v_2(U).$$

To see why, consider the variable Y_n . The view v_1 entails that it has one filler for the role `child` that has less than one child (the variable Z in the definition of v_1) while the view v_2 says that its child U has at least 3 children. Therefore, Y_n has at least 2 children. The same line of reasoning can be used to see that Y_{n-1} has at least 2 children, and continuing in the same way, we get that X has at least two children, i.e., $q(X)$ holds. The point is that inferring $q(X)$ required a chain of inference that involved an arbitrary number of constants, and the length of the chain does not depend on the query or the views. Furthermore, for any value of n , we can build a database such that the union of q'_1, \dots, q'_n is *not* the maximally-contained rewriting, and therefore we cannot find a maximally-contained rewriting that is a union of conjunctive queries.

We can still find a maximally-contained rewriting in this example if we allow the rewriting to be a recursive Datalog program. The following is a maximally-contained rewriting:

$q'(X) : -v_2(X)$

$q'(X) : -v_1(X, Y), q'(Y).$

This recursive program essentially mimics the line of reasoning we described above.

The problem of answering queries using views in the context of description logics is further discussed in (Beeri et al., 1997; Calvanese et al., 1999). In (Beeri et al., 1997) it is shown that under very tight restrictions, it is always possible to find a maximally-contained rewriting of the query using a set of views, and otherwise, it may not be possible to find a maximally contained rewriting in Datalog.

7 CONCLUDING REMARKS

In this article I touched upon some of the problems of data integration in which logic-based methods provided useful solutions. In particular, I showed how semantic relationships between the contents of data sources and relations in a mediated schema can be specified using limited forms of logic, and illustrated the associated reasoning algorithms. There are additional areas in which logic based techniques have been applied, including descriptions of source completeness (Etzioni et al., 1994; Levy, 1996; Duschka, 1997), and descriptions of query processing capabilities of data sources (Levy et al., 1996c; Vassalos and Papakonstantinou, 1997).

There are two areas in which I believe significant future research is required. The first area concerns the role of more expressive knowledge representation languages in data integration. Currently, relatively little is known about the query reformulation problem in cases where the mediated schema and/or the data source schemas are described using a richer knowledge representation language. In addition, we need to carefully study which extensions to the expressive power are really needed in practice.

The second area, which is of wider applicability than the problem of data integration, concerns extending logical reasoning techniques to deal with non-logical extensions that have been needed in database systems. In order to deal with real world applications, commercial database systems provide support for dealing with issues such as bag (multiset) semantics, grouping and aggregation, and nested structures. In order to deal with many real world domains, we need to develop reasoning techniques (extensions of query containment algorithms and algorithms for answering queries using views) to settings where the queries involve bags, grouping and aggregation, and nested structures (see (Cohen et al., 1999; Levy and Suciu, 1997; Chaudhuri and Vardi, 1993; Srivastava et al., 1996) for some work in these areas). Another interesting question is whether some of these features can be incorporated into knowledge representation languages.

References

- Abiteboul, S. and Duschka, O. (1998). Complexity of answering queries using materialized views. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 254–263, Seattle, WA.

- Adali, S., Candan, K., Papakonstantinou, Y., and Subrahmanian, V. (1996). Query caching and optimization in distributed mediator systems. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 137–148, Montreal, Canada.
- Arens, Y., Knoblock, C. A., and Shen, W.-M. (1996). Query reformulation for dynamic information integration. *International Journal on Intelligent and Cooperative Information Systems*, (6) 2/3:99–130.
- Beeri, C., Levy, A. Y., and Rousset, M.-C. (1997). Rewriting queries using views in description logics. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 99–108, Tucson, Arizona.
- Cadoli, M., Palopoli, L., and Lenzerini, M. (1997). Datalog and description logics: Expressive power. In *Proceedings of the International Workshop on Database Programming Languages*, 281–198.
- Calvanese, D., Giacomo, G. D., and Lenzerini, M. (1999). Answering queries using views in description logics. In *Working notes of the KRDB Workshop*, pages 6–10.
- Catarci, T. and Lenzerini, M. (1993). Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems*, 55–62.
- Chandra, A. and Merlin, P. (1977). Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90.
- Chaudhuri, S., Krishnamurthy, R., Potamianos, S., and Shim, K. (1995). Optimizing queries with materialized views. In *Proc. of Int. Conf. on Data Engineering (ICDE)*, Taipei, Taiwan, 190–200.
- Chaudhuri, S. and Vardi, M. (1993). Optimizing real conjunctive queries. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 59–70, Washington D.C.
- Chaudhuri, S. and Vardi, M. (1994). On the complexity of equivalence between recursive and nonrecursive Datalog programs. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 55–66, Minneapolis, Minnesota.
- Cohen, S., Nutt, W., and Serebrenik, A. (1999). Rewriting aggregate queries using views. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 155–166.
- Cohen, W. (1998). Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 201–210, Seattle, WA.
- Donini, F. M., Lenzerini, M., Nardi, D., and Schaerf, A. (1991). A hybrid system with Datalog and concept languages. In Ardizzone, E., Gaglio, S., and Sorbello, F., editors, *Trends in Artificial Intelligence*, volume LNAI 549, pages 88–97. Springer Verlag.
- Duschka, O. (1997). Query optimization using local completeness. In *Proceedings of the AAAI Fourteenth National Conference on Artificial Intelligence*, 249–255.

- Duschka, O., Genesereth, M., and Levy, A. (1999). Recursive query plans for data integration. *Journal of Logic Programming, special issue on Logic Based Heterogeneous Information Systems*, 43(1):49–73.
- Duschka, O. M. and Genesereth, M. R. (1997a). Answering recursive queries using views. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, 109–116, Tucson, Arizona.
- Duschka, O. M. and Genesereth, M. R. (1997b). Query planning in infomaster. In *Proceedings of the ACM Symposium on Applied Computing*, San Jose, CA.
- Duschka, O. M. and Levy, A. Y. (1997). Recursive plans for information gathering. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, 778–784.
- Etzioni, O., Golden, K., and Weld, D. (1994). Tractable closed world reasoning with updates. In *Proceedings of the Conference on Principles of Knowledge Representation and Reasoning, KR-94*, pages 178–189. Extended version to appear in *Artificial Intelligence*.
- Florescu, D., Raschid, L., and Valduriez, P. (1996). Answering queries using OQL view expressions. In *Workshop on Materialized Views, in cooperation with ACM SIGMOD*, pages 84–90, Montreal, Canada.
- Friedman, M., Levy, A., and Millstein, T. (1999). Navigational plans for data integration. In *Proceedings of the National Conference on Artificial Intelligence*, pages 67–73.
- Friedman, M. and Weld, D. (1997). Efficient execution of information gathering plans. In *Proceedings of the International Joint Conference on Artificial Intelligence, Nagoya, Japan*, 785–791.
- Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., and Widom, J. (1997). The TSIMMIS project: Integration of heterogeneous information sources. *Journal of Intelligent Information Systems*, 8(2):117–132.
- Ives, Z., Florescu, D., Friedman, M., Levy, A., and Weld, D. (1999). An adaptive query execution engine for data integration. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 299–310.
- Klug, A. (1988). On conjunctive queries containing inequalities. *Journal of the ACM*, pages 35(1): 146–160.
- Kwok, C. T. and Weld, D. S. (1996). Planning to gather information. In *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*, 32–39.
- Lattes, V. and Rousset, M.-C. (1998). The use of the CARIN language and algorithms for information integration: the PICSEL project. In *Proceedings of the ECAI-98 Workshop on Intelligent Information Integration*.
- Levy, A. and Rousset, M.-C. (1998). Combining Horn rules and description logics in carin. *Artificial Intelligence*, 104:165–209.
- Levy, A. Y. (1996). Obtaining complete answers from incomplete databases. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, Bombay, India, 402–412.
- Levy, A. Y. (1999). Answering queries using views: A survey. Submitted for publication.

- Levy, A. Y., Fikes, R. E., and Sagiv, S. (1997). Speeding up inferences using relevance reasoning: A formalism and algorithms. *Artificial Intelligence*, 97(1-2).
- Levy, A. Y., Mendelzon, A. O., Sagiv, Y., and Srivastava, D. (1995). Answering queries using views. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 95–104, San Jose, CA.
- Levy, A. Y., Rajaraman, A., and Ordille, J. J. (1996a). Query answering algorithms for information agents. In *Proceedings of AAAI*, pages 40–47.
- Levy, A. Y., Rajaraman, A., and Ordille, J. J. (1996b). Querying heterogeneous information sources using source descriptions. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, Bombay, India, pages 251–262.
- Levy, A. Y., Rajaraman, A., and Ullman, J. D. (1996c). Answering queries using limited external processors. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, 227–237, Montreal, Canada.
- Levy, A. Y. and Sagiv, Y. (1993). Queries independent of updates. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 171–181, Dublin, Ireland.
- Levy, A. Y. and Suciu, D. (1997). Deciding containment for queries with complex objects and aggregations. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 20–31 Tucson, Arizona.
- Litwin, W., Mark, L., and Roussopoulos, N. (1990). Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22 (3):267–293.
- MacGregor, R. M. (1994). A description classifier for the predicate calculus. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 213–220.
- Papakonstantinou, Y., Abiteboul, S., and Garcia-Molina, H. (1996). Object fusion in mediator systems. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 413–424, Bombay, India.
- Pottinger, R. and Levy, A. (1999). A scalable algorithm for answering queries using views. To appear in the *proceedings of the 26th conference on very large databases, VLDB-2000*, Cairo, Egypt, 2000.
- Qian, X. (1996). Query folding. In *Proc. of Int. Conf. on Data Engineering (ICDE)*, pages 48–55, New Orleans, LA.
- Sagiv, Y. (1988). Optimizing Datalog programs. In Minker, J., editor, *Foundations of Deductive Databases and Logic Programming*, pages 659–698. Morgan Kaufmann, Los Altos, CA.
- Sagiv, Y. and Yannakakis, M. (1981). Equivalence among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655.
- Shmueli, O. (1993). Equivalence of Datalog queries is undecidable. *Journal of Logic Programming*, 15:231–241.
- Srivastava, D., Dar, S., Jagadish, H. V., and Levy, A. Y. (1996). Answering SQL queries using materialized views. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 318–329, Bombay, India.

- Srivastava, D. and Ramakrishnan, R. (1992). Pushing constraint selections. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 301–315, San Diego, CA.
- Tsatalos, O. G., Solomon, M. H., and Ioannidis, Y. E. (1996). The GMAP: A versatile tool for physical data independence. *VLDB Journal*, 5(2):101–118.
- Vassalos, V. and Papakonstantinou, Y. (1997). Describing and using query capabilities of heterogeneous sources. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 256–265, Athens, Greece.
- Yang, H. Z. and Larson, P. A. (1987). Query transformation for PSJ-queries. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 245–254, Brighton, England.

Index

- A-Prolog, 260
- Abduction, 17
- Abductive event calculus planning, 235
- Abductive meta-interpreter, 240
- Abductive reasoning, 4, 6, 16
- Abductive sensor data assimilation, 235
- Ability operator, 300
- Absence terms, 415
- ACL2, 26, 547
- Action description, 259
- Action language \mathcal{L}_{ACT} , 282
- Action languages, 14, 149, 151, 163, 187
- Action precondition axioms, 216
- Actions, 215, 282
- Active logic, 370
- Ayclic generalized action theories, 75
- ADL, 151, 205
 - planner, 198
- Adverbials
 - duration, 408
 - location, 408
- Advice oracle, 479
- Advice Taker, 561
- Agent attitudes, 298
- Agent-performed actions, 219
- Agents, 4, 282
 - real world, 303
- Aggregates, 21
- Agre P., 172
- Algorithm EvalLIT, 458–459
- Alternative definition, 558
- Ambiguity tolerance, 44
- AMD Athlon, 569
- AMD K5, 568
- AMD K7 (Athalon), 26
- AMD K7, 569
- Anaphora, 408
- Anbulagan, 174
- Anderson A. R., 290
- Answer set programming, 491
- Answer set, 5, 21
 - planning, 151
 - programming, 273
 - semantics, 110
- Answer sets, 83
- Answering queries using views, 582
- Anti-extensions, 418–419
- Antimonotone operator, 130
- Approximate concepts, 43
- Approximate theories, 43
- Approximating operator on a bilattice, 134
- Approximating operator, 129, 133
- Approximation, 128
- Arrow's impossibility theorem, 346
- Ashby, 321
- Aspectual verb classes, 417
- Assumption-based Truth Maintenance System, 353
- Atelic predicates, 417, 421
- Atelic sentences, 407
- Atemporal predicates, 417–418
- Atom, 82
- Atomic evaluation associated with a KB, 66
- Austel V., 570
- Autoepistemic logic, 7, 128
- Automated reasoning, 59
- Automated theorem proving, 4
- Auton L., 177
- Bacchus-Kabanza style planner, 225, 230
- Bacchus F., 14, 16, 172, 174, 183
- Bakale, 321
- Baltag A., 304
- Bar-Hillel Y., 149
- Baral C., 6, 15, 17
- Barwise J., 408
- Basic action theories, 68
- Bayardo Jr. R., 174
- Bayes, 319
- BCF
 - see Blake Canonical Form, 67
- Belief content, 371
- Belief revision, 17
- Belief, 4, 361, 382

- Benigni, 321
- Bilattice, 131
- Bill of Materials, 535
- Binary
 - direct dependency relation, 287
- Bisimulation, 460, 567
- BlackBox, 20–21
- Blackbox, 22–25
- BlackBox, 169, 513
- Blake Canonical Form (BCF), 67
- Blum A., 170–171
- Boolean circuit, 482
- Boolean formula, 471
- Bounded informatic situation, 39
- Boutilier C., 14, 338
- Boyer-Moore NQTHM prover, 26
- Boyer R., 548, 550, 571
- Brave reasoning, 90
- Brock B., 568, 571
- Brooks R. A., 251
- Bucket algorithm, 584
- Built-in function, 507
- Buvač S., 43
- Byte-code verifier, 570
- Can operator, 300
- Carbonell J., 172
- Cardinality constraint, 492, 497
 - lower bound, 497
 - satisfaction, 497
 - upper bound, 497
- CASE, 321
- Causal Calculator
 - see CCALC, 149
- Causal logic, 158, 162
- Causal relations, 407, 409, 423, 437
- Causal theories, 20
- Causation, 14
- Causes, 411, 416
- Cautious monotonicity (CM), 344
- Cautious reasoning, 90
- CCALC, 20–25, 158, 513
 - planner, 205
- Chaining rules, 431
- Chapman D., 172
- Checking part, 84
- Cheeseman P., 183
- CHILL system, 324
- Choice model, 526
- Choice program, 526
- Circumscription, 7, 47, 162, 237, 389
- Closed under a set of rules, 496, 498
- Closed world assumption, 153–154, 162
- Closed-world database, 59
- Closed-world reasoning, 60
- CLP
 - see Constraint logic programming, 5
- CNF
 - see Conjunctive normal form, 63
- Coalescing temporal intervals, 532, 538
- COBOL, 570
- Cognitive robotics, 8, 16
- Cohen B., 170
- Cohen R., 570–571
- Colmerauer A., 5–6
- Common Lisp, 550
- Common sense informatic situation, 40
- Commonsense law of inertia, 153, 155, 157
- Commonsense physics, 41
- Commonsense reasoning, 4, 9
- COMPACT, 321
- Compatibilism, 48
- Compilability, 475
- Compilation scheme, 474
- Complete approximation, 130
- Complete lattice, 130
- Complete
 - logical, 62
- Completion, 162
- Composition
 - parallel, 283
- Computational logic, 4, 19
- Computational tractability, 59
- Concurrent actions, 8, 154
- Conditional effect, 471
- Conditional literal, 506
- Conflict-free literals, 68
- ConGolog, 251
- Congruence, 552
- Conjunctive aggregation of fuzzy preference profiles, 349
- Conjunctive normal form, 63, 348, 471
- Conjunctive query, 576
- Conjunctivity, 43
- Consciousness, 10
- Consequence, 6
- Consequent, 109
- Conservative extensions, 43
- Consistency
 - strong, 318
 - weak, 318
- Consistent operator on a bilattice, 134
- Consistent pair of elements, 131
- Constraint logic programming, 5, 494
- Constraint Satisfaction Problem, 206
- Constraint satisfaction problem, 494
- Constraints
 - temporal, 240
- Context, 382
- Contexts, 43
- Continuous actions, 8
- Continuous time, 16, 213
- Correct
 - logical, 62
- Corroboration (of models), 564, 570
- Cowles J., 571
- CProgol, 21–25
- Crawford J., 177
- Creativity, 52

- Crisp goals, 347
CSP
 see Constraint Satisfaction Problem, 206
CSR
 see Commonsense reasoning, 4
CTL, 450
CTL*, 450
CTLog, 460
Cut, 345
Cyc, 41
Data integration systems, 576
Data integration, 4, 26, 575
Databases
 geographic, 324
Datalog LITE program, 455
Datalog LITE, 443
Datalog LIT, 451
Datalog_{IS}, 535
Davidson D., 152, 407–411, 413–414, 416, 420, 423, 429, 434, 436
Davidson D.
 neo-Davidsonianism, 409, 435
Davis-Putnam procedure, 511, 513
Davis-Putnam-Loveland, 177
Davis E., 10
DDBs
 see Deductive databases, 5
DDL
 dynamic database logic, 288
De Giacomo G., 11
Deductive closure, 496, 498
Deductive Database System, 21, 523
Deductive databases, 5–6
Deductive planning, 150
Deep Space 1, 26
Default Logic, 11, 47
Default Logic, 108–109
Default logic, 128, 162, 494
Default reasoning, 7
Default rule, 109
 generating, 110
Default theory, 109
 ordered, 113
 set-ordered, 118
Definite program, 495
Deictic pronouns, 408, 414
Deletion
 passive and active, 286
Delgrande J., 11
Deliberative, 268
Denecker M., 12
Dennett D., 39
Deontic logic, 17, 290
Deontic operators, 291
Deontic properties, 291
DEREK, 321
DeReS, 12, 21–25, 81, 111, 494
Description logics, 575, 589
Descriptions of data sources, 578
Design stance, 48
Diagnosis, 118
 network, 91
Dignum F., 305
Dijkstra weakest precondition calculus, 294
Discounting a preference profile, 349
Discourse entities, 408
Discrimax likelihood relation, 355
Discrimax likelihood relations, 344
Disjunctive databases, 79
Disjunctive Datalog program, 83
Disjunctive deductive databases, 6
Disjunctive logic program, 494
Disjunctive logic programming (DLP), 6
Disjunctive logic programming, 79
Disjunctive normal form (DNF), 63
Disjunctive normal form, 348, 471
Distributed computing, 565
Divide-and-conquer, 187
DLV system, 11, 170
DLV, 12, 21–25, 79, 111, 494, 513
DNF, 63
Doherty P., 172
Domain knowledge, 169
Domain predicate, 504
Domain-independent planning, 169
Domain-restricted rule, 504
Dynamic binding, 410
Dynamic causal laws, 14, 262
Dynamic deontic logic, 290
Dynamic laws, 154
Dynamic logic, 17, 281
 propositional, 282
Dynamic update logic, 285
E-commerce, 570
Early returns, 532
ECLIPSE 3.5.2
 ECRC Common Logic Programming System, 225
ECLIPSE Prolog, 213
EDS Inc., 570
Eiter T., 11, 170
Elaboration tolerance, 42
Element of lattice approximated by a pair, 130
Episodic Logic, 407, 409, 437
 EL, 19
 quantifiers in, 409
Epistemic group updates, 304
Epistemic logic, 301, 384
Epistemologic problems, 4
Epistemological problems of logical AI, 38
Epistemologically adequate languages, 40
Epistemology, 38
Equality axioms, 64
Equivalence, 557
Equivalent rewritings, 583
Ernst M., 174, 177
Etzioni O., 172, 176

- Event calculus hierarchical planning, 235
- Event calculus, 16, 235
- Event predicates, 408
- Events, 407, 437
 - culminating, 417
- Eventualities, 407, 437
 - in Davidsonian event theories, 414
 - in Davidsonian event theories, 434
- Executability, 551, 564, 568–571
- Exogenous logic, 306
- Expert systems, 40
- Explanation, 267
- Explicitly locally stratified programs, 536
- Expressive power, 469
- Extended disjunctive logic programs, 80
- Extended logic program, 110, 123
- Extending operator on a bilattice, 134
- Extending operator, 132
- Extension, 109
 - existence of, 117
- Extreme oscillating pair, 131
- EPILOG, 437
- Facts, 408, 415–416
- Family of Boolean circuits, 482
- Fikes R., 7, 172, 188
- First-order logic, 108, 416, 436
- Fischer M., 282
- Fitting M., 9, 12
- Fixed plan-size initial-state existence problem, 478
- Fixpoint, 128
- Flatau A., 569, 571
- Floating point division, 568
- Fluent predicates, 416, 422
- Fluent predictions, 418
- Fluent, 68, 215
- FOL**, 19, 407, 437
 - models for, 419, 421
 - rules of inference, 431
 - truth in, 421, 423
 - valid formulas, 423
- Fox M., 183
- Frame axioms, 4, 7
- Frame problem, 4, 7, 49, 69, 149, 153, 162, 237, 293
- Frames, 10
- Frank J., 183
- Free will actions, 213
- Free will, 16, 48
- FStrips
 - see Functional Strips, 195
- Functional dependencies, 589
- Functional Strips, 15, 187
- Furst M., 170–171
- Fuzzy logic, 13
- Fuzzy set, 347
- Gaifman graph, 457
- Gallaire H., 6
- Geffner H., 14
- Gelfond-Lifschitz transform, 83
- Gelfond M., 6, 8, 14–15, 17
- Generalized literal, 455
- Generate-and-test paradigm, 81
- Generic sentences, 417
- Genericity, 529
- Gerevini A., 183
- GF, 462
- Gil Y., 172
- Ginsberg M., 14
- Giunchiglia G., 14
- Global as view (GAV) source description, 581
- Global schema, 577
- Global variable, 506
- GnT, 98
- Goal operator, 300
- Goal specification, 472
- Golem
 - ILP system, 320
- GOLOG interpreter, 223
- Golog language, 8
- Golog, 16, 20, 22–25
- GOLOG, 205, 222, 250–251
- Gomes C., 174
- Gottlob G., 19
- GPT, 20, 22–25
- Graph coloring, 492, 505, 507
- Graphical user interface (GUI), 94
- Graphplan, 170, 189, 193
- Greatest lower bound, 130
- Green C. C., 7, 222, 250
- Greve D., 570–571
- Gripper domain, 202
- Gripper, 189
- Groundedness, 110
- GSAT, 60, 172
- GSFA
 - see guarded sensed fluent axiom, 72
- GSSA
 - see guarded successor state axiom, 72
- Guard, 452, 551
- Guarded Datalog rule, 452
- Guarded fixed point logic, 462
- Guarded fragment, 462
- Guarded sensed fluent axiom (GSFA), 72
- Guarded successor state axiom (GSSA), 72
- Guess and Check-paradigm, 11, 79
- Guessing part, 84
- GUI, 94
- Haas A., 8
- Hamiltonian cycle, 510, 513
- Hamiltonian path, 11, 86
- Hanks S., 8
- Hardin D., 570–571
- Hardware verification, 26
- Harel D., 282
- Hayes P., 5
- HCF, 96

- Head-cycle-free (HCF), 96
 Hendler J., 183
 Herbrand literal base, 83
 Herbrand Theorem, 64
 Herbrand universe, 64, 83
 Heuristic planners, 193
 Heuristic problems of logical AI, 38
 Heuristics, 45
 High-level robotics, 4
 History, 73
 Hoare Logic, 282, 307
 Hobbs J., 434
 HOL, 549, 571
 Homogeneous predication, 417, 419, 421
 Horn clauses, 5
 Horn constraint rule, 498, 501
 HPATH
 see Hamiltonian path, 86
 IISP
 heuristic search planners, 204
 Huang Y.-C., 170, 183
 Hunt W., 571
 IBM 4758, 570
 IEEE 754, 568
 IG, 96
 IMPACT, 21–25
 Incomplete knowledge, 59
 Inconsistent beliefs, 363, 367, 369
 Inconsistent, 197
 Inductive logic programming, 21
 ILP, 315
 Inductive reasoning, 4
 Inference
 goal-driven, 437
 input-driven, 437
 Infons, 409
 Information ordering of L^2 , 131
 Inheritance, 91
 property, 119, 121
 Initial state, 472
 Insertion
 passive and active, 286
 Integrity constraint, 492, 497, 508
 Intelligent agent architecture, 267
 Intelligent agents, 257
 Intelligent grounding module (IG), 96
 Intentional stance, 48
 Interleaving semantics, 565
 Interpretation, 191
 standard, 64
 Inverse rule, 586
 Inverse-rule algorithm, 586
 Iterated choice fixpoint, 528
 Iterative sampling, 11
 Java Virtual Machine, 26
 Java, 561, 570
 Join semilattice, 407, 410
 Joseph R., 172
 Joslin D., 177
 Justification, 109
 JVM, 561, 570
 Kabanza F., 16, 172, 174, 183
 Kahn D., 172
 Kambhampati S., 176
 KARO logic, 301, 305
 Katukam S., 176
 Kaufmann M., 548, 569–571
 Kautz H., 170, 174, 176–177, 183
 Khepera
 miniature robot, 234
 robot, 16, 241
 Knapsack problem, 493
 Knoblock C., 172, 176
 Knowledge compilation, 470
 Knowledge representation, 10, 407
 KR, 4, 6
 Kowalski R. A., 5
 Kozen D., 307
 KR
 see Knowledge representation, 4
 Kraus S., 344
 Kripke model for \mathcal{L}_{SA} , 299
 Kripke model
 full, 285
 Kripke Structure, 449
 Kripke-Kleene fixpoint, 128
 Kripke-model, 283
 Kvarnstrom J., 172
 Ladner R., 282
 Lambda calculus, 5
 Lattice ordering of L^2 , 131
 Lattice, 128, 130
 LBAI
 Logic-Based Artificial Intelligence, 3
 LDL⁺⁺, 494
 LDL⁺⁺, 21–25
 Least upper bound, 130
 Legal reasoning, 292
 Lehmann D., 344
 Levesque H., 11, 172
 Levy A., 26
 Lifschitz V., 5, 7–8, 14, 17, 170, 491
 Lifting principle, 342
 Likelihood relation, 341
 Linear constraint solver, 213
 Lin F., 14
 Lisp, 5, 550
 Literal completion, 162
 Literal, 82, 471, 493, 497
 classical, 82
 complementary, 82
 negation as failure, 82
 positive, 82
 Lits(S), 65
 Li C.M., 174
 LLL
 learning language in logic, 327
 LNTPC, 221

- Local as view (LAV) source description, 581
- Local variable, 506
- Locality invariance, 457
- Logic and constraints, 4
- Logic and language, 4
- Logic for actions, 4
- Logic for causation, 4
- Logic in computer science, 39
- Logic programming, 47, 162–163
 - LP, 5
- Logic-Based Machine Learning (LBML), 315
- Logical AI, 37, 39
- Logical form, 408, 416, 436
- Logical language L_{DL} , 282
- Logical language L_{DL} , 282
- Logistics planning domain, 170
- Long D., 183
- Loveland D., 26
- LP
 - see Logic Programming, 5
- Lynch T., 569, 571
- Machine learning, 315
- Magidor M., 344
- Mally E., 290
- Manolios P., 571
- Marginean F., 13
- Materialized view, 577
- Mathematical logic, 38
- Maximally-contained rewritings, 583
- Maximization statement, 503
- McAllester D., 176
- MCC, 523
- McCain N., 14
- McCarthy J., 4, 7, 9, 11, 59, 148, 315, 547
- McCune W., 5
- McDermott D., 8
- Meaning postulates, 410
- Mechanical checker, 26
- Mechanical checking, 4
- Mediated schema, 577
- Mental situation calculus, 48
- Merge sort, 556, 559
- Meyden R. van der, 292
- Meyer J.-J., 17
- Microcode, 568
- Microprocessor verification, 26
- Microprocessor, 548
- Millstein T., 174, 177
- MiniCon algorithm, 588
- Minimization statement, 503
- Minker J., 6, 183
- Minsky M., 6
- Minton S., 172, 176
- Mitchell D., 172
- ML, 449
- Modal Datalog, 460
- Modal logic, 382
- Modal Logic, 449
- Modal logics, 282
- Modal operators, 429
- Modal, 365
- Model checker, 96
- Model checking, 567
- Model generator, 96
- Model-theoretic planning, 270
- Modeling, 187
- Monadic Datalog rule, 452
- Monotone aggregation, 533
- Monotone operator, 130
- Monotonic aggregates, 531
- Monotonic operator W_P , 96
- Moore J., 26, 548, 550
- Moore R., 282, 434
- Motorola CAP digital signal processor, 26
- Motorola CAP, 567
- Muggleton S., 13
- Multi-agent systems, 382
- Multiple agents, 17, 21
- Muscettola N., 172
- Muskens R., 409
- Muti-agents, 303
- Mutual dependency group, 287
- Mutuality, 375
- NAF
 - negation as failure, 82
- Name, 111, 114
- Namjoshi K., 571
- Narrative, 50
- Narratives, 411, 437
- Natural actions, 16, 213, 219
- Natural language competence, 18
- Natural language processing, 18
- Natural language understanding, 18, 407, 409, 436–437
- Nau D., 172
- Nayak P., 172, 177
- Nebel B., 19
- Necessity ordering, 354
- Necessity, 318
- Negation as failure, 162
 - Fitting's theory, 12
- Negation
 - action, 283
 - strong, 80
 - default, 5
 - stratified, 5
- Newell A., 39
- Nicolas J.-M., 6
- Niemela I., 21, 170
- Nilsson N., 7, 172, 188
- NLP
 - natural language processing, 322
- NMR
 - see Nonmonotonic reasoning, 4
- Non-uniform complexity class, 479
- Nondeterminism, 526, 565
- Nonmonotonic logic, 108

- Nonmonotonic reasoning, 11, 46, 79, 153, 162, 333
 NMR, 4
 Normal logic program, 493, 495, 497
 NP-complete, 63, 478
 NP-SPEC, 494
 NQTHM, 548
 Obligation, 290
 Observation, 260
 Occurrences, 417
 Omniscience, 363
 Ontologies, 10
 Ontology, 39, 45
 Open-world database, 59
 Open-world reasoning, 60
 Operator, 128, 130
 parallel, 296
 preference, 296
 Opportunity operator, 300
 Optimistic criterion, 336
 Optimization problems, 503
 Oracle, 81
 Oracle, 565
 Ordered list, 556
 Oscillating pair for an operator, 131
 Otter, 549, 571
 Parallelism, 565
 Parsimony, 119
 Parsons T., 409
 Part-of ordering, 407, 410, 418–419, 424
 Partial worlds, 409
 Path formula, 450
 PDDL, 190
 Propositional Dynamic Database Logic, 285
 Pednault E., 8
 Peirce C. S., 12
 Penberthy J., 172
 Peot M., 176
 Perception, 233
 Perez A., 172
 Permission, 290
 Permutation, 557
 Perry J., 408
 Persistence of information, 407, 410, 417, 419, 421, 423, 430–431
 Pessimistic axioms, 339
 Pessimistic criterion, 336
 Physical stance, 48
 Pirri F., 16–17
 Plan existence problem, 473
 Plan, 472
 according abductive definition, 239
 Planning as satisfiability, 169
 Planning instance, 472
 Planning operator, 471
 Planning, 4, 20, 50, 161, 169, 187, 233
 domain-independent, 203
 hierarchical, 249
 with natural actions in the situation calculus, 221
 Plotkin G., 315
 Pollack M., 183
 Polynomial advice, 479
 Poor entities, 42
 Possibilistic constraints, 351
 Possibilistic knowledge base, 347
 Possibilistic logic, 13, 333
 Possibilistic measure, 345
 Possibility measure, 354
 Possibility ordering, 354
 Possibility theory, 334
 Possible intended operator, 300
 Possible model semantics, 494
 Postcondition, 471
 Practical possibility operator, 300
 Pratt V., 282
 Precondition, 471
 Predicate completion, 240
 Predicate, 82
 Preference, 13, 113, 334
 dynamic, 118
 set, 118
 static, 114
 Preferential nonmonotonic relation, 345
 Prerequisite, 109
 Primitive actions, 216
 Privacy axiom, 304
 Probabilistic reasoning, 46
 Probability and decision theory, 13
 Problem solving, 4, 187
 Progol, 321
 ILP system, 320
 Prohibition, 290
 Projection task, 68
 Projection, 50
 Prolog II, 6
 Prolog, 5
 Proper formula, 65
 Propositional atom, 471
 Propositional satisfiability, 508–509, 513
 Propositional solvers, 158, 163
 Propositions, 415, 434
 PSPACE, 473
 PSPACE-complete, 63
 PVS, 549, 571
 Qualification constraints, 155
 Qualification problem, 49, 293
 Qualitative decision theory, 335
 Qualitative possibility ordering, 354
 Query containment, 577
 Query equivalence, 577
 Query language, 266
 Query optimization, 579
 Query reformulation, 579
 Quine W.V.O., 45
 Qu Y., 176

- Rabinov A., 8
 Ramification problem, 10, 50, 153, 155, 162, 293
 Ramsey axiom, 304
 RASH, 321
 RAUL
 relational algebra update logic, 288
 RCPS
 resource constrained project scheduling problem, 229
 Reactive, 268
 Realisability operator, 300
 Recursive rewritings, 588
 Reduct, 83, 495, 499, 502
 Regression, 68
 Reichenbach H., 408, 410–411, 429, 436
 Reification, 45
 Reilly S., 172
 Reiter R., 8, 16–17, 162, 172, 215
 Resolution principle, 4
 Resolution-based theorem proving, 233
 Resources, 202
 Rewriter module, 93
 Rich entities, 42
 Right weakening, 344
 Robinson resolution principle, 4
 Robinson J. A., 4
 Robins H., 5
 Robot, 233
 example, 301
 Robotics, 4
 Rockwell-Collins Avionics JEM1, 570
 Rockwell-Collins JEM1 microprocessor, 26
 Roy A., 177
 RTL, 569
 Rubik's cube, 203
 Rule body, 497
 Rule head, 497
 Rule, 82
 disjunctive, 82
 Russinoff D., 569, 571
 Sacerdoti E. D., 172
 Safe Datalog, 21
 Sandewall E., 10
 SAT encodings, 172
 SAT
 planner, 189, 193, 205
 Satisfiability planning, 150, 158, 163
 Satisfiability, 169
 Sato system, 513
 SATPLAN, 21, 169
 Satz system, 513
 Savage decision theory, 340
 Sawada J., 571
 Schaub T., 11
 Schema, 576
 Schrag R., 174
 Schubert L., 8, 19, 183
 SCOP (Structural Classification of Proteins), 322
 Scripts, 10
 Secure coprocessor, 570
 Select-project-join queries, 584
 Selman B., 170, 172, 174, 176–177, 183
 Semantic networks, 10
 Semantic representation, 407
 Semantics
 declarative, 5
 fixpoint, 5
 procedural, 5
 stable model, 5
 stratified, 5
 well-founded, 5, 21
 Sense-model-plan-act architecture for planning, 8
 Sensing functions, 71
 Sensing information, 59
 Sensor-fluent formula, 71
 Separable, 67
 Shakey, 7, 233, 250
 Shanahan M., 16–17
 Shapiro E., 315
 Shared memory, 565
 Sierra C., 289
 Sierra J., 45
 Simons P., 21
 Situation abstract, 432
 Situation abstracts, 417, 419
 support sets for, 422–423, 425–427, 430–431, 433, 436
 Situation calculus, 4, 49, 68, 150–151, 213–214
 Situation calculus, 417
 Situation consistency, 432
 Situation Semantics, 407–408, 410–411, 413, 416–417, 423
 Situation, 215
 Situations, 407, 437
 and absence terms, 415
 atelic descriptions of, 407, 410, 417, 419, 421–423, 425
 causal relations between, 411, 416, 423, 437
 characterizing descriptions of, 407, 409, 419, 421–424, 431, 436
 complex descriptions of, 407, 413, 416, 426, 430
 complex, 423
 concurrent parts of, 407, 418–419, 424
 conjunctive, 412–414, 426–428
 homogeneous descriptions of, 430
 individuation of, 429
 join semilattice of, 410, 416, 419, 424
 joins of, 407
 negative, 410, 413, 416, 420, 434–436
 partial descriptions of, 408, 413, 418, 421–424, 434, 436

- quantified, 413–414, 416, 426–428, 435
 semilattice of, 410, 430–431
 telic descriptions of, 407, 410, 417,
 422–423, 425
 temporal segments of, 407, 410, 418–419,
 424
 transparency in descriptions of, 410, 427,
 429–430
Skolem constants and anaphora, 410
Skolem functions, 64
Skolem-Lowenheim Theorem, 64
 Slaney J., 172
SLDNF, 240
 Smith D., 14, 176
Smodels system, 170
Smodels, 12, 21–25, 80, 111, 494, 503–504,
 511, 513
SMV, 444
Softbots
 software agents, 298
Sorting algorithm, 556
Sound
 logical, 62
Space platform example, 16
Spanning tree, 529
Specificity, 119–120
Specifying the history, 265
SQL3, 93
Stable model semantics, 491, 493
Stable model, 21, 128, 496, 499, 507
Stable operator, 136
Stanford GraphBase, 513
State formula, 450
State model, 198
State representations, 188
State, 157, 188, 417, 471
State-based encodings, 172
State-space model, 190
Statelog, 536
Static causal law, 262
Static laws, 153
Sternberg M., 322
STRATCOMP, 87
Strategic set, 87
Strict partial order, 113
STRIPS, 7, 20, 151, 172, 214, 469
Strips
 language, 190
STRIPS
 planner, 250
Strips
 see STRIPS, 187
Stuttering, 567
Stutz J., 183
Successor state axioms, 216
Sufficiency, 318
Sumners R., 571
Supported model, 128
Symmetric operator on a bilattice, 132
Syntax-semantics interface, 410
System P, 345
Tabling, 21
Telic predicates, 417
Telic sentences, 407
Temporal logic, 307
Temporal reasoning, 213
Term, 82
Thematic roles, 409, 435
Thiebaux S., 172
Thresholding a preference profile, 349
Time, 151, 419
TIPTS, 321
Tiuryn J., 307
TLPlan, 20, 22–25, 189, 204–205
TOPKAT, 321
Tower of Hanoi, 198
Transition diagram, 262
Transition system, 154, 157, 163
Transitive closure, 506
Turing A., 315
Turner H., 14
Uncertainty, 334
Understanding, 45
Unification algorithm, 4
Uniform families of Boolean circuits, 482
Unique names axiom, 246
Unique names axioms for actions, 216
Unique-name assumption, 64
Useful counterfactuals, 44
User-defined aggregates, 532
Utility function μ_C , 346
Van Emden-Kowalski operator, 128
Van Emden M. H., 5
Van Linder B., 305
Veloso M., 172
Vere S., 172
Vertex cover, 492, 503
View, 575, 577
Vivid reasoning, 60
Vivid representation, 11
Von Wright G. H., 290
WalkSat, 170, 172, 177
Walksat, 513
Wang X., 21, 172
Weak constraint, 494, 504
Weak derivation, 432
Weight constraint, 493, 500
 local weight function, 501
 lower bound, 500
 satisfaction, 501
 upper bound, 500
Weight rule, 493, 495
 computational complexity, 503
 satisfaction, 497
Weld D., 172, 174, 177, 183
Well-founded fixpoint of an operator, 138
Well-founded model, 128, 536
Well-founded semantics, 494, 513

- Well-founded vertex, 453
Wilding M., 570–571
Wilensky R., 435
Williams B., 172, 177
Wrappers, 579
XRay, 111
XSB, 12, 21–25, 494
XY-stratification, 535
XY-stratified choice programs, 539
XY-stratified programs, 21
Y2K, 570
Yale Shooting Problem, 8, 49
Zaniolo C., 21