Book Reviews

Abduction, Reason, and Science: Processes of Discovery and Explanation—A Review

Atocha Aliseda

Broadly speaking, abduction is a reasoning process invoked to explain a puzzling observation. There are however, a variety of different approaches that claim to capture the true nature of this concept. One reason for this diversity lies in the fact that abductive reasoning occurs in a multitude of contexts. It concerns cases that cover the simplest selection of already existing hypotheses to the generation of new concepts in science. It also concerns cases where the observation is puzzling because it is novel versus cases in which the surprise concerns an anomalous observation. For example, if we wake up, and the lawn is wet, we might explain this observation by assuming that it must have rained or that the sprinklers have been on. This is a practical setting found in our day-to-day commonsense reasoning when a novel phenomenon is needed for an explanation. Abduction also occurs in more theoretical scientific contexts. For example, it has been claimed (Hanson 1961; Peirce 1958) that Johannes Kepler's great discovery that the orbit of the planets is elliptical rather than circular was a prime piece of abductive reasoning. What initially led to this discovery was his anomalous observation that the longitudes of Mars did not fit circular orbits. Moreover, before even dreaming that the best explanation involved ellipses instead of circles, he tried several other forms. Kepler had to make some other assumptions about the planetary system, without which his discovery does not work. His heliocentric view allowed him to think that the sun, so near to the center of the planetary system, and so large, must somehow cause the planets to move as they do. In addition to this strong conjecture, he also had to generalize his findings for Mars to all planets by assuming that the same physical conditions could be obtained throughout the solar system.

Research on abduction in AI dates back to the 1970s, but it is only fairly recently that it has attracted great interest in areas such as logic programming, knowledge assimilation, and diagnosis. It has been a topic of several workshops at AI conferences (1996, 1998, 2000 European Conference on Artificial Intelligence; 1997 International Joint Conference on Artificial Intelligence) as well as model-based reasoning conferences (1998, 2001 Model-Based Reasoning Conference). It has also been at the center of recent publications (Flach and Kakas 2000; Josephson and Josephson 1994). In all these places, the discussion about the different aspects of abduction has been conceptually challenging but also shows a (terminological) confusion with its close neighbor, induction.

Abduction, Reason, and Science: Processes of Discovery and Explanation, Lorenzo Magnani, New York, Kluwer Academic/Plenum Publishers, 2001, 205 pages, ISBN 0-306-46514-0 (hardback).

This book contributes to the rapidly growing literature on the topic of abductive reasoning. By placing abduction at the heart of the foundations of AI from philosophical, cognitive, and computational perspectives, the author makes us aware that abduction is not at all a new topic of research. In addition, by introducing fine distinctions in abductive kinds, it shows its relevance as a recent topic of research in all these fields.

The importance of abduction has been recognized by leading researchers in all relevant fields. Although for Jaakko Hintikka, abduction is the fundamental problem of contemporary epistemology, in which abductive inferences are assembled as "answers to the inquirer's explicit or (usually) tacit question put to some definite source of answers (information)" (p. 129), for Herbert Simon, the nature of the retroductive process (another term for abduction) "is the main subject of the theory of problem solving" (p. 16). For Paul Thagard, several kinds of abduction play a key role as heuristic strategies in the program PI (for "processes of induction"), a working system devoted to explaining in computational terms the main problems of philosophy of science, such as scientific discovery, explanation, and evaluation (p. 49).

In agreement with other current approaches to abduction, for the author of this book, there are two main epistemological meanings of the word abduction: (1) abduction that only generates plausible hypotheses and (2) abduction as inference to the best explanation that also evaluates them to further obtain the best one. In this book, the first meaning is further divided into selective or creative. Selection takes place in contexts such as medical diagnosis, in which the task is to select a diagnosis from a precompiled set of diagnostic entities. Creativity is present in issues such as the discovery of a new disease. The latter meaning, abduction as inference to the best explanation, is described by the complete abduction-deduction-induction cycle, represented in an epistemological model for diag-

nostic reasoning (ST-MODEL) in which selective abduction introduces a set of plausible hypotheses, followed by deduction to explore their consequences and induction to test them, to either increase their likelihood or refute all but one. Even though the selection and evaluation phases of abduction are in fact integrated in cognitive models of human thought, it is nevertheless a useful methodological distinction for the development of real artificial reasoning systems.

Moreover, Magnani proposes an interesting distinction between theoretical abduction and manipulative abduction. Although theoretical abduction "is the process of reasoning in which explanatory hypotheses are formed and evaluated" (p. 18), and can, in turn, be sentential or model based (to be explained later), manipulative abduction is action based, the case in which "the suggested hypotheses are inherently ambiguous until articulated into configurations of real or "imaginated" [SIC] entities (p. 54). The interplay of these two aspects "consists of a superimposition of internal and external, where the elements of the external structures gain new meanings and relationships to one another, thanks to the constructive explanatory theoretical activity" (p. 59).

Magnani offers an impressive and comprehensive overview of the logical approaches to abduction, covering the deductive model, the abductive logic programming paradigm, and approaches linking abduction to the well-known framework of belief revision, all of which are considered as cases of theoretical sentential abduction, given their logical character. His conclusion is that all these logical approaches deal primarily with the selective and explanatory aspects of abduction, leaving aside many other creative processes, such as conceptual change, an essential ingredient to study revolutionary changes in science. Thus, "we have to consider a broader inferential one (view) which encompasses both sentential and what I call model-based sides of creative abduction" (p. 36). (The term *model-based reasoning* is used here to indicate the construction and manipulation of various kinds of representations, not necessarily sentential or formal).

This book presents as well applications of the several distinctions of abduction introduced earlier. For example, the case of medical diagnostic reasoning is described as an instance of theoretical abduction, showing that its machinery fits the epistemological model mentioned earlier; the patient data are abstracted and used to select hypotheses, and these hypothetical solutions in turn provide starting conditions for the forecast of expected consequences, which are later compared to the patient's data to evaluate (corroborate or eliminate) those hypotheses that come from deduction. A particular application is the system NEOANEMIA, a working diagnostic system for disorders such as anemia developed at the home university of the author (Pavia). Visual and temporal aspects of abduction are also taken into account and presented as cases of model-based creative abduction. This perspective highlights the importance of both spatial reasoning and anomaly resolution for hypotheses generation and scientific discovery. The concrete example is the discovery of non-Euclidean geometries, about which it is argued that some of the hypotheses created by Lobachevsky were indeed image based, something that helped to deal with the fifth parallel postulate by manipulation of symbols. Interestingly, the author claims that the anomalous (or problematic) aspect of the Euclidean fifth postulate lies in that "we cannot draw or 'imagine' the two lines at infinity" (p. 165), which, in contrast to the rest of the postulates, is empirically unverifiable and, thus, opens the door to the possibility of creating alternative geometries to that of Euclid.

To summarize, in the first three chapters of this book, certain frameworks for the various facets of abductive reasoning are put forward, covering the selection of explanations as well as their evaluation from the standpoint of theoretical abduction (chapter 2) and manipulative abduction (chapter 3). In the following chapters, various applications are presented in the area of diagnosis (chapter 4) as well as in discovery in science with special emphasis on visual and temporal aspects of abduction (chapter 5). The rest of the chapters are dedicated to the analysis of the role of inconsistencies in scientific discovery (chapter 6) and the makeup of hypotheses withdrawal in science (chapter 7).

Abduction, Reason, and Science is a book for those interested in the subject of discovery who are willing to get an integrated picture from philosophy, cognitive science, and computer science, all disciplines concerned with the question of creative reasoning. A beginner can get a pretty good idea of the recent status of research on abduction, and at the same time, it pleases the expert by informing him/her about approaches in other fields and offering a detailed and interesting notice about the original views of its author, Lorenzo Magnani.

I found it interesting that the ST-MODEL resembles the American pragmatist Charles S. Peirce's complete framework of abduction, which considers not only its inferential logical structure (which is what most approaches take) but also two further aspects, namely, (1) testing and (2) economy. I think that it has become quite clear by now—and this book makes a strong point in this direction —that the study of the role of testing in abduction is fundamental for its understanding.

References

Flach, P., and Kakas, A., eds. 2000. *Abductive and Inductive Reasoning: Essays on Their Relation and Integration.* Applied Logic Series. New York: Kluwer Academic.

Hanson, N. R. 1961. *Patterns of Scientific Discovery.* Cambridge, U.K.: Cambridge University Press.

Josephson, J., and Josephson, S., eds. 1994. *Abductive Inference: Computation, Philosophy, Technology.* New York: Cambridge University Press.

Peirce, C. S. 1958. *Collected Papers of Charles Sanders Peirce, Volumes 1–6,* eds. C. Hartshorne and P. Weiss. *Volumes 7–8,* ed. A. W. Burke. Cambridge, Mass.: Harvard University Press.

Atocha Aliseda (atocha@servidor.unam.mx) is a full professor at the Institute for Philosophical Research of the National Autonomous University of Mexico and "breedtestrategiepostdoc" at the Faculty of Philosophy in Groningen, The Netherlands. She obtained her Ph.D. from Stanford University in 1997. Her main research interests are abductive logic and the connection between philosophy of science and AI.

The Logic of Knowledge Bases A Review

Enrico Motta

A knowledge-based system (KBS) contains (by definition) an explicitly codified body of knowledge, which causally determines its behavior. Hence, at a coarse-grained level of abstraction, KBSs can be characterized in terms of two components: (1) a knowledge base, encoding the knowledge embodied by the system, and (2) a reasoning engine, which is able to query the knowledge base, infer or acquire knowledge from external sources, and add new knowledge to the knowledge base. Levesque and Lakemeyer's *The Logic of Knowledge Bases* deals with the "internal logic" of a KBS: It provides a formal account of the interaction between a reasoning engine and a knowledge base. Clearly, this analysis is not the same as providing a formal account of the behavior of a KBS as a whole. A knowledge-level account of a KBS (that is, a competence-centered, implementation-independent description of a system), such as Clancey's (1985) analysis of first-generation rule-based systems, focuses on the task-centered competence of the system; that is, it addresses issues such as what kind of problems the KBS is designed to tackle, what reasoning methods it uses, and what knowledge it requires. In contrast with task-centered analyses, Levesque and Lakemeyer focus on the competence of the knowledge base rather than that of the whole system. Hence, their notion of competence is a task-independent one: It is the "abstract state of knowledge" (p. 49) denoted by the contents (implicit or explicit) of a knowledge base at any particular time in its life cycle. This is an interesting assumption, which the "proceduralists" in the AI community might object to: According to the procedural viewpoint of knowledge representation, the knowledge modeled in an application, its representation, and the associated knowledge-retrieval mechanisms have to be engineered as a function of the task in hand; for example, see Bylander and Chandrasekaran's (1988) discussion on the interaction hypothesis. As a result, they would argue, it is not possible to discuss the knowledge of a system independently of the task context in which the system is meant to operate. I won't go into too many details here because a detailed discussion of the declarative versus the procedural argument is well beyond the scope of this review. The important point to make is that Levesque and Lakemeyer's approach is situated in a precise AI research paradigm, which considers knowledge bases as declaratively specified, task-independent representations of knowledge.

The Logic of Knowledge Bases, Hector J. Levesque and Gerhard Lakemeyer, Cambridge, Massachusetts, The MIT Press, 282 pp., $45.00, ISBN 0-262-12232-4.

Starting from such a declarativist standpoint, Levesque and Lakemeyer view a knowledge base as an epistemic agent, and they set out to specify formally what knowledge can be attributed to such a system. To talk about the epistemic state of a knowledge base, Levesque and Lakemeyer introduce an extra logical symbol, *K*, to be able to distinguish what is known from what is true. At this point, a reader might wonder why the authors can't simply stick to classical first-order logic and describe what is true about the world? The reason, argue Levesque and Lakemeyer, has to do with incomplete knowledge. If a system has complete knowledge, then of course there is no difference between what is known and what is true: The two sets coincide. However, when a system has incomplete knowledge, things become more complicated. For example, a system might know that Mary has a teacher without knowing the identity of such an individual. Explicitly distinguishing what is true from what is known makes it possible for a system to deal correctly with these scenarios.

The introduction of the epistemic operator, *K*, requires the adoption of a possible world semantics for the interaction language (the language used by a reasoning engine to interact with a knowledge base at the knowledge level). In short, the abstract state of the knowledge of an agent (that is, its epistemic state) can be characterized as the collection of all possible worlds that are consistent with the knowledge held by the agent. If the knowledge of the agent is complete, then the epistemic state contains only one world. A nice feature of Levesque and Lakemeyer's treatment of epistemic logic is that in contrast to many other treatments of modalities, the discussion is reasonably easy to follow for people who are not experts in the field. This is the result of two main features of this analysis: First, the authors introduce some simplifying assumptions, such as the use of standard names to identify individuals in the universe of discourse, that do not affect the substance and the general applicability of the proposed models. Second, although the proposed language extends first-order logic with an epistemic operator, Levesque and Lakemeyer succeed in reconciling their analysis within a standard first-order–logic framework. Thus, the reader is not forced into learning a new syntax, and the underlying model theory is a "reasonably conservative" extension of standard model theory for first-order logic. More importantly, the previous statement can be given a strong interpretation because the representation theorem, discussed in chapter 7, shows that for finite knowledge bases both Tell and Ask operations (see later for more details on the Tell and Ask protocol) can always be realized using objective sentences, that is, sentences expressed in standard first-order logic. The ability to reduce formal treatments of modalities to standard first-order logic is an important result, given that standard first-order logic is far better understood

and computationally tractable than modal logics. Having said so, one slightly confusing aspect of Levesque and Lakemeyer's analysis is that although Levesque and Lakemeyer talk about modeling knowledge, they are actually modeling beliefs. For those used to "classical" theories of knowledge, this is slightly confusing because in Levesque and Lakemeyer's treatment, an agent might know something that is not true, but in many other treatments, everything known must be true (that is, the real world is always one of the possible worlds available to an epistemic agent). This is a bit disconcerting at first, but eventually the reader gets used to it; that is, he/she eventually rests assured that it is Levesque and Lakemeyer's epistemic agents who might be affected by solipsism rather than he or she.

The problem of how to handle formally and effectively incomplete knowledge is one of the two main goals of the book. The other has to do with precisely characterizing the behavior of a knowledge base at the knowledge level. The perspective of a knowledge-level view of intelligent systems was first proposed by Allen Newell in 1981 and has since informed much work in knowledge representation. In contrast with Newell's goal-oriented view of a knowledge-level system as an epistemic agent, as already mentioned, Levesque and Lakemeyer are not concerned with goal-driven behavior. For them, a knowledge base is essentially a task-independent body of knowledge to be interacted with by means of two basic operations: Tell, to add new knowledge, and Ask, to find out what the system knows or what is true about the world. Chapter 5 in the book formally specifies Tell and Ask as operations that take as arguments a sentence and an epistemic state and return either an element in the set {yes, no} (Ask) or a new epistemic state (Tell).

The first half of the book covers the basics, and the second half shows applications of the framework to nonmonotonic reasoning, tractable reasoning, and reasoning about actions. I won't go into too many details here, but essentially Levesque and Lakemeyer show how their framework can be used to reconstruct approaches such as Robert Moore's epistemic logic and John McCarthy's situation calculus. Of particular interest are chapters 12 and 13, which discuss a semantic approach to logical omniscience. A standard problem in knowledge representation is the trade-off between expressivity and computational efficiency. Most solutions err either on the side of expressivity (that is, you might wait a long time for an answer) or efficiency (that is, you are guaranteed an answer, but there is a lot that you are prevented from representing). Levesque and Lakemeyer show that it is possible to distinguish between explicit and implicit beliefs by providing a four-valued semantics (in addition to standard true and false truth assignments, sentences can now be either true and false or neither true nor false). The distinction between implicit and explicit beliefs enables very efficient decision procedures for explicit beliefs in the context of a very expressible language.

Thus, what is the general assessment of this book, and what audience is the book relevant to? The answer to the first question is quite easy. This is clearly a very good book, which provides a powerful, formal, and detailed (but reasonably easy-to-follow) logical treatment of some of the thorniest issues in knowledge representation: incomplete knowledge, nonmonotonic reasoning, reasoning about actions, and logical omniscience. The answer to the second question is a bit more complicated. The book is definitely going to be required material for anybody interested in formal knowledge representation or in formal theories of knowledge. However, what about the wider world of knowledge-based systems? After all, one would expect a book with the words *knowledge base* in the title to be of interest to the wider community of researchers and practitioners in the area of knowledge-based systems. In addition, it would be nice if such interest was not going to be driven purely by intellectual curiosity but also by the possibility of applying these results to the engineering of real systems in real contexts. Unfortunately, no attempt is made in the book to link the analysis either to concrete applications or to research in task-performing knowledge-based systems, for example, the work by Bill Clancey, Mark Stefik, and Balaknishman Chandrasekaran in the United States and by Bob Wielinga and others in Europe (Bylander and Chandrasekaran 1980; Clancey 1985; Fensel et al. 1999; Stefik 1995; Schreiber et al. 2000). Au contraire, the reader gets the feeling that the authors are actually not that interested in applications of knowledge representation technology: The examples in the book tend to be of the "Tweety is a bird" variety, and none of the 143 references seems to be related to some application.

In conclusion, this is an excellent book, which is very much grounded in the AI tradition of symbolic representation of knowledge. Anybody interested in formal representations of knowledge and epistemic agents should definitely read this text. However, those readers who are primarily interested in knowledge-based systems viewed as task-performing agents should definitely note that the word *systems* does not follow the words *knowledge base* in the title of the book. This omission is indeed a significant one!

References

Bylander, T., and Chandrasekaran, B. 1988. Generic Tasks in Knowledge-Based Reasoning: The Right Level of Abstraction for Knowledge Acquisition. In *Knowledge Acquisition for Knowledge-Based Systems, Volume 1,* eds. B. Gaines and J. Boose, 65–77. San Diego, Calif.: Academic.

Clancey W. J. 1985. Heuristic Classification. *Artificial Intelligence* 27(1): 289–350.

Fensel, D.; Benjamins, V. R.; Motta, E.; and Wielinga, B. 1999. UPML: A Framework for Knowledge System Reuse. Paper presented at the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99), 31 July–5 August, Stockholm, Sweden.

Schreiber, G.; Akkermans, J. M.; Anjewierden, A. A.; de Hoog, R.; Shadbolt, N. R.; Van de Velde, W.; and Wielinga, B. J. 2000. *Knowledge Engineering and Management: The CommonKADS Methodology.* Cambridge, Mass.: MIT Press.

Stefik M. 1995. *Introduction to Knowledge Systems*. San Francisco, Calif.: Morgan Kaufmann.

Enrico Motta is the director of the Knowledge Media Institute (KMi) of the Open University in the United Kingdom. His main interest is in knowledge technologies, and his current research focuses on the specification of reusable knowledge-based components.

Heterogeneous Agent Systems A Review

P. Ravi Prakash

The notion of software agents has been around for more than a decade. Since its beginning, the definition of agent, like the definition of intelligence, has been quite controversial and often provoked hot discussions. Questions such as the following normally come up in such arguments: What is an agent? Should a piece of software be categorized as an agent by looking at its behavioral characteristics or by the methodology using which it was produced? Is a printer daemon an agent? If a piece of software is not an agent, is there a way to make it an agent? Many attempts have been made to define the notion of agent or agency, ranging from quite generic definitions to restrictive definitions.

This book adopts a generic definition of an *agent:* a piece of code that does a small and well-defined job by offering services. Some of its characteristics are declarative specification, autonomous behavior, and interactiveness. This book primarily concentrates on abstracting the data sources and related software to make them agents, hence this less restrictive definition. Although this book might not put an end to the debates mentioned earlier, it tries to answer the most practical question of how to convert a normal program into an agent.

Until now, most of the books in this field have discussed the issues in an informal manner or only theoretically without talking about concrete implementations, leaving the reader wondering, "It is all fine, but how do I implement an agent?!" This book fills the gap to some extent. It extensively talks about the implementation of agents, using the IMPACT agent development environment. Also, most of the books in this field are a collection of articles written by different authors, often lacking coherence and consistency. Although written by seven authors, the treatment in this book is quite coherent and consistent.

The book starts off with the following questions: What is an agent? If a piece of software is not an agent, how do you make it an agent? As a response, the authors give 10 desiderata for agents and agent platforms. These desiderata roughly cover issues such as the accessing of heterogeneous data sources, a declarative framework for specifying actions, types of reasoning, security, efficiency, reliability, and validation. Of these issues, validation of an infrastructure by deploying a set of applications seems out of place. The main contributions of this book are an approach to making a normal program an agent and providing a practically implementable formal theory of agent construction and agent interactions.

Heterogeneous Agent Systems, V. S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatmaözcan, and Robert Ross, Cambridge, Massachusetts, MIT Press, 580 pp., $60.00, ISBN 0-262-19436-8.

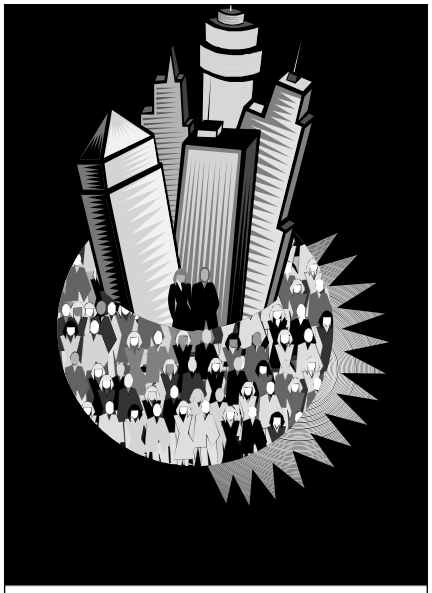
To discuss and illustrate the concepts discussed throughout the book, three motivating examples are considered. This approach is one of the good features of this book because using these examples throughout gives the book a sense of continuity. The motivating examples are a department store application, a flight terrain application, and a supply-chain management application. The department store application is supposed to proactively provide information and make multimedia presentations of the products depending on the profile of the customer. The flight terrain application is primarily used to maintain the path of a flight, taking into consideration the current position and three-dimensional terrain information provided by satellites and terrain databases, respectively. The supply-chain management application takes care of the inventory levels and the ordering logistics for production companies.

All the applications, especially the last two, have characteristics that make them suitable for using agents. First, they consist of multiple, largely independent, and well-defined tasks to be done. Second, there are independent data sources, and these sources are independently updated. Third, the actions executed can vary depending on the circumstances. Fourth, the entities in the applications need to reason with uncertain data and the beliefs they hold about other entities in the domain.

This book can roughly be divided into three parts: (1) basic concepts, (2) implementation, and (3) advanced concepts.

The first part discusses basic concepts such as the IMPACT architecture, service description language, the converting of legacy data and software into agents, and the development of agents. In chapter 2, the IMPACT architecture is discussed. One distinguishing feature of IMPACT is that it supports fuzzy matchmaking using concept hierarchies. The service description language is discussed in chapter 3. Chapter 4 contains the core concepts of the book—converting, by use of code calls, legacy data and software application program interfaces (APIs) into services. After converting the APIs and defining the constraints, the agent code looks similar to a PROLOG program. Chapter 6 discusses the components of agent programs—action base and action constraints—which is followed by a discussion of the syntax and semantics of the agent programs. Part 1 is not very formal and is highly readable.

The second part discusses implementation issues. The IMPACT server implementation and protocol details are given in chapter 5. Chapter 12 discusses the implementations for issues such as the compiling of the agent programs and safety checks.



Simulating Organizations

Computational Models of Institutions and Groups

Edited by Michael J. Prietula, Kathleen M. Carley, and Les Gasser

6 x 9, 350 pp., $45.00, ISBN 0-262-66108-X
(Prices higher outside the U.S. and subject to change without notice.)

To order, call 800-356-0343 (US and Canada) or (617) 625-8569.
Distributed by The MIT Press, 5 Cambridge Center, Cambridge, MA 02142

The third part (chapters 7 to 11), consuming roughly half the book, is devoted to advanced concepts such as belief, temporal and uncertainty reasoning, and security. This part is very formal; it is meant for researchers interested in the formal treatment of the theory. Others are advised not to venture into reading these chapters, at least during the first read, lest they might think that programming agents is an esoteric discipline that is beyond the reach of the everyday programmer!

Chapter 13 discusses a logistics application developed by the authors to illustrate the concepts discussed in the book. This chapter is quite weak; it does not cover most of the concepts discussed in the book, such as belief reasoning and temporal reasoning. Considering the supposedly significant features of agents introduced early on in the book, a more complex application that demonstrates the usefulness of the features should have been chosen.

One noticeable exclusion is agent communication mechanisms. There is no mention of communicative acts; performatives; or FIPA (Foundation for Intelligent Physical Agents), the upcoming agents standard.

There are significant advantages to applying this paradigm to make normal programs into agents, especially legacy code. The first and foremost is the declarativeness that is achieved. It allows easier program comprehension; modification; and correctness checking, manual as well as automatic. It also allows you to plug high-level reasoning capabilities, such as constraint, belief, and temporal and uncertainty reasoning. into the program (agent). However, it is obvious that not all programs require such complex and sophisticated mechanisms: For example, the department store example does not need it, but applying this paradigm to the flight terrain application might be quite worthwhile. If the flight terrain application was programmed in a traditional way, it would not have the advantages of declarative programming mentioned earlier. Also, the advantages of an agent architecture such as a matchmaking service will be missed, although it is debatable whether fuzzy matching is appropriate for all applications. Thus, if a program is complex enough, then using this paradigm will be more advantageous than the traditional way of programming.

Each chapter starts with an overview and ends with a section on related work containing brief discussions of other approaches, along with a comparison with their approach. For all the algorithms discussed in this book, the complexity analysis, as well as experimental results, are given. There is an appendix giving the codes for the motivating examples. There is an impressive list of references at the end of the book. One thing lacking in this book is exercises. Including them for at least some chapters would have made this book more useful for academic purposes. The editing should have been tighter: A few grammatical errors have slipped through. Most of the pages look a bit cluttered because of mathematics symbols. Indentation for the items in the enumerated and bulleted lists and a bold font for theorem, definition, and example headings would have improved the readability.

This book might not be suitable as a primary textbook for a course on agents because of its specificity and lack of exercises. However, it will be useful as a supplementary textbook or a reference book. It will be of use to the agent programmers (practitioners) who want to get a concrete idea about the implementation issues. This book also might interest those who want to acquire an understanding of the practical implementations of the advanced concepts such as reasoning with beliefs, uncertainty, and security with respect to agents. The bulk of the book, I feel, is for the researchers with a bent for formal treatment.

Acknowledgment

I would like to thank my colleague, M. Sasikumar, for his valuable comments on this review.

P. Ravi Prakash is a senior staff scientist at the National Centre for Software Technology, India. He received his B.Tech in computer science and engineering from JNTU, Kakinada, India. His research interests include intelligent agents, multiagent systems, planning and scheduling, soft computing, and parallel and distributed computing. His e-mail address is ravi@ncst.ernet.in.

Dynamic Logic A Review

Andrés Silva

The real world is dynamic, and any intelligent perception of the world should include the concept of time. Remember that time and space are a priori conditions of human perception in Kant's philosophy. On the one hand, time is inherent to action and change; on the other, action and change are possible because of the passage of time. According to McDermott, "Dealing with time correctly would change everything in an AI program" (McDermott 1982, p. 101).

It should not be surprising then that temporal reasoning has always been a very important topic in many fields of AI, particularly areas dealing with change, causality, and action (planning, diagnosis, natural language understanding, and so on). AI developments based on temporal reasoning lead to general theories about time and action, such as McDermott's (1982) temporal logic, Vilain's (1982) theory of time, and Allen's (1984) theory of action and time. Work on the application of these results has taken place in fields such as planning and medical knowledge-based systems.

However, action and change are not an exclusive interest of AI. In mainstream computer science, any execution of a "traditional" computer program is considered to perform an action that leads to a change of state. From this point of view, the field of program verification, traditionally focused on the correctness of actions carried out by program executions, can potentially provide AI with many approaches suitable for dealing with action and change. Temporal logic and dynamic logic are two of the approaches that have been used in the fields of both AI and program verification, temporal logic being the most popular. Both temporal and dynamic logic provide alternative applications of modal logic to program specification and verification. The main difference between the two is that temporal logic is endogenous, and dynamic logic is exogenous. A logic is *exogenous* if programs (actions) are explicit in the language. Temporal logic is *endogenous*, so its programs (actions) are never explicit in the language. Dynamic logic subsumes temporal logic.

Harel, D.; Kozen, D.; and Tiuryn, J. *Dynamic Logic*. Cambridge, Massachusetts, The MIT Press, 2000, 459 pp., $50.00, ISBN 0-262-08289-6.

Some cross-fertilization has already take place between AI, temporal logic, and dynamic logic. I focus on dynamic logic, which is the topic that is covered at length in the book under review. *Dynamic logic* is an approach to program verification with strong AI potential. One of the most prominent uses of dynamic logic in AI was Moore's (1990) approach. Moore formalized some issues related to agency, with a focus on what an agent needs to know to be able to perform an action. For more information on this topic and a clear demonstration of the usefulness of dynamic logic for agent reasoning and action, see the survey by Meyer (1999). Also, some research in knowledge engineering inspired by, or making use of, dynamic logic has been published van Harmelen and Balder (1992) and Fensel (1995).

Dynamic logic is an eclectic approach to program verification, as is evidenced by its history. This history starts with the pragmatics of programming, that is, the study of the actions that programs perform and the correctness of these actions. This has been a major issue in computer science since Dijkstra's (1968) attacks on the GOTO statement. Perhaps the most popular formal approach aimed at proving program correctness is Hoare's (1969), which is based on correctness assertions. In Hoare's logic, statements of the form {a}*P*{b} say that if program *P* starts in an input state satisfying *a*, then if and when *P* halts, it does so in a state satisfying *b*. Hoare provided some inference rules used to infer assertions about programs from assertions about other programs.

In 1976, Pratt (1976) made the connection between program logic and *modal logic,* an older tradition in which classical logic is extended with modalities. The two most important modalities used in modal logic are necessity and possibility, whose respective modal operators are $\Box$ and $\Diamond$. Therefore, if *f* is a formula, then so are $\Box f$ and $\Diamond f$. $\Box f$ should be read as "it is necessary that *f*," and $\Diamond f$ should be read as "it is possible that *f*." Semantically, modal formulas are interpreted according to Saul Kripke's semantics, best known as Kripke frames. Basically, an interpretation in modal logic consists of a collection of many possible worlds or states. Pratt's discovery, further developed by other authors, led to the association of programs with modal operators. As a result, program logic can now make use of the well-developed corpus of modal logic.

Briefly, the dynamic logic approach to program logic is as follows: The association of a modal operator, [] and <>, with a program *P*, gives birth to the operators [*P*] and <*P*>. The exogenous characteristic of dynamic logic is now clear. If *f* is a formula (propositional or first order), then [*P*]*f* should be read as "necessarily, halting executions of *P* result in a state satisfying *f*." However, <*P*>*f* should be read as "possibly, halting executions of *P* result in a state satisfying *f*." Therefore, Hoare's logic statements such as {*a*}*P*{*b*}, in dynamic logic are expressed as $a \rightarrow [P]b$. Actually, dynamic logic subsumes Hoare logic and temporal logic as well. The semantics of dynamic logic are based on Kripke frames, demonstrat-

ing that dynamic logic is built on solid modal logic foundations.

Here we have a book that provides a deep insight into the topic of dynamic logic. However, readers of this magazine should be warned: This book does not provide tips on how to apply the concepts of dynamic logic to AI because the main focus of the authors is the use of dynamic logic as a formal system for reasoning about programs.

This 460-page book is divided into three parts: (1) fundamental concepts, (2) propositional dynamic logic, and (3) first-order dynamic logic. The first part provides readers with the necessary background to understand dynamic logic and makes the book self-contained. Despite the introductory aim of this part, its contents are rather deep, amounting to one-third of the book. This first part covers mathematical preliminaries, computability, complexity, logic, and reasoning about programs. Also, the authors provide an introduction to other topics related to dynamic logic, such as temporal logic, process logic, and Kleene algebra (but, strangely enough, these topics are covered in the last chapter of the book). The second part introduces propositional dynamic logic, covering syntax, semantics, properties, completeness, complexity, and so on. The third part, on first-order dynamic logic, is the most involved part of the book and introduces syntax and semantics, uninterpreted and interpreted levels, complexity, axiomatization, expressive power of languages, and so on.

This book is a comprehensive source of information on dynamic logic. It is aimed at researchers, teachers, and students of the subject. The book can be used in a dynamic logic course because all chapters come with exercises that teachers will find useful. If you are interested in program logics and program verification using dynamic logic, this is your book. Do not expect to find any information on the application of dynamic logic to AI or knowledge representation. However, AI researchers who want to deepen their understanding of the capabilities and limits of dynamic logic will find useful information in the book.

References

Allen, J. 1984. Toward a General Theory of Action and Time. *Artificial Intelligence* 23:123–154.

Dijkstra, E. W. 1968. Go To Statement Considered Harmful. *Communications of the ACM* 11(3): 147–148.

Fensel, D. 1995. *The Knowledge-Based Acquisition and Representation Language* KARL. New York: Kluwer Academic.

Hoare, C. A. R. 1969. An Axiomatic Basis for Computer Programming. *Communications of the ACM* 12(10): 576–580, 583.

McDermott, D. 1982. A Temporal Logic for Reasoning about Processes and Plans. *Cognitive Science* 6(2): 101–155.

Meyer, J.-J.Ch. 1999. Dynamic Logic Reasoning about Actions and Agents. Paper presented at the Workshop on Logic-Based Artificial Intelligence, 14–16 June, Washington, D.C.

Moore, R. C. 1990. A Formal Theory of Knowledge and Action. In *Readings in Planning*, eds. J. F. Allen, J. Hendler, and A. Tate, 480–519. San Francisco, Calif.: Morgan Kaufmann.

Pratt, V. R. 1976. Semantical Considerations on Floyd-HCare Logic. In Proceedings of the Seventeenth Symposium on the Foundations of Computer Science, 109–121. Washington, D.C.: IEEE Computer Society.

van Harmelen, F., and Balder, J. R. ML^2: A Formal Language for KADS Models of Expertise. *Knowledge Acquisition* 4(1): 127–167.

Vilain, M. 1982. A System for Reasoning about Time. In Proceedings of the Second National Conference on Artificial Intelligence, 197–201. Menlo Park, Calif.: American Association for Artificial Intelligence.

Andrés Silva is assistant professor at Universidad Politécnica de Madrid. He started his research career at the Laboratory of Artifical Intelligence at Universidad Politécnica de Madrid and with the Group of Artificial Intelligence and Learning (GRAIL) at Universidad Carlos III. In 1998 to 1999, he worked at the Joint Research Centre of the European Commission in Ispra (Italy). He holds a Ph.D. in computer science from Universidad Politécnica de Madrid. His e-mail address is asilva@fi.upm.es.

Automated Theorem Proving: Theory and Practice A Review

Geoff Sutcliffe

Automated Theorem Proving (ATP) deals with the development of computer programs that show that some statement (the conjecture) is a logical consequence of a set of statements (the axioms and hypotheses). ATP systems are used in a wide variety of domains: A mathematician might use the axioms of group theory to prove the conjecture that groups of order two are commutative; a management consultant might formulate axioms that describe how organizations grow and interact and, from these axioms, prove that organizational death rates decrease with age; or a frustrated teenager might formulate the jumbled faces of a Rubik's cube as a conjecture and prove, from axioms that describe legal changes to the cube's configuration, that the cube can be rearranged to the solution state. All these tasks can be performed by an ATP system, given an appropriate formulation of the problem as axioms, hypotheses, and a conjecture. Most commonly, ATP systems are embedded as components of larger, more complex software systems, and in this context, the ATP systems are required to autonomously solve subproblems that are generated by the overall system. To build a useful ATP system, several issues have to carefully be considered, independently and in relation to each other, and addressed in a synergetic manner. These issues include the choice of logic that will be used to represent the problems, the calculus that will be used for deduction, the programming language that will be used to write the ATP system, the data structures that will be used to hold the statements of the problem and the statements deduced by the system, the overall scheme for controlling the deduction steps of the system, and the heuristics that will control the fine-grained aspects of the deduction steps (the heuristics are most likely to determine the success or failure of an ATP system because they most directly control the system's search for a solution). Current state-of-the-art ATP systems, such as E, VAMPIRE, E-SETHEO, and WALDMEISTER, have been developed with the benefit of years of experimentation and effort (information about these systems can easily be found on the World Wide Web). For a newcomer to the field, it is a daunting, if not seemingly impossible, task to start on the road to building a decent ATP system. It is almost imperative that a budding ATP system developer should start with the benefit of previous experience. Monty Newborn's book, *Automated Theorem Proving: Theory and Practice,* can contribute to this learning process.

Automated Theorem Proving: Theory and Practice, Monty Newborn, Berlin, Springer-Verlag, 231 pp., $54.95. ISBN 0-387-95075-3.

Monty Newborn is a professor of computer science at McGill University, Canada. Newborn is probably better known for his involvement with computer chess than with ATP. In particular, he is the long-standing chairperson of the ACM Computer Chess Committee, which organized the famous 1997 match in which IBM's DEEP BLUE program defeated the human world champion, Gary Kasparov. It is Newborn's background in the search issues of computer chess that led to his more recent interest in ATP. His book provides an introduction to some of the basic logic, calculi, heuristics, and practicalities of first-order ATP. Rather than working rigorously through the theory of mathematical logic, the book focuses on just the necessary foundations for understanding how first-order ATP systems operate and links these foundations to the implementation of two ATP systems, HERBY and THEO. The book comes with a CD containing the source code for HERBY and THEO and several suites of ATP problems for HERBY and THEO to attempt.[1] Thus, the reader is able to experiment as both a user and a developer of ATP systems. The provision of the ATP systems and problems sets this book apart from most other introductory books on ATP that provide a more in-depth treatment of the theory but fail to get readers over the initial hurdles of using an ATP system (a notable exception is Wos et al.'s book, *Automated Reasoning: Introduction and Applications* [McGraw-Hill, 1992], which comes with the well-known ATP system OTTER). Newborn's book is suitable as an introduction to ATP for undergraduate university students and independent, interested readers.

After an introductory chapter explaining the structure of the book and software installation, chapters 2 to 4 introduce first-order logic (in the syntax used by HERBY and THEO) and the basic mechanics and semantics of resolution-based ATP. Chapters 5 and 6 then provide the underlying theory and describe the calculi for the two ATP systems, chapter 5 corresponding to HERBY and chapter 6 corresponding to THEO. The architecture, use, and implementation of HERBY are described in chapters 7, 8, and 11, respectively, and the same information is provided for THEO in chapters 9, 10, and 12. The last chapter steps aside to briefly discuss the CADE ATP System Competition (CADE [the Conference on Automated Deduction] is the major forum for the presentation of new research in all aspects of automated deduction). Variants of HERBY and THEO participated in the competitions in 1997 and 1998.

Each chapter ends with a set of exercises that are useful for testing the reader's understanding of the chapter material.

Many first-order ATP systems and calculi, including those described in this book, use the clause normal form (CNF) of first-order logic statements. A procedure for converting a problem in first-order form to CNF is described in chapter 3, along with instructions for using the conversion program supplied on the CD. The use of CNF is largely motivated by the resolution inference rule, which generalizes the intuitively understandable modus ponens rule: If *A* is true, and it is true that *A* implies *B*, then *B* is true. THEO is a resolution-based ATP system. Chapter 4 describes the binary-resolution and factoring inference rules, which, in combination, implement the full-resolution inference rule. Chapter 4 also introduces the very important subsumption rule, which is used to eliminate redundant clauses that might be inferred (although the book admits that HERBY and THEO implement only a weakened form of subsumption, called s–subsumption).

The underpinning for resolution-based ATP is Herbrand's theorem, which shows that logical consequence can be established by considering instances of a set of clauses. The semantic tree calculus implemented in HERBY is a direct application of Herbrand's theorem. Herbrand's theorem and the semantic tree calculus are presented in chapter 5. The semantic tree calculus is appropriate for an introductory book on ATP because it provides a way to see the immediate consequences of Herbrand's theorem in action. It also provides a clear framework in which to illustrate some issues of heuristic control in an ATP system, as is done in chapter 7. Unfortunately, the semantic tree calculus fails to benefit from the abstraction inherent in the resolution inference rule, thus making it hard to develop a high-performance ATP system based on this calculus. The reader should note that the examples in chapter 5 all use a finite Herbrand universe and that a semantic tree can obviously get a lot deeper when the Herbrand universe is infinite, as exemplified by the example in chapter 8.

The resolution calculus, the basis for THEO, is described in chapter 6 with some nice examples. The explanation of how a resolution proof can be extracted from a closed semantic tree provides an excellent link to the preceding chapter and Herbrand's theorem. Linear resolution, a refinement of the resolution calculus, is introduced next. Linear resolution is significant as the basis for the PROLOG programming language (where the format of the problem is restricted to Horn clauses, providing completeness for linear-input resolution) and its use in various high-performance ATP systems, for example, PTTP, PROTEIN, and METEOR. Linear resolution also parallels the tableau calculus used successfully by the E-SETHEO system. Interestingly, Newborn chooses to extend linear resolution to linear-merge resolution. Historically, resolution with merging was developed separately but around the same time (late 1960s) as linear resolution. The link between them was noticed later, and several ATP systems exploited the link until the late 1970s when interest in developing their combination seemed to disappear. The THEO system is thus uniquely interesting among the linear resolution–based systems available now. Chapter 9, as well as describing the architecture of THEO, describes some strategies that can be used by a broad range of resolution-based systems to improve performance. Some of these strategies come at the cost of completeness; that is, if such a strategy is used, then some theorems can no longer be proved. However, it is well known that almost all contemporary high-performance ATP systems use incomplete strategies (sometimes even unintentionally).

The chapters describing the implementations of HERBY and THEO will be too technically messy for a reader who is primarily interested in the deductive issues and will also be inadequate for a programmer who wants to understand the internal workings of the systems (although the programmer does have the option of examining the source code provided on the CD). These chapters, however, will give all readers a feel for the complexity of source code and data structures that are required to implement an ATP system in an imperative programming language such as C.

Overall, Newborn's book meets the needs of its intended audience as a straightforward and practical introduction to ATP. If it whets your appetite, you can take advantage of the bibliography of appropriate further material. If not, you'll walk away with at least a basic understanding of ATP.

Note

1. The first-order problems in the WFF directory on the CD use the keyword theorem to mark the conjecture to be proved, but the compile program, for converting problems to clause normal form, expects to find the keyword conclusion. To use the compile program, it is necessary to edit either the note.WFF files or the compile source code file lexwff.c in the COMPSC directory.

Geoff Sutcliffe is a faculty member in the Department of Computer Science at the University of Miami. His research focuses on the evaluation and appropriate application of automated theorem-proving (ATP) systems, including the development of parallel and distributed ATP systems, and easy-to-use ATP system interfaces. His e-mail address is geoff@cs.miami.edu.