COMPUTATIONAL PHILOSOPHY OF SCIENCE

Paul Thagard

Computational Philosophy of Science

Paul Thagard

A Bradford Book
The MIT Press
Cambridge, Massachusetts
London, England

First MIT Press paperback edition, 1993

This book was set in Palatino by Asco Trade Typesetting Ltd., Hong Kong, and printed and bound by Halliday Lithograph in the United States of America.

Library of Congress Cataloging-in-Publication Data

Thagard, Paul
Computational philosophy of science/Paul Thagard.
p. cm.
"A Bradford book."
Bibliography: p.
Includes index.

4 (PB)

1. Science—Philosophy. 2. Psychology. 3. Artificial intelligence. I. Title.
Q175.T479 1988 501—dc19 87-21892

Contents

Preface

To some ears, "computational philosophy of science" will sound like the most self-contradictory enterprise in philosophy since business ethics. On the contrary, central philosophical issues concerning the structure and growth of scientific knowledge can be greatly illuminated by drawing on ideas and techniques from the field of artificial intelligence. This book uses PI, a computer program for problem solving and induction, to illustrate the relevance of computational ideas to questions concerning the discovery, evaluation, and application of scientific theories.

The first part of the book is concerned with computational models of scientific thinking, and should appeal to those interested in artificial intelligence and cognitive psychology as well as to philosophers. Later chapters turn to more traditional philosophical issues, concerning the relation between how reasoning is done and how it ought to be done, truth, the justification of scientific methods, and the difference between science and pseudoscience. Some of the general conclusions about the nature of scientific method are applied to the particular fields of psychology and artificial intelligence. The book concludes with a highly speculative chapter concerning what computational models might add to our understanding of two key aspects of the process of inquiry: the interrelations of theory and experiment, and the importance of group rationality in science.

I have tried to make this book accessible to an interdisciplinary readership by clarifying philosophical and computational terms as they arise. To provide background for readers of different fields without interrupting the argument, appendix 1 contains four tutorials providing essential philosophical, computational, and psychological introductions. Each chapter concludes with a summary of its most important claims.

The book is offered in the hope that it will be read without arbitrary categorizations of what is philosophy, artificial intelligence, or psychology, and in the conviction that an understanding of scientific reasoning can only come through interdisciplinary cooperation.

Acknowledgments

This book has depended on the assistance of many people and institutions. It was begun when I was an Associate Professor of Philosophy at the University of Michigan-Dearborn, and I am grateful for the freedom provided there by my colleagues. Much of the work was done while I was also associated with the Cognitive Science Program at the University of Michigan, Ann Arbor, which provided support of many kinds. The book has been completed using the excellent facilities of the Cognitive Science Laboratory at Princeton University, where I am now doing interdisciplinary research as part of the Human Information Processing Group.

In the period during which many of the ideas of this book were being developed, I was fortunate to collaborate with John Holland, Keith Holyoak, and Richard Nisbett on our 1986 book: *Induction: Processes of Inference, Learning, and Discovery*. Some of the themes developed here were sketched in a preliminary way there. I am particularly indebted to Keith Holyoak, since the processing system PI, whose philosophical implications and applications are discussed at length here, is a collaborative project with him. I am grateful to Richard Nisbett for first acquainting me wth work in cognitive psychology, and to John Holland for supervising my M.S. in computer science.

Earlier drafts of this book received valuable comments from Paul Churchland, Robert Cummins, Daniel Hausman, Stephen Hanson, Gilbert Harman, Ziva Kunda, Richard Nisbett, and Ian Pratt. I am especially grateful to Lindley Darden for detailed comments on two separate drafts.

Over the years that this book developed, I have been grateful for the support of the National Science Foundation, the Sloan Foundation, the Systems Development Foundation, the Basic Research Office of the Army Research Institute for Behavioral and Social Science, and the McDonnell Foundation.

Some sections of this book derive from previously published articles. I am grateful to the respective publishers for permission to use the following material.

Section 1.4 and tutorial D draw on "Frames, Knowledge, and Infer-

Computational Philosophy of Science

Chapter 1

Computation and the Philosophy of Science

> Epistemology without contact with science becomes an empty scheme. Science without epistemology is—insofar as it is thinkable at all—primitive and muddled.
> (Albert Einstein, 1949, pp. 683ff.)

1.1. A New Approach

Philosophy of science and artificial intelligence have much to learn from each other. The central questions that can benefit from a multidisciplinary investigation include

1. What are scientific theories?
2. What is scientific explanation and problem solving?
3. How are theories discovered and evaluated?
4. How do theoretical concepts become meaningful?
5. What are the roles of theorizing and experimentation in the process of scientific inquiry?
6. How can descriptive studies of how science is done be relevant to normative judgments about how it ought to be done?

This book presents an integrated set of answers to these questions within a computational framework. Here is a preliminary sketch of what is proposed in later chapters.

1. Theories are complex data structures in computational systems; they consist of highly organized packages of rules, concepts, and problem solutions.
2. Explanation and problem solving are computational processes mediated by the rules, concepts, and problem solutions that can constitute theories.
3. The discovery and evaluation of theories are subprocesses that are triggered in the context of explanation and problem solving.
4. Theoretical concepts are meaningful because of their generation

by discovery processes and because of their connections with other concepts.
5. Theorizing and experimentation play complementary roles in scientic inquiry, with neither dominant.
6. Descriptive studies of how science is done can provide an essential contribution to the determination of how science ought to be done.

Fleshing out these vague claims will proceed in later chapters. To substantiate them, I shall describe an artificial intelligence program for problem solving and induction, showing how its operation helps to illustrate the processes by which scientific theories are constructed and used. I shall argue that richer philosophical accounts of scientific problem solving, discovery, and justification can be developed using the resources of artificial intelligence than are possible with the traditional techniques of logic and set theory. I do not pretend to have solved the numerous difficult problems concerning such topics as explanation and justification that are addressed here; but I do hope to show that a computational approach offers ideas and techniques for representing and using knowledge that surpass ones usually employed by philosophers. Before launching into computational details, I want to situate the enterprise I am calling "computational philosophy of science" in relation to more familiar fields.

1.2. Artificial Intelligence, Psychology, and Historical Philosophy of Science

Artificial intelligence (AI) is the branch of computer science concerned with getting computers to perform intelligent tasks. In its brief three decades of existence, AI has developed many computational tools for describing the representation and processing of information. Cognitive psychologists have found these tools valuable for developing theories about human thinking. Similarly, computational philosophy of science can use them for describing the structure and growth of scientific knowledge.

To a large extent, then, the concerns of AI, cognitive psychology, and computational philosophy of science overlap, although philosophy has a greater concern with normative issues than these other two fields. We must distinguish between descriptive issues, concerning how scientists do think, and normative issues, concerning how scientists ought to think. Cognitive psychology is dedicated to the empirical investigation of mental processes, and is interested in normative issues only to the extent of characterizing people's departures from assumed norms (see Nisbett and Ross, 1980, for a recent survey). Similarly, artificial intelligence understood as cognitive modeling can confine itself to the descriptive rather than the normative. AI, however, is also sometimes concerned with improving on the performance of people and therefore can be interested in what is optimal and normative.

For philosophy of science, discussion of normative questions is inescapable, although we shall see in chapter 7 that descriptive and normative issues are intimately related.

Current research in AI divides roughly into two camps, which have colorfully been characterized as "neats" and "scruffies". The distinction is based largely on attitudes toward the importance of formal logic in understanding intelligence. The neats, such as John McCarthy (1980) and Nils Nilsson (1983), view logic as central to AI, which then consists primarily of constructing formal systems in which logical deduction is the central process. In contrast, scruffy AI, represented, for example, by Marvin Minsky (1975) and Roger Schank (1982), takes a much more psychological approach to AI, claiming that AI is more likely to be successful if it eschews the rigor of formal logic and investigates instead the more varied structures and processes found in human thought. Using the computer programmers' term for a complex and unsystematically put together program, Minsky remarks that the brain is a "kluge". A third influential approach to AI, the production systems of Newell and Simon (1972), falls somewhere between the neat and scruffy camps. Psychologists range from neats who emphasize the role of logic in thinking (Braine, 1978; Rips, 1983) to scruffies who deny that logic is at all central (Johnson-Laird, 1983; Cheng et al., 1986).

Philosophy also has its neats and scruffies. No one was ever neater than the logical positivists, who used the techniques of formal logic to analyze the nature of theories and other key problems. It is therefore not surprising that formally inclined philosophers are displaying a growing interest in such AI endeavors as algorithmic analysis and logic programming (Glymour, Kelly, and Scheines, 1983). But this trend reflects the relation only of neat AI to neat philosophy of science. Since any computer implementation requires formalization, which was the hallmark of the logical positivists, one might suppose that any artificial intelligence approach to the philosophy of science would fall within the positivist camp. This conclusion, however, sorely underestimates the intellectual resources of artificial intelligence.

In the 1950s and 1960s, philosophy of science saw a rebellion against logical positivist accounts of science, led by such writers as Hanson (1958) and, especially, Kuhn (1970b). (For a sketch of developments in the philosophy of science, see tutorial A in appendix 1.) Critics argued that the positivists' emphasis on formal models had led them farther and farther away from the practice of actual science. Many philosophers of science have since adopted a methodology that avoids formalization, instead giving less precise descriptions of the methods of scientists based on historical case studies. Kuhn, for example, drew heavily on such examples as Lavoisier's theory of oxygen and Einstein's theory of relativity to back his account of the growth of science.

Historical philosophy of science has contributed to a much more rich and subtle account of the nature of science than could be developed within the framework of the logical positivists. But it has lacked one of the most appealing features of the positivist program: analytical rigor. Kuhn described scientific revolutions as the surpassing of one paradigm by another, but the central concept of a paradigm was left notoriously vague. Similarly, Laudan's (1977) influential work on science as a problem-solving activity never said much about the nature of problem solving.

These gaps can be filled in by computational philosophy of science, which this book places at the intersection of scruffy AI and historical philosophy of science. By offering detailed computational analyses of the structure and growth of knowledge, I hope to show that postpositivist philosophy of science can have some rigor in its scruffiness.

Hanson and Kuhn both made use of ideas from gestalt psychology in developing their alternatives to logical positivist accounts of science. Computational philosophy of science is even more closely tied with psychology, by virtue of the link between scruffy AI and current cognitive psychology, which increasingly employs computational models as theoretical tools. These three fields can collaborate in developing a computational account of how human scientists think. Many researchers in philosophy of science and artificial intelligence would prefer to leave psychology out of the picture, and science may indeed someday be performed by computers using processes very different from those in humans. But for now, at least, science is a human enterprise, and understanding of the development of scientific knowledge depends on an account of the thought processes of humans. Hence computational philosophy of science overlaps as much with cognitive psychology as it does with scruffy AI. Even its normative prescriptions about how science ought to be done should take human cognitive limitations as starting points, according to the view developed in chapter 7.

Computational philosophy of science and much of current cognitive psychology employ computational models, but why? In the next section I shall sketch the methodological advantages of using computer programs for understanding thinking.

1.3. Why Write Programs?

There are at least three major gains that computer programs offer to cognitive psychology and computational philosophy of science: (1) computer science provides a systematic vocabulary for describing structures and mechanisms; (2) the implementation of ideas in a running program is a test of internal coherence; and (3) running the program can provide tests

is a lot more to computation than deduction, making possible the investigation of less constrained processes, such as those underlying scientific discovery.

Because mental processes are postulated to be computational, the computer is potentially an even more powerful tool for psychology than it is for such fields as economics and meteorology that use *weak* simulations in contrast to psychology's *strong* simulations. In a weak simulation, the computer functions as a calculating device drawing out the consequences of mathematical equations that describe the process simulated. A computer can valuably simulate a business cycle or a hurricane, but no one contends that it *has* an economic depression or high winds. In a strong simulation, however, the simulation itself resembles the process simulated. For example, a wind tunnel used to study the aerodynamics of cars is a strong simulation, since the flow of air over the car in the tunnel is similar to the flow of air over the car on the highway. In contrast, a computer model of the car's aerodynamics would only be a weak simulation. Whereas for most fields computers will only provide weak simulations, psychology has the possibility of strong simulations, if the computational theory of mind is correct.

Of course, merely characterizing data structures and processes in computational terms does not tell us how the mind operates. But even getting the program to run provides a test of sorts. Some noncomputational psychologists tend to assume that anything can be programmed, but this is no more credible than the assumption of some computer scientists that any psychological data can be got by a clever experimenter. To run, a computer program has to have at least a coherent interrelation of structures and algorithms. In addition, the threat of combinatorial explosion puts a severe constraint on the realizability of programs: if the program requires exponentially increasing time to run, it will quickly exhaust the resources of the most powerful computers. So developing a computer simulation provides a valuable test of the internal coherence of a set of ideas.

A psychological model should be more than internally coherent: we want it to account for experimental data about how people think. But sometimes, if a model is complex, it is not easy to see what its consequences are. Cognitive models, like many models in the social sciences, often postulate many interacting processes. The computer program enables a researcher to see whether the model has all and only the consequences that it was expected to have. Comparison of these consequences against experimental observations provides the means of validating the model in much greater detail than pencil-and-paper calculations might allow. Computational philosophy of science can benefit from the same model-forming and model-testing benefits that AI provides to cognitive psychology.

Knowledge is both private and public, inhabiting the brains of particular thinkers, but also subject to intersubjective communication and assessment. Weak psychologism aims to capture both these aspects. The real test between weak psychologism and antipsychologism consists in seeing which framework can develop a comprehensive and rich account of human knowledge. This book can be viewed as a computationally oriented attempt to describe some possible results of a weak psychologistic research program. Kindred attempts include the naturalistic epistemology of Quine (1969), the genetic epistemology of Piaget (1970), the epistemics of Goldman (1978, 1986), and the evolutionary epistemology of Campbell (1974). The last of these is criticized in chapter 6.

I share with such authors the view that philosophical method should be more akin to theory construction in science than to the sort of conceptual analysis that has been predominant in much twentieth-century philosophy. No precise analyses of individual concepts will be offered, because there are grounds for doubting whether such analyses are to be had (see sections 2.3.1 and 4.4), and because the larger enterprise of describing systematic connections among such processes as explanation and hypothesis formation is much more interesting.

1.5. Overview

Exploration of computational philosophy of science begins in the next chapter, with a discussion of the basic structures and processes relevant to an understanding of scientific knowledge. The artificial intelligence program PI provides a concrete example of how knowledge can be organized and used in problem solving. Chapter 3 then develops a computational account of the nature of scientific theories and explanations. Chapter 4 describes how abductive inference can be computationally implemented, providing an account of several kinds of scientific discovery. It also discusses how new concepts can be formed and acquire meaning. In chapter 5, I develop an account of theory evaluation as inference to the best explanation and describe its implementation in PI. Chapter 6 uses the ideas about discovery and evaluation of theories developed in earlier chapters to criticize the Darwinian model of knowledge development offered by evolutionary epistemologists. The next three chapters shift concern to normative matters. Chapter 7 develops a model for reaching normative conclusions from descriptive considerations, and the model is applied in chapter 8 to the problems of justifying inference to the best explanation and defending scientific realism. Chapter 9 discusses the normative problem of distinguishing science from pseudoscience. Finally, in chapter 10 I offer some speculative suggestions about what computational philosophy of science may be able to contribute to questions concerning the relation of

theory and experiment and the role of group rationality in science. I have added three appendices to fill in details that would have distracted from the main argument. The first consists of four tutorials providing background information concerning the philosophy of science, logic, data structures and algorithms, and schemas. The second provides a summary of the structure of the computer program PI discussed in chapters 2–5, and the third presents a sample run of PI.

1.6. Summary

Computational philosophy of science is an attempt to understand the structure and growth of scientific knowledge in terms of the development of computational and psychological structures. It aims to offer new accounts of the nature of theories and explanations, and of the processes underlying their development. Although allied with investigations in artificial intelligence and cognitive psychology, it differs in having an essential normative component.

Chapter 2

The Structure of Scientific Knowledge

This chapter begins a computational analysis of scientific knowledge by discussing how such knowledge can be represented and used in computer programs. Artificial intelligence provides a new set of techniques for representing different parts of the scientific corpus, including laws, theories, and concepts. To present concretely the need for complex representations of these essential ingredients of scientific knowledge, I shall describe PI, a running program for problem solving and induction.

2.1. Structure and Process

In the last chapter, we saw that that there are good reasons for the dramatic influence of computational ideas in psychology. From artificial intelligence, psychology has gained a new stock of ideas concerning representations and processes, as well as a new methodology of testing ideas using computer simulation. This and later chapters will exhibit similar reasons for a computational approach to epistemology and the philosophy of science.

The case for the epistemological relevance of computation rests on a simple but extremely important point: Structure cannot be separated from process. We cannot discuss the structure of knowledge without paying attention to the processes that are required to use it. This point is familiar to most practitioners of artificial intelligence, but is new to philosophers, who have in this century had a relatively simple view of the structure of knowledge. Since the pioneering work of Frege and Russell, formal logic has been the canonical way of describing the structure of knowledge. In first-order predicate calculus, a simple atomic sentence such as "Fred is angry" is represented by a predicate and an argument such as **A(f)**. AI use of the predicate calculus is less cryptic, so that the same sentence is represented by **angry (Fred)**. More complex sentences are built up using connectives like **and**, **or**, and **if-then**, and by quantifiers such as **some** and **all**. For example, the sentence, "All criminals are angry." can be represented as **(for all x)(if criminal(x) then angry(x))**. Predicate calculus has many strengths as a starting point for representing knowledge, but we shall see below that it does not provide sufficient structure for all processing pur-

poses. (Readers in need of a brief introduction to predicate calculus should consult tutorial B.)

In twentieth-century philosophy, the most studied technique for using knowledge is deduction in logical systems, in which rules of inference can be precisely defined. For example, modus ponens is the rule of inference that licenses the inference from **if p then q** and **p** to **q**. But there must be more to a processing system than deduction. If a system is large, assembling the relevant information at a particular time can be highly problematic. In epistemological systems based on logic, a corpus of knowledge is generally taken to consist of all the deductive consequences of a set of statements, even though the set of consequences is infinite. For more realistic systems, it becomes crucial to ask the question, What shall we infer when? So even in a system designed to do deduction, we need processes that take into account what information is available and what rules of inference are appropriate.

In any system designed for learning as well as performance, for acquisition of knowledge as well as its use, nondeductive processes are required. Scientific discovery is multifaceted, requiring diverse processes for generating concepts, forming general laws, and creating hypotheses. Such processes depend, we shall see, on complex representations of concepts and laws.

My concern in this book is with scientific knowledge. Hence the next section will discuss what kinds of structures and processes are most important for characterizing scientific knowledge. Then I shall describe a comprehensive processing system to illustrate in much greater detail how structure and process are interrelated.

2.2. Scientific Knowledge

To represent scientific knowledge, we need to find a formal expression for at least three kinds of information: observations, laws, and theories. Philosophers of science have differed on the relative importance of these aspects in the development of scientific knowledge. On one simple account of how science develops, scientists start by making experimental observations, and then use these to generate laws and theories. On an equally simple and misleading account, scientists start with laws and theories and make predictions that they then check against observations. In most scientific practice, there is rather an interplay of hypotheses and observations, with new observations leading to new laws and theories and vice versa (see chapter 10). To describe the process of science computationally, we need to be able to formalize observations, laws, and theories in structures that can be part of computer programs. In addition, I shall argue that it is also necessary to use a rich representation of scientific concepts. Formalization is necessary

but not sufficient for representation, since we could formalize a body of scientific knowledge in predicate calculus or set theory without it being represented in a form that is computationally usable. Formalization and representation must go hand in hand, putting the knowledge into a form that can be processed.

A particular observation that a specimen, call it **specimen27**, is blue can easily be represented in predicate calculus as **blue(specimen27)**. More complex observations concern relations between objects, which predicate calculus can represent by allowing more than one argument. For example, that one specimen is observed to be to the left of another can be represented by **left-of(specimen27, specimen42)**. Relations become even more important if temporal information is also to be added: we can formalize the information that specimen 27 was blue at time t by writing **blue(specimen27, t)**. So predicate calculus appears to be an excellent way of representing observations, particularly about relations. Any representation of scientific knowledge will have to be able to distinguish between x being to the left of y, and y being to the left of x, which predicate calculus does very handily by contrasting **left-of(x, y)** with **left-of(y, x)**.

Science obviously does more than just collect observations. A central aim is to organize observations by means of laws. In physics, these can be highly general, as in the law that any two objects have a gravitational force between them. In the social sciences and in much of twentieth-century physics it is common to speak of *effects* rather than laws, indicating a statistical relation rather than full generality. General laws can naturally be represented by quantified expressions in predicate calculus. For example, the simple law that copper conducts electricity becomes **(for all x)(if copper(x) then conducts-electricity(x))**. It might seem, then, that predicate calculus is all we need for laws too.

But that conclusion neglects the important point about process made above. If all we wanted to do with laws was to use them in logical deductions, then predicate calculus might be fine. But laws have many important additional roles to play. They are discovered using observations, serve to predict new observations, help enormously in problem solving and explanation, and are explained by theories. To function in all these processes, it is useful to give laws a more complex representation such as that used for rules in the system PI discussed below.

From a logical point of view, theories look just like general laws. Newton's theory of gravitation, for example, says that between any two bodies there is a force. This could be represented by the rule, If x is a body and y is a body, then there is a force z between x and y. But theories differ from laws in their origins and explanatory roles. Whereas laws are generalized from observations, theories are evaluated by seeing how well they explain laws (see chapter 5). Moreover, since theories go beyond what is observed

tutorial B). Predicates are associated with sets of objects in a domain, interpreted as those objects of which the predicate is true. However, we shall see in chapter 4 that model-theoretic semantics is inadequate as a theory of meaning of scientific predicates.

Scientists are more aware of the value of concepts than logicians. Without appropriate concepts, formation of useful laws and theories is impossible. For example, Einstein and Infeld (1938, p. 133) include the following exclamation in their discussion of the physical concept of a field: "How difficult it would be to find these facts without the concept of a field! The expression for a force acting between a wire through which a current flows and a magnetic pole is very complicated. In the case of two solenoids [coils of wire] we should have to investigate the force with which two currents act upon each other. But if we do this, with the help of the field, we immediately notice the character of all those actions at the moment when the similarity between the field of a solenoid and that of a bar magnet is seen." To understand the importance of concepts in this kind of discovery and in scientific thinking in general, a richer representation of concepts than mere predicates will turn out to be necessary. In particular, chapter 4 will describe how the development of theoretical concepts requires that concepts have a rich internal structure.

2.3. Structure and Process in PI

To be more concrete about the importance of rich representations, I shall now outline an artificial intelligence program called *PI*, which stands for "processes of induction" and is pronounced "pie". PI implements in the programming language LISP a general model of problem solving and inductive inference developed in collaboration with cognitive psychologist Keith Holyoak. The intention in describing PI is not to propose is as a canonical language for doing science; its limitations will be described. Nor is PI claimed to constitute in itself a solution to the host of difficult problems in the philosophy of science concerning explanation, justification, and so on. Rather, I present it as an illustration of how representation and process interact and of how an integrated general account of scientific discovery and justification can begin to be developed within a computational framework. Supplemental descriptions of the operation of PI can be found elsewhere (Holland et al., 1986; Thagard and Holyoak, 1985), and appendices 2 and 3 contain much more detailed information about PI's implementation in LISP.

2.3.1. Knowledge Representation

PI represents particular results of observation and inference by *messages*, which are similar to sentences in predicate calculus and to what are called

"facts" in production systems. A message is a list that includes the following information: predicate, argument, truth-value, confidence, and message-name. For example, the observation that the planet Mars is red is represented by the list **(red (Mars) true 1)**. A similar structure can also represent simple hypotheses. The information that Mars is hypothesized to be devoid of life could be represented by the list **(has-life (Mars) projected-to-be-false .7 hypothesis-26)**. In addition to the obvious truth values **true** and **false**, PI also allows more tentative projected values. The number .7 indicates how confident the system is in the message, while the message name can be used to store additional information, for example, about the evidence for the hypothesis. Thus PI's messages, although starting with a structure derived from predicate calculus, add more information that will play an important role in problem solving and inductive inference.

Laws are represented by rules, which are if-then statements such as **If x is copper then x conducts electricity**. Even more than for messages, it turns out to be useful to add much more structure than a statement in predicate calculus would have. For a start, we want to give rules names to keep track of their successes and failures. Past successes and failures are summed up in a quantity called *strength*, which in PI is a number between 0 and 1. As we shall see below, it is important for problem solving that rules be attached to concepts, so the full profile of the above rule about copper might be

Name:	Rule-22
Data-type:	rule
Concepts-attached-to:	copper
Condition:	If x is copper
Action:	Then x conducts electricity
Strength:	.7
...	...

LISP programmers will recognize this as a property list of the atom Rule-22. Pascal programmers can think of it as a record with various fields. Basic and Fortran programmers will have to think of it as a more complex kind of array than they are used to. Logicians usually call the condition of a rule its "antecedent" and the action of a rule its "consequent". Complex conditions and actions make possible the representation of mathematical laws. Newton's law F = ma becomes, If x is force and y is mass and z is acceleration, then x = y times z.

Concepts in PI are still more complicated, in that they are represented by rich structures akin to the frames of Minsky (1975). A frame represents a typical kind of object or situation (see tutorials C and D for background). Each of PI's concepts includes information about its place in a hierarchical network of concepts: dogs, for example, are kinds of animals and have

Projection-status:	nil
Current-value:	0
Action-instances:	nil

The conditions, actions, slot, status, confidence, and strength are all set up by the programmer. The other properties of the rule, from Old-matches on down, are initially empty but get filled in by the program as it proceeds. For example, it is crucial to keep track of old matches—what messages have previously matched all the conditions and led to firing of the rule—to stop the same rule being applied over again in the same inference and preventing other rules from firing. In rule-based systems, this is called "refraction". Satisfies-goal? is used to keep track of whether firing a rule would satisfy a problem's goal, in order to ensure that such a rule will fire. Current-value gets calculated when the conditions of a rule are matched and determines whether the rule will be selected as one of the rules to be fired, taking into account such factors as the strength and degree of activation of the rule. Action-instances are the actions of the rule with variables bound when the conditions are matched against messages. Appendix 2 provides an outline of the LISP functions for firing rules in PI.

Note that rules such as Rule-3 do not constitute a strict analysis or definition of "sound". They express what is typical of sounds, not what is universally necessary and sufficient for being a sound. Dictionaries are of little help in forming such definitions, as the following typical dictionary entry shows (Guralnik, 1976, p. 1360):

> **sound** 1. *a)* vibrations in air, water, etc. that stimulate the auditory nerves and produce the sensation of hearing. *b)* the auditory sensation produced by such vibrations.

In the first place, this definition is highly theoretical, in that it relies on the scientific view that sounds are vibrations. In the second place, it turns out to be quite circular, since the dictionary defines "auditory" in terms of "hearing", and "hearing" in terms of perceiving sounds. Such circularity is no problem for the account of meaning discussed in chapter 4.

Rules generally specify what is characteristic of typical objects, not what is universally true of them. Through the critiques of Wittgenstein (1953) and Putnam (1975) in philosophy, Rosch (1973) in psychology, and Minsky (1975) in artificial intelligence, the traditional notion of concepts as defined by necessary and sufficient conditions has been discredited. Wittgenstein pointed out that there are no definitions that capture all and only the instances of complex concepts such as "game". Such definitions are rarely to be found outside mathematics. The experiments of Rosch and others showed that peoples' concepts are organized around prototypes: a robin, for example, is a more prototypical bird than an ostrich. Minsky

argued that for computational flexibility concepts should be represented as frames that describe typical or idealized instances. Accordingly, the rules in PI provide a rough description of what is typical of sounds, not a definition of them. The traditional notion of concepts as fully defined generates a misleadingly strict account of their meaning, a point that will be important for later discussions of how concepts become meaningful (chapter 4) and of incommensurability of conceptual schemes (chapter 5).

Why does PI use messages, rules, and concepts with so much structure? The justification for complicating these structures is simply to be able to use them in complex processes: they support a far more elaborate and interesting model of problem solving and inductive inference than would otherwise be possible. The question of whether a computational model of thinking must have separate structures corresponding to concepts is controversial, and the important cognitive architectures of Anderson (1983) and Laird, Rosenbloom, and Newell (1986) do not have them. I shall argue that they are an important part of a theory of cognition.

2.3.2. Problem Solving

Problem Solving and Spreading Activation of Concepts PI's central activity is problem solving. Given a set of starting conditions and goals, it fires rules that will lead from the starting conditions to the goals. Here is PI's simple representation of the problem of explaining why sound propagates and reflects:

Name:	explain_sound
Data-type:	problem
Start:	(sound ($x) true)
Goals:	(reflect ($x) true) (propagate ($x) true)
Problem-type:	explanation
Activation:	1

The solution to such a problem is a sequence of rule firings, in this case leading from the supposition that some **$x** is an arbitrary instance of sound to the conclusion that it reflects and propagates. Once the system has the wave theory of sound, the explanation can be a straightforward application of the rules that sounds are waves and that waves propagate and reflect. However, deciding what rules to fire depends on many nonlogical issues, such as what rules are available from memory, what rules are strongest in the sense of having the best record of success, and what rules appear to be most relevant to the current situation. (For a detailed discussion of the operation of these factors in rule-based systems, see Holland et al., 1986).

When people solve problems, only some of the relevant information is available to them in memory at any given time. PI models the varying

accessibility of elements in the memory of an individual scientist by a process of spreading activation of concepts and rules. At any given time, only some of the total set of concepts are active and only some of the total set of rules are available for firing. Rules are attached to concepts: as we saw, attached to the concept of sound are rules such as that if x is sound and y is some person near x, then y hears x. Also attached to the concept of sounds are messages encoding facts about particular sounds, such as that a particular sound is loud. PI matches all the rules from active concepts against all the messages from active concepts; rules whose conditions are matched then become candidates for firing. Any number of rules can be fired at a time, which simulates parallel processing. (Parallelism is computationally and epistemological important; see chapter 10.) When a rule is fired, the concepts used in its action become active. So if the rule "If x is a dog, then x has fur" is matched by the message "Lassie is a dog", then the new message "Lassie has fur" will be produced, and, equally important, the concept of fur will become active. Hence at the next timestep, new sets of messages and rules about fur will become active. Activation can also spread backward from the goal to potentially useful concepts and rules. In addition, in the current version of PI (in contrast to the version described in Holland et al., 1986), activation spreads automatically up and down the conceptual hierarchy, for example, from sound up to its superordinates sensation and physical phenomenon and down to its subordinates music, voice, whistle, and bang. The process of rule firing and spreading activation of concepts continues until the goals of the problem have been accomplished. This process is summarized in figure 2.2.

PI solves the problem of explaining the propagation and reflection of sound by forming a wave theory of sound. Just how this occurred to the ancient Greek or Roman who first discovered the wave theory of sound is unknown, but fragments from Chrysippus (Samburski, 1973) and Vitruvius (1960) suggest that an association was made between sound and water waves. PI has simulated various ways in which the concept of sound and wave might have become simultaneously active, for example, through associations from sound to music to instruments to strings to vibrations to waves. PI's solution of the problem of explaining why sound propagates and reflects proceeds by rule firings and spreading activation, including activation of the concept of a wave that makes possible formation of the hypothesis that sound is a wave. One chain of associations from sound to wave that has been simulated in PI is depicted in figure 2.3. Rule firings are indicated by arrows, and spreading of activation to subordinates and superordinates is indicated by vertical lines. In this simulation, activation spreads from sound down to its subordinate music, and down again to instrumental music. Then the rule that instrumental music is played by an instrument fires, and stringed-instrument is activated as a subordinate of instrument.

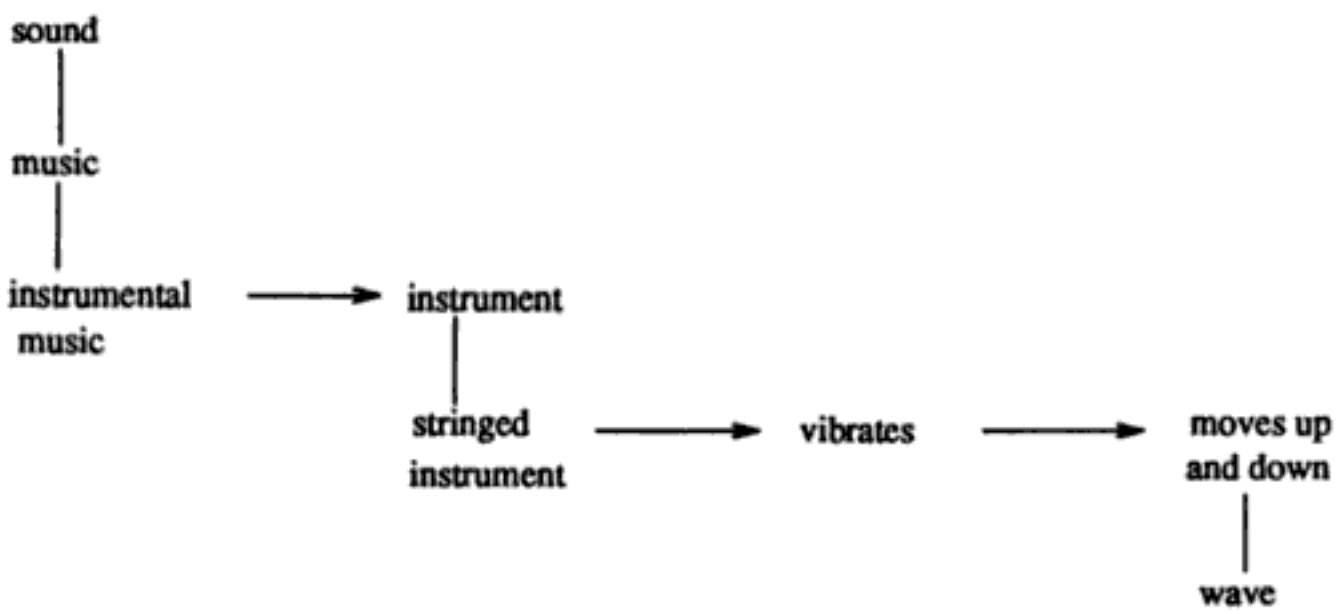


Figure 2.3
Spreading activation from sound to wave.

Two rules then fire: stringed instruments vibrate, and what vibrates moves up and down. Finally, the link that waves are a kind of moving up and down leads to activation of the concept of a wave. See appendix 3 for a fuller description of PI's run on this example. Clearly, this is only one of many chains of association that might have occurred when the wave theory of sound was initially discovered by the ancient Greeks. Moreover, in a more realistic simulation, many other activations and rule firings would also occur simultaneously with the ones just described. Nevertheless, the simulation gives some idea of how an association between sound and wave might occur during an attempt to explain why sound propagates and reflects.

Analogical Problem Solving Problem solving can be greatly facilitated by the use of past successful problem solutions. Keith Holyoak and I have adapted PI's mechanism of direct spreading activation to provide an analysis of how old problem solutions can be used to solve new problems (Holyoak and Thagard, 1986). The two key questions in analogical problem solving are (1) How, while solving a problem, does one retrieve relevant existing problem solutions? (2) How, once a relevant problem solution is found, does one exploit the analogy between them? In PI, directed spreading activation provides similar answers to both these questions.

We now have running a highly simplified simulation of the ray problem of Duncker (1945). The ray problem consists of figuring out how to use a ray source to destroy a tumor inside a patient, when radiation at full strength will destroy flesh between the source and the tumor, leading to the death of the patient. Subjects have great difficulty coming up with a solution to this problem (Gick and Holyoak, 1980, 1983), but their performance is greatly improved when they are first told of an analogous problem. The fortress problem consists of trying to figure out how an army

can capture a fortress when a frontal attack by the whole army is impossible. One solution is to split up the army and have it attack the fortress from different sides. This solution suggests an analogous solution for the ray problem, leading to irradiation of the tumor with lower intensity rays from different directions.

Our current simulation models analogical problem solving in the following steps. First, the base problem (here the fortress problem) must be solved, and its solution stored by association with the concepts mentioned in its problem description. The solved fortress problem, for example, is represented by the following structure:

Name:	capture_fortress
Data-type:	problem
Start:	(army (obj_1) true) (fortress (obj_2) true) (road (obj_3) true) (between (obj_3 obj_1 obj_2) true)
Goals:	(capture (obj_1 obj_2) true) (destroyed (obj_1) false)
Activation:	1
Concepts-attached-to:	(army fortress roads between capture destroyed)
Rules-used:	rule_1_army, etc.
Effectors:	(split (obj_1) true) (move-separately-to (obj_1 obj_2) true)

Second, solution of the target problem (here the ray problem) is attempted. This begins directed spreading activation in two directions: forward from concepts mentioned in the starting conditions of the target problem by rule-firing, and backward from the concepts mentioned in the goal conditions. Third, this process of rule-firing leads to activation of concepts to which the fortress problem has been attached. Figure 2.4 shows one possible path of activation that PI has been used to simulate. Here an association from ray to shoot to shoot-bullet to gun to weapons to fight to conflict to battle to army leads to activation of the concept **army**. Some of these associations are by firing of rules, such as that rays can shoot, while others are by subordinate/superordinate relations, for example, from fight to its superordinate conflict and down to another subordinate, battle. Thanks to PI's simulated parallelism, at the same time an association from the goal of destroying the tumor leads from destroy to defeat (since one way of destroying something is to defeat it) and then to conquer and capture. Since the stored solution of the fortress problem is attached to the newly activated concepts of army and capture, it gradually accumulates

ray problem. The analogy with the fortress problem does not provide a complete solution to the ray problem, but it does suggest potentially key steps in its solution.

After PI solves a problem analogically, producing one solution using a previous one, it constructs an analogical *schema*, which is an abstraction from the two previous solutions (for an introduction to the psychological notion of a schema, see tutorial D). Since the fortress problem has contributed to a solution to the ray problem, PI examines the statement of the two problems to see what they have in common. Using the rules stored with the concepts of the respective problems, it attempts to derive an abstract version of the two problems. In the fortress and ray problems, there is enough similarity to produce the following structure:

Name:	capture-fortress/destroy-tumor
Data-type:	problem schema
Start:	(force ($x) true)
	(target ($y) true)
Goals:	(overcome ($x $y) true)
Effectors:	(split ($x) true)
	(move_separately_to ($x $y) true)

This structure is then associated with the relevant concepts, such as **force**, and is available for future analogical problem solving. The schema, however, is potentially much more usable than the two problems from which it was formed, since it will be easier to map a new concrete problem to this abstraction than to the fortress or ray problems. Any new problem whose concepts sufficiently activate the concepts of force, target, and overcome will be be able to exploit the possible solution of splitting the force.

The processes just described simulate many of the experimental results of Duncker (1945) and of Gick and Holyoak (1980, 1983). Holyoak and Thagard (1986) describe how this model can account for such experimental results as the effectiveness of hints in problem solving, the efficacy of problem schemas, and the fact that structural similarities (ones that play a causal role in determining solutions) are more important than surface similarities in analogical transfer.

Extralogical Processes A logician will naturally ask, Why bother with all this spreading activation? Why not just take the logical consequences of current beliefs and add them to the set of beliefs? We have already seen that no finite memory could handle such a procedure. A system must not clutter up a finite memory with useless facts. Still, one might argue that it would be more elegant to consider, at each timestep, *all* of the messages and rules stored in memory. The computational problem with this suggestion is simply that there are too many of them in a large system such as a human

BRADFORD BOOKS

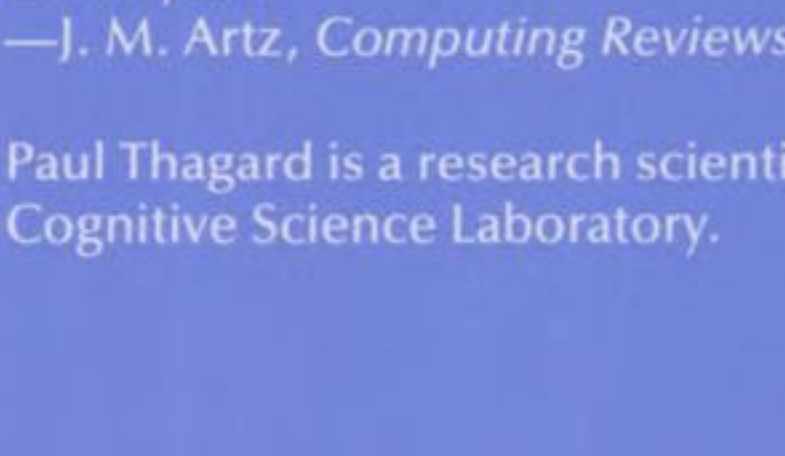
Computational Philosophy of Science
Paul Thagard

By applying research in artificial intelligence to problems in the philosophy of science, Paul Thagard develops an exciting new approach to the study of scientific reasoning, using computational ideas to shed light on how scientific theories are discovered, evaluated, and used in explanations. He describes a detailed computational model of problem solving and discovery that provides a conceptually rich yet rigorous alternative to accounts of scientific knowledge based on formal logic, and he uses the model to illuminate such topics as the nature of concepts, hypothesis formation, analogy, and theory justification.

"The writing reflects an enviable clarity of thought and economy of expression. Thagard has a remarkable ability to reduce complicated philosophical positions to their essential simplicity and state them in clear, flowing arguments. . . . To say that I liked this book would be an egregious understatement. Indeed, I read it twice and enthusiastically underlined nearly half of it in the process. . . . [T]he must-read book of the year."
—J. M. Artz, *Computing Reviews*

Paul Thagard is a research scientist at the Princeton University Cognitive Science Laboratory.

The MIT Press
Massachusetts Institute of Technology
Cambridge, Massachusetts 02142



THACP
0-262-70048-4