The Architecture of Cognition

John R. Anderson

This page intentionally left blank

This page intentionally left blank

The Architecture of Cognition

John R. Anderson

Psychology Press
Taylor & Francis Group
New York London

First Published by
Lawrence Erlbaum Associates, Inc., Publishers
10 Industrial Avenue
Mahwah, New Jersey 07430

Transferred to Digital Printing 2009 by Psychology Press
270 Madison Ave, New York NY 10016
27 Church Road, Hove, East Sussex, BN3 2FA

Originally published 1983.

Library of Congress Cataloging-in-Publication Data

Anderson, John Robert 1947-
The architecture of cognition.
p. cm.
Includes bibliographical references and index.
ISBN 0-8058-2233-X (pbk. : alk. paper)
1. Cognition--Data processing. 2. Human information processing. 3. Digital computer simulation. I. Title.
BF311.A5894 1983 153 82-21385

Publisher's Note
The publisher has gone to great lengths to ensure the quality of this reprint but points out that some imperfections in the original may be apparent.

To Russ and J.J.,
from whom I have learned much
about the evolution and development
of cognition

This page intentionally left blank

Preface

In the mid 1960s, when I was an undergraduate, three of the active areas of research in psychology were learning theory, psycholinguistics, and cognitive psychology. At that time there was no coherent connection among the three, but the question of how language could be acquired and integrated with the rest of cognition seemed interesting. However, there was no obvious way to tackle the question because the field just did not have the relevant concepts. There were the options of pursuing a graduate career in learning theory, cognitive psychology, or psycholinguistics. I chose cognitive psychology and I believe I chose wisely (or luckily).

When I went to Stanford in 1968, Gordon Bower assigned me a hot issue of the time: to understand the categorical structure in free recall. As we analyzed free recall it became clear that we needed a complete model of the structure of human memory, with a particular focus on meaningful structure. Bower and I worked on this for a number of years, first developing the model FRAN (Anderson, 1972), then HAM (Anderson and Bower, 1973). Through this work we came to appreciate the essential role of computer simulation in developing complex models of cognition. HAM, the major product of that effort, was a complete model of the structures and processes of human memory, having as its central construct a propositional network representation. It successfully addressed a great deal of the memory literature, including a fair number of experiments on sentence memory that we performed explicitly to test it.

Critical to predicting a particular experimental phenomenon was deciding how to represent the experimental material in our system. Usually our intuitions about representation did lead to correct predictions, but that was not a very satisfactory basis for

prediction. The memory system had to be understood as a component of a more general cognitive system. The reason for choosing a particular representation lay in the properties of the general system. For instance, choice of a representation for a sentence depended on the operation of a natural-language parser, which, unlike the facts represented in HAM, is a skill. There is a fundamental distinction between declarative knowledge, which refers to facts we know, and procedural knowledge, which refers to skills we know how to perform. HAM, a theory of the declarative system, was incomplete because the representation chosen in the declarative system depended on operations of components of the procedural system such as the parser.

The first step to developing a more complete theory was identifying an appropriate formalism to model procedural knowledge. My first effort was to use the augmented transition networks (ATNs), which had seemed so appropriate for natural-language processing. A theory of language acquisition was developed in that framework. However, ATNs proved to be at once too restrictive as computational formalisms and too powerful as models of human cognition. These difficulties led me to consider the production-system models Allen Newell had been promoting, which had some similarities to ATNs. I have to admit that at first, when I focused on surface features, production systems seemed unattractive. But as I dug deeper, I became more and more convinced that they contained key insights into the nature of human procedural knowledge.

The ACT system, the product of my three years at Michigan, 1973–1976, and described in Anderson (1976), was a synthesis of the HAM memory system and production-system architecture. In this system a production system was used for the first time as an interpreter of a propositional network. ACT also borrowed the spreading activation concept from other researchers, such as Collins and Quillian. Its production system operated on a working memory defined by the active portion of its long-term memory network. In a computer simulation version of ACT, called ACTE, it was possible to "program" production sets that modeled various tasks. Much of my 1976 book describes such production sets that modeled various small tasks.

However, although ACTE answered the question of where propositional representations came from—they came from the actions of productions—now the natural question was, where do productions come from? This led to the issue of a learning theory for production systems. Paul Kline, Charles Beasley, and

I then developed ACTF, the first production system to contain an extensive theory of production acquisition. A viable version of ACTF was running in 1977.

After moving to Carnegie-Mellon in 1978, I spent the next four years trying to "tune" the ACT system and make it address language acquisition. My attempts produced some major reorganizations in the theory, including changes in the spreading activation mechanisms, a theory of production pattern matching, augmentations to the architecture to handle goals, and additions to the production-learning mechanisms, all of which are described in this book. Paul Kline and David Neves collaborated in the early part of this effort. In the later years I was guided by the A.C.T. research group, which has included Frank Boyle, Gary Bradshaw, Renee Elio, Rob Farrell, Bill Jones, Matt Lewis, Peter Pirolli, Ron Sauers, Miriam Schustack, and Jeff Shrager. I was able to finally apply ACT to language acquisition. The new version of ACT is called ACT* (to be read ACT-star). The theory has evolved from the original concern with categorical free recall to address an ever-widening set of questions. Each theory along the way raised questions that the next one answered. Finally, the widening circles have expanded to encompass my original interest in learning theory, psycholinguistics, and cognitive psychology. While it is undoubtedly not the final theory, it achieves a goal set fifteen years earlier.

ACT* is a theory of *cognitive architecture*—that is, a theory of the basic principles of operation built into the cognitive system. ACT stands for Adaptive Control of Thought. It is worth reviewing what this title means and why it is apt. First, this theory concerns higher-level cognition or thought. A major presupposition in this book is that higher-level cognition constitutes a unitary human system. A central issue in higher-level cognition is control—what gives thought its direction, and what controls the transition from thought to thought. As will become apparent, production systems are directed at this central issue. A major concern for me has been to understand the principles behind the control of thought in a way that exposes the adaptive function of these principles. Ultimately, understanding adaptive function brings us to issues of human evolution, which are largely excluded from this book (but see Anderson, 1982c).

It needs to be emphasized that production systems address the issue of control of cognition in a precise way that is relatively unusual in cognitive psychology. Other types of theoretical analyses may produce precise models of specific tasks, but

how the system sets itself to do a particular task in a particular way is left to intuition. In a production system the choice of what to do next is made in the choice of what production to execute next. Central to this choice are the conflict resolution strategies (see Chapter 4). Thus production systems have finally succeeded in banishing the homunculus from psychology.

In writing this book I have been helped by a great many people who provided comments on drafts. Gary Bradshaw, Bill Chase, Renee Elio, Ira Fischler, Susan Fiske, Jim Greeno, Keith Holyoak, Bill Jones, Paul Kline, Steve Kosslyn, Matt Lewis, Brian MacWhinney, Michael McCloskey, Allen Newell, Jane Perlmutter, Rolf Pfeifer, Steven Pinker, Peter Pirolli, Zenon Pylyshyn, Roger Ratcliff, Lance Rips, Paul Rosenbloom, Miriam Schustack, and Jeff Shrager have commented on one or more of these chapters. Robert Frederking, Jay McClelland, Ed Smith, and two anonymous reviewers have read the manuscript in its entirety. Eric Wanner has been an exceptional editor—reading and commenting on the entire book. Lynne Reder has gone through this book with me, word by word and idea by idea, exposing weaknesses and providing a number of the ideas developed here. The A.C.T. research group, as well as my "Thinking" classes, have gone over these ideas many times with me.

Many people have worked hard to get the programs and experiments running. My graduate students, Gary Bradshaw, Renee Elio, Bill Jones, Matt Lewis, Peter Pirolli, and Miriam Schustack, have been valuable collaborators. Takashi Iwasawa and Gordon Pamm wrote countless experiment-running programs and data analyses; Barbara Riehle and Rane Winslow greatly assisted in testing subjects; and Takashi Iwasawa and Frank Boyle did much work on the geometry project. Two outstanding undergraduates, Rob Farrell and Ron Sauers, have been responsible for the GRAPES simulation of the acquisition of LISP programming skills. Among her many other duties, Rane Winslow had primary responsibility for preparing and coordinating this manuscript, and I am deeply grateful to her for all of her resourcefulness and hard work. Monica Wallace was responsible for the last months of shepherding the book into publication.

Finally, I want to thank those government agencies who have done so much to support my research. Research grants from NIE and NIMH were critical in the development of ACTE and ACTF, and a current grant from the Information Sciences branch of NSF (IST-80-15357) has supported the geometry

project. My simulation work has taken place on the SUMEX facility at Stanford, supported by grant P41RR-00785-08 from NIH; on Carnegie-Mellon's computer facilities, supported by contract F33615-81-K-1539 from ARPA; and on our Psychology VAX, purchased through grant BNS-80-13051 from NSF. My language acquisition research is supported by a contract from ONR. I would like to express special appreciation to the Personnel and Training Research Program of ONR (Marshall Farr, Henry Halff) and to the Memory and Cognition Program at NSF (Joe Young). These two groups have been long-standing supporters of the ACT research and have provided the stable funding that a project of this scale needs; the current sources of support are N00014-81-C-0335 from ONR and BNS-82-08189 from NSF. Marshall Farr, Henry Halff, and Joe Young have also provided valuable input to the research.

This page intentionally left blank

Contents

This page intentionally left blank

1 | Production Systems and ACT

A Unitary Theory of Mind and Production Systems

THEORISTS ARE STRONGLY influenced by their various preconceptions. The most deeply rooted preconception guiding my theorizing is a belief in the unity of human cognition, that is, that all the higher cognitive processes, such as memory, language, problem solving, imagery, deduction, and induction, are different manifestations of the same underlying system. This is not to deny that there are many powerful, special-purpose "peripheral" systems for processing perceptual information and coordinating motor performance. However, behind these lies a common cognitive system for higher-level processing. Moreover, the essence of what it is to be human lies in the principles of this core, higher-level system. We may not differ from the many mammalian species to which we are related in our peripheral perceptual and motor processes, but we assuredly do differ in our complex thought patterns and our intelligence.

The view that the mind is unitary is certainly not universally held; it may not even be a majority opinion. To quote the most noted proponent of the alternative view:

> We may usefully think of the language faculty, the number faculty, and others as "mental organs," analogous to the heart or the visual system or the system of motor coordination and planning. There appears to be no clear demarcation line between physical organs, perceptual and motor systems, and cognitive faculties in the respects in question. In short, there seems little reason to insist that the brain is unique in the biological world, in that it is unstructured and undifferentiated, developing on the basis of uniform principles of growth or learning—say those of some

> learning theory, or of some yet-to-be conceived general-purpose learning strategy—that are common to all domains. (Chomsky, 1980, p. 3)

This faculty approach holds that distinct cognitive principles underlie the operation of distinct cognitive functions. The unitary approach holds that all higher-level cognitive functions can be explained by one set of principles. In some ways the faculty approach seems just plain common sense. Many cognitive theories, extending back at least to the phrenology of Gall (see Boring, 1950, for a discussion of this and the more "reputable" faculty theories) have held this view. Its truth might almost seem a tautology: clearly we perform different intellectual functions, so it might seem that we must have different faculties for these functions. The faculty proposals that have been advanced have always gotten into difficulties in their specifics, but it never has been clear whether there is anything fundamentally wrong with the faculty approach.

The early proposals for unitary systems (for example, stimulus-response, or S-R, theories) were also shown to be basically inadequate. However, the unitary theory found an important metaphor in the modern general-purpose computer and, perhaps more significantly, in symbolic programming languages, which showed how a single set of principles could span a broad range of computational tasks. It also became clear that the set of computational functions was unlimited, meaning that general processing principles were essential to span broad ranges of tasks. It made no sense to create a special system for each conceivable function.

A number of candidates for a general system have been offered, including general problem solvers (Fikes and Nilsson, 1971; Newell and Simon, 1972; Sacerdoti, 1977), general inference systems (Green and Raphael, 1969; McDermott and Doyle, 1980; Robinson, 1967), and general schema systems (Bobrow and Winograd, 1977; Minsky, 1975; Rumelhart and Ortony, 1976; Schank and Abelson, 1977). My research has been predicated on the hypothesis that production systems provide the right kind of general computational architecture for achieving a unitary mental system. The particular line of production system theories I have developed all go under the name ACT. This book will describe a special ACT instantiation called ACT* (to be read ACT-star). As will become clear, ACT* is not just a random member of the ACT series. It is

the product I have been working toward for the past seven years. In ACT* the same core system if given one set of experiences develops a linguistic facility, if given another set of experiences develops a geometry facility, and if given another set of experiences develops a programming facility. Therefore ACT* is very much a unitary theory of mind.

Arguments for a Unitary Theory of Mind

One thing that distinguishes us from other creatures is our ability to acquire complex skills. All distinctively human activities—such as mathematics, language, chess, computer programming, sculpture—are acquired skills. There may be a significant innate component to their successful acquisition, but with the possible exception of language it is totally implausible to suggest that we have evolved special faculties or "organs" for mathematics, chess, computer programming, or sculpture. People become expert at activities for which there was no possibility of anticipation in our evolutionary history, and the essence of the human genius is just this plasticity. It is unnecessary to propose special organs for special abilities when we can fashion articulate abilities where there is no possibility of a prior organ. If all these abilities are fashioned from the same initial system (which hardly need be a tabula rasa), then in an important sense the adult human mind is a unitary construction.

Language is an important special case that might be the exception to the rule. It is not totally implausible to propose that it has had a long evolutionary history in which various language-specific adaptations have occurred. However, it seems more plausible that the language-specific adaptations are few and minor, that the language faculty is really the whole cognitive system. In our evolution we may have developed or enhanced certain features to facilitate language, but once developed, these features were not confined to language and are now used in nonlinguistic activities. Thus the mind is a general pool of basic structures and processes, which has been added to under evolutionary pressure to facilitate language. The additions have been used in skills, for example, computer programming, that were not anticipated in the original evolutionary developments. Part of the evidence for this view are the remarkable communalities between language and other skills, which will be discussed later in this book.

There is a tendency to regard the existence of "language areas" and other localizations of function in the brain as strong

evidence for faculties. However, there is nothing necessary about this inference, as shown by a computer analogy: two programs can occupy different areas of computer memory, much as two different cognitive abilities might lie in two separate regions of the brain. However, the two programs may have identical principles. For instance, I can have one ACT simulation doing language and another doing geometry. Thus, there need be no connection between distinct physical location and distinct cognitive principles. The real issue concerns the uniqueness of the structure and processes underlying cognitive functions, not their physical location.[1]

Another major reason for not believing in an organ for language or for other cognitive activities is that the boundaries between these organs cannot be drawn a priori. It is pretty clear where the activity of the lung leaves off and that of the circulatory system takes over, but this cannot really be said for cognitive faculties. The lung and the heart are both involved in an activity such as running, but it is possible to identify their distinctive contributions. It has been proposed that there is a language faculty, a number faculty, a deduction faculty, and a problem-solving faculty, but if there are such faculties, their activities are terribly intertwined in a task like computer programming. When we look at an expert programmer creating a program, we cannot separate the contributions of the various faculties. Indeed, if we applied any reasonable criterion for individuating faculties, we would have to conclude that computer programming was a separate faculty. This is because some of the core principles for this skill organization, such as strategies for creating recursive programs, apply across the entire range of programming behaviors and are seldom if ever evoked elsewhere. Since it is nonsense to suggest a programming faculty, we should be more skeptical of other proposed faculties.

An expert's execution of a skill is special in that a strong task-specific cognitive organization has developed through extensive experience. This is not the case with the novice, but analysis of novice behavior gives no more comfort to the faculty approach. The remarkable feature of novices is that they are able to put together so many different facets of knowledge to solve a task. A novice programmer brings together recent facts learned about programming and the programming language, facts from mathematics, real-world experiences as analogies, general problem-solving skills, deductive strategies, linguistic analyses —all to solve the problem. The novice's attempts at synthesizing this knowledge can be terribly off target, but this is only for

lack of the right knowledge, not because of a fundamental incompatibility of the knowledge categories. What is remarkable is the ease with which novices switch among categories and the sheer impossibility of identifying where one faculty might begin and another end. Compartmentalization is similarly impossible in the case of language use (see Schank and Birnbaum, in press).

In summary then, there are three lines of evidence for the unitary approach. One is the short evolutionary history of many of the higher human intellectual functions, such as those concerned with mathematical problem solving. The second is that humans display great plasticity in acquiring functions for which there was no possibility of evolutionary anticipation. The third is that the various cognitive activities have many features in common.

I would like to head off two possible misinterpretations of my position. First, the unitary position is not incompatible with the fact that there are distinct systems for vision, audition, walking, and so on. My claim is only that higher-level cognition involves a unitary system. Of course, the exact boundaries of higher-level cognition are a little uncertain, but its contents are not trivial; language, mathematics, reasoning, memory, and problem solving should certainly be included. Second, the unitary position should not be confused with the belief that the human mind is simple and can be explained by just one or two principles. An appropriate analogy would be to a programming language like INTERLISP (Teitleman, 1976), which is far from simple and which supports a great variety of data structures and functions. However, it is *general-purpose*, that is, one can use the same data structures and processes in programs for language and for problem solving. Individual programs can be created that do language and problem solving as special cases. In analogy to INTERLISP, I claim that a single set of principles underlies all of cognition and that there are no principled differences or separations of faculties. It is in this sense that the theory is unitary.

Production Systems: History and Status

Production system theories have gradually increased in prominence in psychology over the past decade. Their basic claim is that underlying human cognition is a set of condition-action pairs called productions. The condition specifies some data patterns, and if elements matching these patterns are in working memory, then the production can apply. The action

specifies what to do in that state. The basic action is to add new data elements to working memory. Informally stated, a typical production rule might be:

IF person 1 is the father of person 2
and person 2 is the father of person 3
THEN person 1 is the grandfather of person 3.

This production would apply if *Fred is the father of Bill* and *Bill is the father of Tom* were active in working memory. It would make the inference *Fred is the grandfather of Tom* and deposit this fact in working memory.

Production systems can be traced back to the proposals of Post (1943), but Post production systems bear little resemblance to current production systems except that the current condition-action pairs can be viewed as derived from the rewrite rules of Post's formalism. Production system theories are similar to stimulus-response theories in many ways but with important differences that remove the computational inadequacies of S-R theories. Most notably, the production is very much like the stimulus-response bond (Anderson, 1976; Newell and Simon, 1972). One might conceive of production systems as "cognitive" S-R theories, despite the contradiction in the connotations of these terms.

Modern production systems began with Newell's work at Carnegie-Mellon in the early sixties and Waterman's later dissertation work (1970) at Stanford. Right from the start production systems had an ambiguous status, being in part programming languages for computer science and in part psychological theories. A series of publications in the early seventies (Hunt and Poltrack, 1974; Newell, 1972, 1973; Newell and Simon, 1972) brought production systems to the awareness of psychologists.

The concept of a production system is vague, and it is hard to determine where its boundaries end and where other computer science formalisms or psychological theories begin. In computer science a more general category, sometimes called *pattern-directed systems* (see Waterman and Hayes-Roth, 1978), includes deduction systems like MYCIN (Shortliffe, 1976) and other types of architectures, such as schema systems and linguistic rewrite systems. As in production systems, control is in response to the appearance of data, but behavior does not necessarily involve adding elements to working memory. For instance, a major category of action in MYCIN was to update the probabilities of various medical diagnoses.

Production systems are not strictly a Carnegie-Mellon phenomenon, although the concentration there is undeniable (a recent report of a subset of the CMU research is contained in Klahr, Langley, and Neches, 1983). Besides my own work, which started elsewhere, the ideas have been used by Bower (1977), Brown and Van Lehn (1980), Collins (1977), Greeno (1976), Jeffries, Turner, Polson, and Atwood (1981), Kieras and Bovair (1981), Lewis (1978), and Ohlsson (1977). Although the idea is gradually gaining popularity, it is by no means a dominant construct in cognitive psychology, partly because of the technical difficulty both in creating production systems and in understanding someone else's production system. This is the price paid for the precision with which production systems model information processing; no other psychological theories have been as precise and detailed in their modeling of cognitive tasks.[2] As with any emerging scientific formalism, there is a lot to learn about the human engineering of the formalism and about communicating to the scientific community the essential results of work with that formalism. Over the past few years there has been progress in this direction.

An Example of a Production System

An example of a production system that performs a specific task will be useful as a concrete referent for interpreting more abstract points that will be made later.

Tracing the Behavior

An example set of productions for performing addition is given in Table 1.1. This example assumes that the subject has memorized the addition table. The conditions of these productions are given by the IF part and their actions by the THEN part.[3] Figure 1.1 illustrates the flow of control among the productions in that set. Application of these productions is controlled by the setting of goals; Figure 1.1 basically shows which productions respond to which goals and which productions set which goals. It is easiest to understand such a production system by tracing its application to an example problem such as the following addition problem:

614
438
683

Production P1, the first to apply, would set as a subgoal to iterate through the columns. Production P2 then changes the sub-

Table 1.1 *A production system for performing addition*

P1 IF the goal is to do an addition problem
THEN the subgoal is to iterate through the columns of the problem.

P2 IF the goal is to iterate through the columns of an addition problem
and the rightmost column has not been processed
THEN the subgoal is to iterate through the rows of that rightmost column
and set the running total to 0.

P3 IF the goal is to iterate through the columns of an addition problem
and a column has just been processed
and another column is to the left of this column
THEN the subgoal is to iterate through the rows of this column to the left
and set the running total to the carry.

P4 IF the goal is to iterate through the columns of an addition problem
and the last column has been processed
and there is a carry
THEN write out the carry
and POP the goal.

P5 IF the goal is to iterate through the columns of an addition problem
and the last column has been processed
and there is no carry
THEN POP the goal.

P6 IF the goal is to iterate through the rows of a column
and the top row has not been processed
THEN the subgoal is to add the digit of the top row to the running total.

P7 IF the goal is to iterate through the rows of a column
and a row has just been processed
and another row is below it
THEN the subgoal is to add the digit of the lower row to the running total.

P8 IF the goal is to iterate through the rows of a column
and the last row has been processed
and the running total is a digit
THEN write the digit
and delete the carry
and mark the column as processed
and POP the goal.

Table 1.1 (*continued*)

P9	IF the goal is to iterate through the rows of a column and the last row has been processed and the running total is of the form "string digit" THEN write the digit and set carry to the string and mark the column as processed and POP the goal.
P10	IF the goal is to add a digit to another digit and a sum is the sum of the two digits THEN the result is the sum and mark the digit as processed and POP the goal.
P11	IF the goal is to add a digit to a number and the number is of the form "string digit" and a sum is the sum of the two digits and the sum is less than 10 THEN the result is "string sum" and mark the digit as processed and POP the goal.
P12	IF the goal is to add a digit to a number and the number is of the form "string digit" and a sum is the sum of the two digits and the sum is of the form "1 digit*" and another number sum* is the sum of 1 plus string THEN the result is "sum* digit*" and mark the digit as processed and POP the goal.

goal to adding the digits of the rightmost column and sets the running total to 0. Then production P6 sets the new subgoal to adding the top digit of the row (4) to the running total. In terms of Figure 1.1, this sequence of three productions has moved the system down from the top goal of doing the problem to the bottom goal of performing a basic addition operation. The system has four goals stacked, with attention focused on the bottom goal.

At this point production P10 applies, which calculates 4 as the new value of the running total. In doing this it retrieves from the addition table the fact that $4 + 0 = 4$. Production P10 also pops the goal of adding the digit to the running total. Popping a goal means shifting attention from the current goal to the one above it in the hierarchy. In this situation, attention will return to iterating through the rows of the column. Then P7 ap-

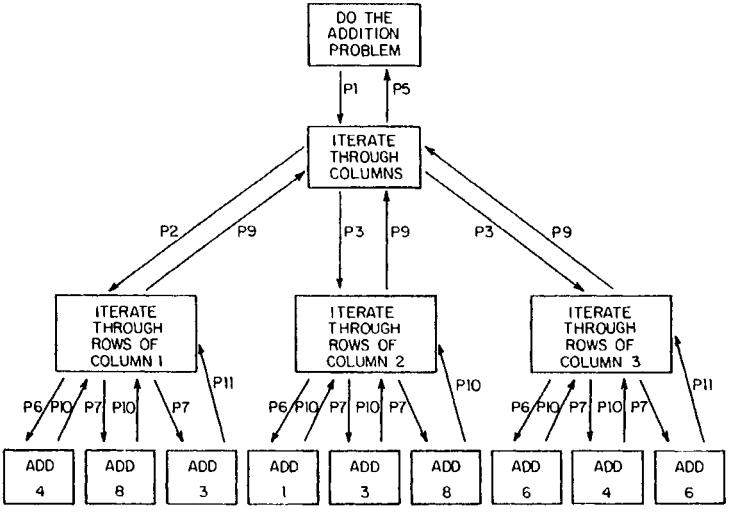

Figure 1.1 *A representation of the flow of control in Table 1.1 among the various goals. The boxes correspond to goal states and the arrows to productions that can change these states. The goal at the origin of the production corresponds to the goal that elicits the production, and the goal at the terminus of the arrow corresponds to the goal that is set after the application of the production. Control starts with the top goal.*

plies, which sets the new subgoal of adding 8 to the running total. P10 applies again to change the running total to 12, then P7 applies to create the subgoal of adding 3 to the running total, then P11 calculates the new running total as 15. At this point the system is back to the goal of iterating through the rows and has processed the bottom row of the column. Then production P9 applies, which writes out the 5 in 15, sets the carry to the 1, and pops back to the goal of iterating through the columns. At this point the production system has processed one column of the problem. I will not continue to trace the application of this production set to the problem. Note that productions P2–P5 form a subroutine for iterating through the columns, productions P6–P9 an embedded subroutine for processing a column, and productions P10–P12 form an embedded subroutine for adding a digit to the running total.

Significant Features

Productions form the system's *procedural* component. For a production to apply, the clauses specified in its condition must

be matched against the information active in working memory. This information is part of the system's *declarative* component. Not everything the system knows is in working memory; information must be retrieved from long-term memory (the system's main declarative component) and deposited in working memory. The example above required the retrieval of addition facts like 4 + 8 = 12.

Both the condition and the action of each production consist of a set of clauses (basically each line of a production corresponds to a clause). The set of clauses in the condition specifies a total pattern that must be matched for the production to apply. That is, there must be a separate data clause in working memory matching each condition clause. The action clauses specify separate actions to be taken. Most actions add to the contents of working memory, but some cause external behavior. The actions are executed in sequence.

Productions contain variable slots, which can take on different values in different situations. The use of these variables is often implicit, as in Table 1.1, but in some cases it is important to acknowledge the variables that are being assumed. As an illustration, if production P9 from Table 1.1 were written to expose its variable structure, it would have the form shown below, in which the terms prefixed by LV are local variables.[4]

```
    IF the goal is to iterate through the rows of LVcolumn
       and LVrow is the last row of LVcolumn
       and LVrow has been processed
       and the running total is of the form "LVstring LVdigit"
THEN write LVdigit
       and set carry to LVstring
       and mark LVcolumn as processed
       and POP the goal.
```

Local variables can be reassigned to new values each time the production applies. For instance, the terms LVcolumn, LVrow, LVstring, and LVdigit will match whatever elements lead to a complete match of the condition to working memory. Suppose, for instance, that the following elements were in working memory:

```
The goal is to iterate through the rows of column 2
Row 3 is the last row of column 2
Row 3 has been processed
Running total is of the form "2 4"
```

The four clauses in the production's condition would match this working-memory information, and the following variable bindings would be created:

LVcolumn = column 2
LVrow = row 3
LVstring = 2
LVdigit = 4

Local variables assume values within a production to match the condition and execute the action. After the production applies, variables are free to be rebound in another application.

PSYCHOLOGICAL COMMENT ON THE EXAMPLE

This example was meant mainly to illustrate the computational character of production systems. However, it should be emphasized that a set of conditional, goal-factored rules such as these can provide a fairly accurate model of arithmetic behavior (Brown and Van Lehn, 1980). That is, such rules can be considered the *units* of the skill. Brown and Van Lehn discuss how various errors or "bugs" in children's subtraction can be explained by the deletion of individual rules.

Frameworks, Theories, and Models

To understand how ACT relates to other ideas in the field, it is useful to make distinctions among the terms *framework, theory,* and *model.* A framework is a general pool of constructs for understanding a domain, but it is not tightly enough organized to constitute a predictive theory. However, it is possible to sample from this pool, tie the constructs together with additional details, and come up with a predictive theory. One might regard "information-processing psychology" as such a framework, although it is an especially loose one. Production systems are a more specific framework within the information-processing framework. There is a general ACT framework that is a further specialization, within which specific ACT theories have been created.

One cannot evaluate a framework according to the standard verificational logic associated with scientific theories. That is, the production system framework makes no unique empirical prediction that distinguishes it from, say, a schema system framework. Rather, one judges a framework in terms of the success, or fruitfulness, of the theories it generates. If the theories lead to many accurate accounts of interesting phenomena, the framework is regarded as fruitful.

A theory is a precise deductive system that is more general than a model. Specific production systems such as Newell's (1973) system and the 1976 ACT (Anderson, 1976) system, are theories. A model is the application of a theory to a specific phenomenon, for instance, performance of a mental arithmetic task. Thus, the production set in Table 1.1 constitutes a modest model.

Production systems are particularly general in that they claim to be *computationally universal*—capable of modeling all cognitive activity.[5] One consequence of computational universality is that a production system can accomplish the same task in a number of different ways.[6] This is a point in favor of production system theory, because we know that people are capable of performing a single task in various ways. If the task is complex, different people will do it differently, and the same person will even behave differently on different occasions. However, because of their plasticity, production systems do not make unique predictions about how a task will be performed. Instead, the predictions are largely implicational: if a subject performs a task in such-and-such a way, then such-and-such behavior will be observed.

Although subjects can perform tasks in many ways, one or a few methods are usually preferred. It is reasonable to ask why a person performs a task in the way he or she does. To answer this question one needs a learning theory to specify that given a particular prior experience, the person will develop a certain production set that in a certain context will lead him to behave in a certain way, which will yield particular phenomena. Chapters 6 and 7 are concerned with such a learning theory for ACT.

The Neoclassical Production System

As noted earlier, the general category of production systems is a framework for theories. By now there are many theories to instantiate that framework. Newell's theories and ideas constitute a major subframework. Two major efforts have been made to establish his ideas as running computer systems: PSG (Newell and McDermott, 1975) and, more recently, OPS (Forgy and McDermott, 1977; Rychener and Newell, 1978). However, neither implementation is identical to Newell's basic ideas. His most recent statement about production system architecture (Newell, 1980) is different in many ways from either. Also, these implementations have features that are not relevant to the psychological status of the theory. Lenat and Harris (1978) have called the Newell system "neoclassical" architecture, and I will continue with that term. Other production systems, including

ACT, can all be seen as baroque variations on the neoclassical architecture in that they attempt to *add* to it to remedy perceived defects.

The neoclassical system places a heavy emphasis on simplicity. It is an attempt to make do with the bare minimum: production rules, a single uniform working memory, simple rules of conflict resolution, and not much else. The motivation for simplicity derives in part from the standard scientific desire for parsimony, but it also derives from the desire to facilitate the development of psychological mechanisms for production learning, which must be capable of producing the full space of productions that are used. If that space is simply structured, it is thought to be easier to define learning mechanisms capable of producing the required productions.

The neoclassical system emphasizes modular productions, in which each production is independent of the others and capable of being added, deleted, or separately modified. This means that production learning can proceed one production at a time without concern for interactions among the productions in a set.

Working Memory and Production Memory

Working memory in the neoclassical system consists of a number of slots, each of which holds one clause or element. An element is a list or relational structure built out of relations and arguments; (8 + 3 = 11) could be an element. Working memory, which can hold a limited number of elements, orders them according to recency. As new elements enter, old ones are pushed out. This conception shows the strong influence of the buffer model of Atkinson and Shiffrin (1968). The original limit on working memory was taken to be on the order of seven elements, reflecting the results of memory span tests. However, it often proves difficult to simulate behavior given this limit, and informal proposals have been made to increase it to twenty or more elements. In nonpsychological engineering applications of the OPS system (McDermott, 1981), the limit on working memory is simply eliminated.

Another feature of the neoclassical architecture is that the only long-term memory is production memory. There is not a separate declarative memory to encode facts like "George Washington was the first president of the United States" or the addition facts used in the simulation in Table 1.1. This is a major point where ACT differs from the neoclassical system. Newell would achieve the effect of a separate declarative mem-

ory by having one or more productions fire to deposit declarative information into working memory. For instance, a production responsible for retrieving this information might look like:

IF George Washington is mentioned
THEN note that he was the first president of the United States.

Parallelism, Seriality, and Conflict Resolution

In the earlier neoclassical architecture (but not in Newell, 1980), multiple productions may have their conditions matched, but only a single production can apply at a time. A set of *conflict resolution principles* determines which production will apply. The original conflict resolution principle in PSG involved a simple ordering of productions, with the highest-ordered production applying. The ordering was simply specified by the person who wrote the productions with the goal of making them function correctly. The conflict resolution principles in the OPS system, which seem more plausible, involve a combination of refractoriness, recency, specificity, and production ordering. Refractoriness prevents a production from repeating if it matches the same data structure, thus preventing most accidental looping. Recency, probably the most powerful principle, selects the production that matches the data element most recently added to working memory. If two or more productions match the same most recent element, then a test is performed to see which has the second most recent element, and so on. Should a tie remain after all the elements have been matched, the specificity principle is applied. Essentially this principle says that the production with more condition elements is preferred. If there still are competing productions (and this is very seldom the case), then the conflict is resolved by an ordering of the productions.

Because only a single production can apply during any cycle of the system, it is difficult to model the parallelism in human cognition. We can be simultaneously perceiving objects, driving a car, generating a sentence, and processing a conversation. Underlying a process like language comprehension are a large number of parallel, interacting processes such as perception, syntactic parsing, semantic interpretation, and inference (Anderson, Kline, and Lewis, 1977). In addition to the behavioral evidence for parallelism, it is clear that our neural hardware supports it. It is argued (J. A. Anderson, 1973) that the brain is not at all a serial computer because individual operations take relatively long times (10 msec or more) to perform. However,

the brain achieves computational power by doing many operations in parallel. The neoclassical theory depends heavily on parallel processing in its pattern matching, but parallelism must go beyond pattern matching.

Newell (1980) proposed a variant of production system architecture called HPSA77, which involved a major departure from the previous serial conception. He proposed it as an analysis of how the HARPY speech recognition system (Lowerre, 1976) could be achieved in a production system framework. HPSA77 is probably the best current instantiation of Newell's beliefs about production system architecture. He distinguished between productions that did not involve variables and those that did. He proposed that productions without variables could apply in parallel without restriction and that productions with variables had a limited serial restriction. His basic claim was that there was a variable-using mechanism that could handle the variables in only one instantiation of a production at a time. All limitations on parallelism came from this limitation; if the production involved no variables, there was no limitation; if it did involve variables, the production would have to "wait its turn" to use the variable mechanism.

Learning

Although many of its design decisions were motivated by learning considerations, the neoclassical system has not had a strong position on how learning proceeds. What has been implemented basically follows the ideas of Waterman (1974, 1975), who proposed that productions could deposit in working memory specifications as to the condition and action of a new production. Then a special BUILD operator would be called to create the production according to these specifications. The following production might cause new productions to be built to encode arithmetic facts:

IF the addition table reads "LVA + LVB = LVC"
THEN tag [the goal is to add LVA and LVB] as condition
and tag [the answer is LVC] as action
and BUILD.

If this production read "3 + 2 = 5" from the addition table, it would deposit in working memory the following clauses:

(CONDITION [the goal is to add 3 and 2])
(ACTION [the answer is 5]).

The BUILD operator finds all clauses in working memory tagged with CONDITION and all clauses tagged with ACTION. It makes the first set of clauses the condition and the second set the action. In the simple case above, the BUILD operator would produce:

IF the goal is to add 3 and 2
THEN the answer is 5.

Now whenever the goal is set to add 3 and 2, this production is there to execute and deposit the answer 5 in working memory.

The BUILD operator has enabled a lot of research on production learning, as has a similar *designation* process in ACT (see Anderson, Kline, and Beasley, 1980). However, production learning should be more automatic and less the result of deliberate strategy than it is with BUILD. As will be discussed at length in Chapter 6, deliberate production building has dangerous consequences and implies psychologically unrealistic skill acquisition capabilities. With the development of the chunking theory (Newell and Rosenbloom, 1981; Rosenbloom and Newell, 1983), the neoclassical architecture has the beginning of a theory of automatic production learning.

Performance Assumptions

In contrast to the uncertainty with respect to production learning, the theory of how existing production sets are executed is quite clear, and it addresses both errors and processing time. Errors in performance are due to failures of working memory; for instance, if a carry flag is lost from working memory, an error will be made in solving an addition problem. The time taken to execute a behavior is a linear function of the number of productions that apply and the number of actions in each production. Notably, processing time is not a function of the complexity of production conditions, even though most of the time involved in computer implementation is in matching the conditions. This reflects a belief in a very powerful parallel pattern-matching architecture.

The ACT Production System

An Overview

The system described in Anderson (1976) was ACTE. Subsequently we developed the ACTF system, which was described to some extent in Anderson, Kline, and Beasley (1977, 1980).

ACTF differed from ACTE in some details of the performance system, and principally in that it had a theory of production acquisition. Both systems were developed as simulation programs that embodied all the assumptions of the then-current theory. These simulations had the character of programming languages in which various ACT models could be implemented. The experience with ACTE and ACTF over the past several years has provided a basis for major reformulations of many aspects of the original theory to improve its psychological accuracy. Each reformulated aspect has been implemented and tested as a simulation program. Simulating the various aspects (in computer cycles) has become quite expensive as we have moved to assumptions that are less efficient in the computer simulation, although not necessarily in the operations of the human brain. Because of the computational cost, we have not created a general simulation that embodies all these assumptions at once.

One might think that by logical progression this system should be called ACTG. However, we call it ACT* to reflect the belief that it is the final major reformulation within the ACT framework. In previous ACT theories the performance subtheories were not fully integrated with the learning subtheories, but this gap is now closed. The assumptions of ACT* have also been revised to remedy the known difficulties of the earlier theories. The only part I feel is tentative concerns certain assumptions about the pattern matcher. These assumptions will be flagged when the pattern matcher is discussed in Chapter 4.

The previous paragraph implies that I expect ACT* to be wrong. This expectation is based not on any weaknesses in the theory, but on the nature of progress in science, which comes from formulating theories that account for a wide range of known phenomena and then finding out what is wrong with these theories. I regard a progression like the one from HAM (Anderson and Bower, 1973) through ACTE (Anderson, 1976) to ACT* as an example of reasonable scientific progress.

In my 1976 book on ACTE, I compared it to the HAM theory: "In completing the HAM book we had the feeling that we had more or less defined the HAM theory once and for all and that the major task ahead of us was to test its empirical consequences. I feel less certain that ACT has achieved its final shape as a theory. There remains much exploration to be done as to the potential of the theory and variations on it, largely because ACT is a theory of much broader generality than HAM and consequently has more potential to explore. In this book I have

caught the ACT theory in a stage of development and presented that [p. 3]." I now believe that ACT has reached the stage of development that HAM did. Except for further work on the pattern matcher, my plan for future research is to try to apply this theory wide and far, to eventually gather enough evidence to permanently break the theory[7] and to develop a better one. In its present stage of maturity the theory can be broadly applied, and such broad application has a good chance of uncovering fundamental flaws.

The General ACT Framework

Since ACT* is substantially revised over the earlier theories, one might wonder why I consider it as part of the ACT framework. To answer this question, we must look at the general framework of which all these theories are instantiations, as illustrated in Figure 1.2. An ACT production system consists of three memories: *working, declarative,* and *production.* Working memory contains the information that the system can currently access, consisting of information retrieved from long-term declarative memory as well as temporary structures deposited by encoding processes and the action of productions. Basically,

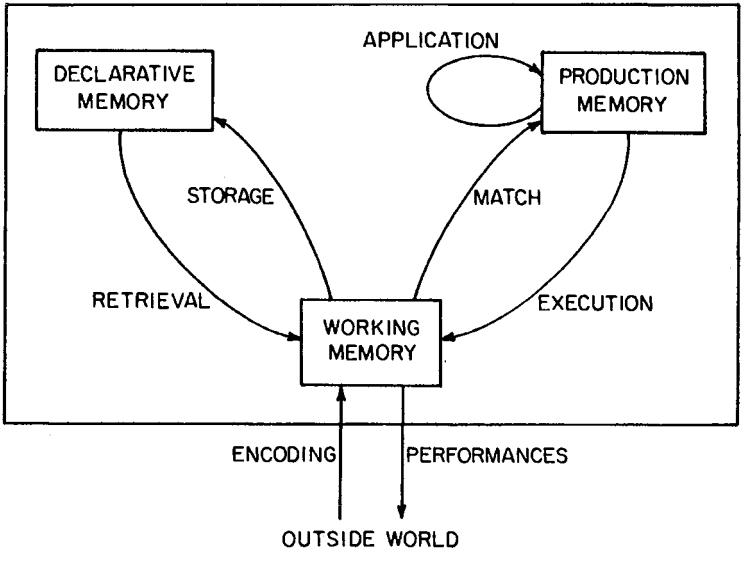

Figure 1.2 *A general framework for the ACT production system, identifying the major structural components and their interlinking processes.*

working memory refers to declarative knowledge, permanent or temporary, that is in an active state.

Most of the processes shown in Figure 1.2 involve working memory. *Encoding* processes deposit information about the outside world into working memory; *performance* processes convert commands in working memory into behavior. These two processes are not central to the ACT theory, unlike the other processes illustrated. The *storage* process can create permanent records in declarative memory of the contents of working memory and can increase the strength of existing records in declarative memory. The *retrieval* process retrieves information from declarative memory. In the *match* process, data in working memory are put into correspondence with the conditions of productions. The *execution* process deposits the actions of matched productions into working memory. The whole process of production matching followed by execution is referred to as *production application*. Note that the arrow called *application* cycles back into the production memory box, reflecting the fact that new productions are learned from studying the history of application of existing productions. Thus, in a basic sense, ACT's theory of procedural learning is one of learning by doing.

By itself, this general framework does not constitute a theory. A predictive theory must specify the following matters:

1. The representational properties of the knowledge structures that reside in working memory and their functional consequences.
2. The nature of the storage process.
3. The nature of the retrieval process.
4. The nature of production application, which breaks down to:
   a. The mechanism of pattern matching.
   b. The process that deposits the results of production actions in working memory.
   c. The learning mechanisms by which production application affects production memory.

ACT* differs from ACTE in its instantiation of each of these points except 2 and 4b, where the two theories basically agree.

The Assumptions of ACT*

Table 1.2 lists the fourteen basic assumptions of ACT* (what is listed as assumption 7 is not an additional assumption), and

Table 1.3 gives, for comparison, the twelve assumptions of ACTE. These lists include both the general architectural assumptions on which the theories agree and the more specific assumptions on which they tend to disagree. Details still need to be worked out about how these assumptions apply in various situations, and this is done in part in later chapters. Here I will go through the assumptions one by one, explain the meaning of each, and indicate some of their motivations.

The first assumption, 0, is a technical one. In ACTE, theoretical development was close to its computer simulation implementation, in which we were forced to generate the behavior in discrete time steps. Therefore, all the theoretical development was done in terms of discrete time intervals, although it does seem implausible that the many parallel processes in human cognition should march forward lockstep in such intervals. This was made part of the ACTE system only as an approximation. ACT* is more removed from any simulation embodiment, and we were motivated to work toward a theory in which time was continuous. All the basic assumptions in ACT*, then, will be cast in terms of continuous time.

The basic architectural assumption. Fundamental to all ACT theories has been the distinction between declarative and procedural knowledge, and this is a major difference between ACT and the neoclassical system. ACT has a number of advantages over the neoclassical architecture because of the decoupling of declarative memory from procedural memory. First, in conflict resolution the process of retrieving data from declarative memory does not have to compete with the productions that perform the task. The example production discussed earlier illustrates how the neoclassical system uses productions to retrieve declarative information. Such memory-retrieval productions must compete with those that perform the task, and the competition is aggravated because conflict resolution in the neoclassical system tended to allow only a single production to apply. The problem would not be as severe if the amount of task-relevant information were small, but the results on associative priming (see Chapter 3) have shown us that the amount of information brought into working memory, at least temporarily, is very large. Thus far there has been no attempt in the neoclassical system to integrate this broad-band, diffuse associative retrieval with the productions required to perform a task. I doubt that it can be done in a way that is consistent with the temporal properties of human information processing.

Another difficulty in the neoclassical approach concerns the