Production Systems and the ACT-R Theory

John R. Anderson
Carnegie Mellon University

1.1 INTRODUCTION

The work of most cognitive psychologists is driven by the same basic question: What is happening in the human head to produce human cognition? A great frustration of our field is that as we begin to search for an answer to what seems to be a straightforward question, we discover (a) that we may not be able to find an answer to the question, (b) that we are not sure what would constitute an answer, and, indeed, (c) that we are not even sure of what the question means. The goal of this book is to describe part of the answer to that question. Given the general uncertainty of our field, however, we must first define an interpretation of that question and specify what would constitute an answer to it. These are the primary goals of this first chapter.

To avoid suspense, however, I offer here the partial answer that this book offers: Cognitive skills are realized by production rules. This is one of the most astounding and important discoveries in psychology and may provide a basis on which to come to a general understanding of human cognition. I suspect, however, that most readers must be wondering what this statement really amounts to. What does it mean to say, “Cognitive skills are realized by production rules”? To help define this statement and place it in perspective this first chapter contains a brief discussion of foundational issues. Section 1.2 specifies what the theoretical status of the production system hypothesis is. Then, Section 1.3 identifies the basic features of the production rule theories of thought. Finally, Section 1.4 discusses the identifiability problems that haunt such accounts and how they are dealt with in the current approach. Each of these sections has as its goal placing the current work in proper relation to the relevant
issues in cognitive science. The sections can be brief because they refer to fuller expositions of the issues elsewhere.

1.2 FRAMEWORKS, THEORIES, AND MODELS

1.2.1 Levels of Specification

Cognitive psychology (and, indeed, psychology more generally) has had an almost fatal attraction to bold, general claims about human cognition (and human nature more generally). Here are a few examples:

1. There are two memory stores: a short-term store and a long-term store.
2. Knowledge is represented in terms of visual images and words.
3. People solve problems by means-ends analysis.
4. Syntactic knowledge and general world knowledge are encapsulated in different modules.
5. Human information processing is achieved by connectionist networks of neural-like elements.
6. Cognitive skills are realized by production rules.

Each of these assertions fails the most fundamental requirement of a scientific theory: empirical falsifiability. There are ways of construing each assertion such that it could be consistent with any empirical result. For instance, almost any form of the retention function could be made compatible with the distinction between long- and short-term memory by suitable auxiliary assumptions. Yet, these assertions are transparently not without meaning and, indeed, can be elaborated into predictions that are empirically falsifiable. To understand what is going on here requires reviewing the distinctions among frameworks, theories, and models (Anderson, 1983a).

Frameworks are composed of the bold, general claims about cognition. They are sets of constructs that define the important aspects of cognition. The distinction between long- and short-term memory, for example, would be a framework. Frameworks, however, are insufficiently specified to enable predictions to be derived from them, but they can be elaborated, by the addition of assumptions, to make them into theories, and it is these theories that can generate predictions. A single framework can be elaborated into many different theories. Certainly, many theories have been built around the distinction between long- and short-term memory; Atkinson and Shiffrin's (1968) theory is, perhaps, the most famous. The details that one must specify in going from a framework to a theory may seem unimportant relative to the ideas that define the framework, but they are absolutely essential to creating a true theory. For instance, it may not seem very important to the concept of short-term memory to assume it is a buffer with a fixed number of slots, but this was essential to the predictive structure of the Atkinson and Shiffrin theory.

Even a precise theory like Atkinson and Shiffrin's, however, is not enough to make precise predictions about a specific situation, such as a particular free recall experiment. One must make additional auxiliary assumptions to define how the theory applies to that situation. For example, within Atkinson and Shiffrin's theory, different rehearsal strategies could be assumed. The theory, with assumptions about its application to a specific situation, defines a model for that situation. There are many models possible within a theory, each corresponding to one way a subject could approach the situation. It is a specific model that one actually tests, although sometimes one could argue that no model derivable from the theory would be consistent with the results. It has, for example, been argued that no version of the Atkinson-Shiffrin theory could produce effects associated with depth of processing (Craik & Lockhart, 1972).

Production rules constitute a particular framework for understanding human cognition, and by now many theories have been proposed as instantiations of that framework. In 1983, I proposed a particular theory called ACT*; here I propose a variant called ACT-R. The details that define a specific production-rule theory, though perhaps insignificant compared to the features that are common to defining production rules in general, are essential if a claim that "cognitive skills are realized by production rules" is to be empirically falsifiable.

1.2.2 Cognitive Architectures

Production systems are particularly grand theories of human cognition because they are cognitive architectures. Cognitive architectures are relatively complete proposals about the structure of human cognition. In this regard, they contrast with theories, which address only an aspect of cognition, such as those involving the distinction between long- and short-term memory. Production systems are not unique as cognitive architectures. Popular, more recent alternatives are the various connectionist theories. To go back to an earlier era, Hullian theory (Hull, 1952) would constitute a cognitive architecture, although the adjective cognitive might seem a little misplaced.

The term cognitive architecture was brought into psychology by Newell, from his work on computer architectures (Bell & Newell, 1971). Just as an architect

---

1Deviating slightly from standard APA form, Anderson (without initials) refers throughout to J. R. Anderson.

2The reader will note this is a step in the direction of parsimony. ACT* was pronounced act star. The current theory is pronounced act ar, deleting the consonant cluster st.
tries to provide a complete specification of a house (for a builder), so a computer or cognitive architecture tries to provide a complete specification of a system. There is a certain abstraction in the architect’s specification, however, which leaves the concrete realization to the builder. So, too, there is an abstraction in a cognitive or computer architecture: One does not specify the exact neurons in a cognitive architecture, and one does not specify the exact computing elements in a computer architecture. This abstraction even holds for connectionist models that claim to be “neurally inspired.” Their elements are in no way to be confused with real neurons.

The major assertion of this book—“Cognitive skills are realized by production rules”—is a general assertion about the architecture of human cognition. It is limited in its scope only insofar as cognitive skill does not encompass all of cognition. This book illustrates some of the scope of “cognitive skill.” Along the way to making precise this general assertion, I define many more detailed assertions and present evidence for them.

A missing ingredient in the discussion so far is a specification of what constitutes a production system. The next section describes the concepts that define the production-system framework. The subsequent section addresses a fundamental indeterminacy that haunts such theoretical proposals. Although this indeterminacy is a problem for all cognitive architectures, this chapter focuses on its manifestation with respect to production systems.

1.3 PRODUCTION-SYSTEM ARCHITECTURE

The basic claim of the ACT-R theory is that a cognitive skill is composed of production rules. The best way to understand what this might mean is to consider a production-system model for a common skill, such as multi-column addition.

1.3.1 An Example Production System for Addition

Production rules are if-then or condition-action pairs. The if, or condition, part specifies the circumstance under which the rule will apply. The then, or action, part of the rule specifies what to do in that circumstance. Table 1.1 lists a set of five production rules that are sufficient to perform a certain amount of multi-column addition. These production rules are informally stated. The next chapter deals with the issue of how to formally specify these rules and with the sticky issue of what we claim is in the human head when we propose such a set of production rules. For now it is sufficient just to get a sense of how these production rules work. These production rules operate on addition problems such as:

\[
\begin{align*}
264 \\
+ 716
\end{align*}
\]

The production rules are organized around a set of goals. One goal is always active at any point in time. The first production rule, NEXT-COLUMN, focuses attention on the rightmost unprocessed column and will start by choosing the ones column.

The next production to apply is PROCESS-COLUMN. It responds to the goal of adding the column digits, but there are other elements in its condition. The second clause, “\(d1\) and \(d2\) are in that column,” retrieves the digits. Its third clause, “\(d3\) is the sum of \(d1\) and \(d2\),” matches the sum of those digits. In its action, it sets the subgoal of writing out \(d3\). The clauses in the condition of a production respond to elements that are said to be in working memory. Working memory refers to the knowledge that the system is currently attending to.

<table>
<thead>
<tr>
<th>TABLE 1.1</th>
<th>Production Rules for Addition*</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEXT-COLUMN</td>
<td>IF the goal is to solve an addition problem and (c1) is the rightmost column without an answer digit THEN set a subgoal to write out an answer in (c1)</td>
</tr>
<tr>
<td>PROCESS-COLUMN</td>
<td>IF the goal is to write out an answer in (c1) and (d1) and (d2) are the digits in that column and (d3) is the sum of (d1) and (d2) THEN set a subgoal to write out (d3) in (c1)</td>
</tr>
<tr>
<td>WRITE-ANSWER-CARRY</td>
<td>IF the goal is to write out (d1) in (c1) and there is an unprocessed carry in (c1) and (d2) is the number after (d1) THEN change the goal to write out (d2) and mark the carry as processed</td>
</tr>
<tr>
<td>WRITE-ANSWER-LESS-THAN-TEN</td>
<td>IF the goal is to write out (d1) in (c1) and there is no unprocessed carry in (c1) and (d1) is less than 10 THEN write out (d1) and the goal is satisfied</td>
</tr>
<tr>
<td>WRITE-ANSWER-GREATER-THAN-NINE</td>
<td>IF the goal is to write out (d1) in (c1) and there is no unprocessed carry in (c1) and (d1) is 10 or greater and (d2) is the ones digit of (d1) THEN write out (d2) and note a carry in the next column and the goal is satisfied</td>
</tr>
</tbody>
</table>

*\(c1, d1, d2, \) and \(d3\) denote variables that can take on different values for different instantiations of each production.
As I demonstrate in this book, we can understand and tutor skills like multi-column addition by assuming that production rules like these are the embodiment of the skill. One sees compelling evidence for production models like the one in Table 1.1 by observing a child acquiring the skill of addition using just these sorts of rules. I discuss tutoring research that displays this power of production-rule models throughout the book.

### 1.3.2 Critical Features of a Production System

You should now have a sense of how production rules function. It is worth emphasizing their critical features:

- Each production rule is thought of as a modular piece of knowledge in that it represents a well-defined step of cognition.
- Complex cognitive processes are achieved by stringing together a sequence of such rules by appropriate setting of goals and other writing to working memory, and by reading from working memory.
- Essential to production rules are their condition-action asymmetry, which as seen in later chapters, is reflected in many asymmetries of human behavior.
- A final important feature of production rules is that they are abstract and can apply in multiple situations. Thus, the rules are not specific to adding the digits 4 and 6, for instance, but can apply to any pair of digits. This generality is achieved by the use of variables in actual production-system formalism. In Table 1.1 this variable use is conveyed through terms like d1, but as shown in the next chapter, the informal specification in Table 1.1 underrepresents the variable use needed to get the correct generality for the rules.

There are a number of terms used to describe production system operation: **Pattern matching** refers to the process of determining if a production's conditions match the contents of working memory. Because multiple productions may match working memory, there arises the issue of deciding which of these will be performed. **Conflict resolution** is the term used to describe the process of determining which production rules to perform. When a production rule is performed it is said to execute or fire. The sequence of matching production rules, performing conflict resolution, and then firing a production is referred to as a **cycle**.

Corresponding to a production system is usually a computer program that actually simulates the behavior described by the production system. Writing a production-system model for a particular task usually takes the form of writing a set of production rules to perform the task. Indeed, production systems are often used as programming formalisms by people working in artificial intelligence who have no particular interest in cognitive modeling. Their status as programming languages has meant that production-system theories are precise and complete theories of particular tasks. This is a considerable virtue.
One problem with production-system theories has been that it is difficult to come to a deep understanding of a model without access to the actual running simulation, and access to other people’s simulations has been hampered by a lack of access to appropriate machines and languages. This barrier has been substantially eliminated by advances in modern technology. This book comes with a disk that includes the ACT-R system and a number of the simulations described here.

1.3.3 Alternative Production Systems

Over the years, multiple production systems that instantiate the general framework have been proposed. An informative overview of these production systems can be found in Klahr, Langley, and Neches (1987). Production systems can be traced back at least to Post’s (1943) proposal for rewrite systems. They also constituted an important formalism in Newell and Simon’s work, which culminated with the publication of their 1972 book, Human Problem Solving. Their early work involved production systems as a theoretical language, without a corresponding running program. The first production system that was implemented as a computer program was one called PSLG, used by Newell as the basis for his original papers on production-system models of mind (1972, 1973). Figure 1.1 (taken from Klahr et al., 1987) shows the lineage of production systems derived from this first implemented system. PSLG was the inspiration for the ACTEproduction system, which was the basis for the cognitive theory proposed in Anderson (1976). Over the next 7 years, this evolved and matured into the ACT* production system reported in Anderson (1983a). The ACT* production system was never completely implemented as a running computer system. GRAPES (Sauers & Farrell, 1982), shown in the figure, and PUPS (Anderson & Thompson, 1989), not shown, were partial implementations of the theory relevant to the acquisition of cognitive skills. One of the advantages of this book is the computer simulation that more completely corresponds to the theoretical statements in the book.

Other lines of production systems have evolved from PSLG. Particularly significant among these are the OPS production systems, which evolved out of a concern for how to do pattern matching and conflict resolution more efficiently. OPSS (Forgy, 1981) and OPS83 (Forgy, 1984) have served as the basis for development of some expert systems in artificial intelligence. The most well known of these expert systems is R1 (McDermott, 1982), which configures computer systems. Laird, Newell, and Rosenbloom (1987) produced a dramatic new system based in OPS called Soar, and Newell (1991) advanced Soar as a unified theory of cognition. A number of comparisons to Soar appear throughout this book.

Anderson (1983a) referred to the PSLG and OPS systems as neoclassical production systems to contrast them with the ACT production systems. Soar certainly differs from these earlier systems in many ways and is much closer to ACT* and ACT-R, but it does preserve one important feature of the earlier systems that contrasts it with the ACT theories. This is that it represents permanent knowledge only in production rules, whereas the ACT theories propose a separate declarative representation. The next chapter discusses the significance of this procedural-declarative distinction.

1.3.4 Evidence for Production Rules

It is worth describing at the outset the general kind of evidence that indicates production rules are psychologically real, although more detailed evidence appears in later chapters. One line of evidence is simply the intuitive reasonableness of a rule set like that in Table 1.1 for describing the cognitive processes involved in a task like addition. The descriptive adequacy of a rule-based account has become apparent to most researchers in cognitive science—even those connectionists who oppose the symbol-manipulation paradigm. Thus, J. A. Anderson and Hinton (1981) acknowledged that “well-learned and regular interactions between patterns of activity can be captured as explicit rules governing...
1. IDENTIFIABILITY

Having explained what kind of theory ACT-R is, I come to the thorny issue of how we can know it is the correct theory. The answer might seem simple: The theory is correct if it corresponds to the available data. However, there are serious problems in using behavioral data to identify the correct theory of mind. The next subsections will explain these identifiability problems and the approach to them taken in ACT-R.

First, though, it is important to point out a tacit assumption made in the subsequent subsections. This is that the production-system framework is the right way to think about cognitive skill. The question addressed in the subsequent subsections is how we know that there are production rules in the head, but rather, how we know ACT-R is the right production-system theory. This might well seem like we are focusing on the wrong question, but one cannot really argue for a framework because it is too poorly specified. The evidence for a framework always comes down to the success of the best theory specified within it. Thus, we have to be concerned with the details of the theory and how we know they are right. If we can get the details right, the framework will be established.

1.4.1 The Problems at the Implementation Level

For many theories, it is possible to make a distinction between an algorithm level and an implementation level (Anderson, 1987a, 1990c). That distinction is particularly well defined in the case of production systems. The algorithm level refers to a description of cognition in terms of the general steps of cognition. In the case of production systems, it is a description in terms of the production rules that are firing. The implementation level refers to a lower level description of cognition in terms of the factors that determine whether a specific production rule will fire and the speed with which it fires. The distinction is like the distinction between a high-level programming language like LISP and its machine-level implementation. Indeed, as shown in chapter 12, one can treat a production system as a programming language and simply ignore implementation issues. If we want a production system to be able to make psychological claims, however, we must be concerned with both the algorithm level and the implementation level.

As stated earlier, the details really matter when we make the claims that cognitive skills have a production-system base. Different production-system theories can differ in their details, at both the algorithm level and the implementation level. Distinguishing between different theories in terms of which production rules are actually at work (the algorithm level) is relatively unproblematic (as long as we do not get into empty debates about representation—see chapter 2). This is because the rules that are firing have such a close connection to the observed behavior. For instance, one could claim, at the algorithm level, that there was a carrying production that augmented the lower digit in an addition problem. This would be a different production system than the one in Table 1.1. It would be confirmed or disconfirmed by the observable behavior of a subject, that is, by whether the lower digit was actually augmented or not.

In contrast, there are profound difficulties in identifying what is going on at the implementation level. These difficulties arise because the theoretical claims about the implementation level are very detailed relative to the empirical data that can be used to judge the theories. These identifiability problems manifest themselves in two different ways:

1. Uniqueness. Very different proposals about what is taking place can result in the same claims about the probability and speed of production firings. These identifiability problems are rampant. For example, one might have one theory that claimed that all the production rules in Table 1.1 were matched in parallel
1. PRODUCTION SYSTEMS AND THE ACT-R THEORY

and another theory that claimed they were matched serially. In general, however, serial and parallel information-processing systems can be shown to be equivalent in their predictions about behavioral measures, such as processing time (Anderson, 1976; Townsend, 1974; Voelber, 1977). Thus, behavioral data cannot distinguish between parallel and serial production matching. This is just one of the many ways that we face the fact that black boxes with very different internal structures can display identical external behavior in all respects.

2. Discovery. There is a huge space of possible implementation proposals and little guidance in finding the correct one. It sometimes seems impossible to discover one that is consistent with the data. It is like trying to find a needle in a haystack. The basic problem is that there are limitless ways to imagine the internal structure of a black box. Certainly, we have seen numerous proposals for production systems, and each has proven unsatisfactory in some regard.

One can question whether these problems really exist, and if they do whether they are peculiar to production-rule modeling. I have argued at length elsewhere (Anderson, 1976, 1987a, 1990c) that these are problems for all cognitive theorizing—not just for production systems—and it seems unnecessary to repeat the arguments. The news offered here is a way to approach these problems.

One can imagine the space of all possible cognitive theories, although infinite, as distinguishable into three ordered sets. First, there is the set of all theories. A subset of that is the set of theories consistent with the behavioral data collected so far. This subset is a tiny fraction of all theories. A much smaller subset of this subset is the set of all theories consistent with all behavioral data that could be collected. The discovery problem is that this final subset is such a tiny part of the set of theories. The uniqueness problem is that there is more than one theory in this final subset.

It needs to be emphasized that these are two independent problems. Even if we could recognize the right theory when we found it (i.e., solve the uniqueness problem), we would still have the problem of finding it. Even if we could find a theory consistent with the data (i.e., solve the discovery problem), we would face the fact that there are many equivalent theories. Thus, there are two separate problems, and they require two separate solutions. Although I cannot claim to have solved either problem completely, ACT-R reflects an approach to each problem that offers the hope of eventual solutions.

1.4.2 The Neural Approach to the Uniqueness Problem

The solution to the uniqueness problem that I have adopted is to commit to a particular style of implementation. Because cognition must be implemented in the human brain, it seems transparent that the implementation of ACT-R should be in terms of neural-like computations. Thus, the constraint used to choose among behaviorally equivalent proposals is that the mechanisms proposed correspond to what is known about neural processing. For instance, with respect to the parallel-serial issue, we know that neural computation is highly parallel. This tells us that many processes, such as memory retrieval, have to be parallel, including the matching of production rules.

The style of neural implementation assumed in ACT* and continued in ACT-R is activation based. Declarative memory structures vary in their level of activation, and this determines their availability. Also, the rate of production-rule matching is determined by the activation levels of the declarative structures the rules match. Rules compete by inhibitory processes. A major component of learning is increasing the strength of declarative structures and production rules. Thus, when we dig below the surface of an ACT theory we find a system of computation that looks much like a connectionist system. However, in ACT, these connectionist computations at the implementation level are being used to support a system that is symbolic at the algorithm level. The computer analogy is that the primitive machine operations support the symbolic processing of LISP.

We have only partially acted on our commitment to a neural-style implementation of production systems. The activation-based computations described in subsequent chapters are only a gloss of the computations a true connectionist would want to see specified in further detail. In the last chapter, I discuss some tentative ideas about further layers of elaboration.

Note, too, that the commitment to a neural-style implementation of a production system is no guarantee of a solution to the uniqueness problem. There may well be equivalent implementations consistent with what is known about neural processing. In such cases, we have to wait for more knowledge about what neural processing is like.

1.4.3 The Rational Approach to the Discovery Problem

The problem I had with the ACT* theory and other theories (even if they were neurally based) was that they do not solve the discovery problems: finding an implementation consistent with present and future data. There are an enormous number of ways to implement production systems in a neural-like system. Why believe one is more correct than another? One can try to find a theory consistent with the available data, but whatever reason is there to believe it will be consistent with the next empirical phenomenon? This is exactly what happened with ACT* (Anderson, 1983a). No sooner had the theory been published than results came forth that seemed inconsistent with it. Such an occurrence is hardly unique to ACT*. Of course, one can always revise the theory slightly, to make it consistent. This is what happened in Anderson (1987b), and the same thing has happened with other theories.

An infinite number of theories that are consistent with any finite body of data will make different predictions about data yet to be collected. We need some
1. PRODUCTION SYSTEMS AND THE ACT-R THEORY

reason to believe that when we commit to one of these theories it will hold up
when the new data come in. As argued elsewhere (Anderson, 1983a, 1990a),
certain factors, such as parsimony, which help us choose among theories in some
sciences, do not work that well in cognitive science. Biology in general and cog-
nition in particular do not select systems on the basis of their parsimony.

It was this discovery problem that led to the rational analysis that was
described in Anderson (1990a). In line with the arguments of Marr (1982), ra-
tional analysis seeks to provide some guidance in proposing the implementa-
tion details. Rational analysis was an attempt to understand cognition, based on
the thesis that cognition is adapted to the structure of the environment. Anderson
(1990a) was an attempt to explore this thesis with respect to the cognitive func-
tions of memory, categorization, causal inference, and problem solving. In each
case, it was argued that cognition seemed to maximize achievement of
information-processing goals within the constraint of minimizing computa-
tional costs. The effort in the 1990 book was an attempt to show optimization with
minimal commitment to mechanism, but rational analysis can be used to con-
strain the mechanisms that implement the production-system architecture. This
is what happened with respect to ACT and has resulted in the new theory, ACT-
R (with the R for rational). The mechanisms in ACT* were tuned and slight-
ly changed in ACT-R to yield adaptive processing under the 1990 rational analysis.

1.5 THE REST OF THE BOOK

The theory to be proposed in this book comes from the intersection of four con-
straints:

1. That it be consistent with the wide variety of data deemed relevant.
2. That it be expressed as a production-system architecture, which seems
to capture many salient features of the performance of cognitive skill.
3. That it be implemented in terms of neural-like processes, so that it is some-
thing that might inhabit a human brain.
4. That these processes be configured to yield optimal behavior (given the
statistical structure of the environment), so that we can have additional
reason to believe in their correctness.

The constraints just listed are a consequence of the concern with identifiabil-
ity issues. In addition to these constraints, it is worth acknowledging two other
factors that influenced the shape of the theory. The first is the legacy of ACT*;

although the theory is not exactly ACT*, its structure bears considerable resem-
bance to that of ACT*. The second is the commitment to providing a runnable
and usable system on the disk that accompanies this book. The need to have
all of these claims embodied in one system forced a high degree of clarity and
consistency in the development of the theory. I am hopeful that the accompanying
simulation will do much to promote scientific communication.

As indicated earlier, there is a tension between grand claims at the frame-
work level and the details that give reality to these claims at the theory level.
This book shows this tension. Much of the book presents evidence that sup-
ports the general production-rule framework and discusses how that framework
can be used, but the book also specifies the details of the ACT-R theory and
the evidence for those details.

The next three chapters (2 through 4) are devoted to a detailed discussion of
the theoretical assumptions of the ACT-R theory. Chapters 5 and 6 are con-
cerned with research that supports critical aspects of the ACT-R conception
of skilled performance. Chapters 7 through 10 report studies concerned with
the acquisition of cognitive skill. The final chapters touch on wider issues rel-
ed to production systems with production rules: Chapter 11 considers issues
surrounding the use of intelligent tutors based on production systems; chapter
12 discusses how to build productions in ACT-R; and the concluding chapter,
13, reviews the assumptions of the theory and the prospects for the theory's
future development.

When I wrote the 1990 book, I was not sure what the relationship between rational analysis
and mechanistic theory was, although I did speculate that rational analysis could be used to guide
mechanistic theory.
2

Knowledge Representation

John R. Anderson
Carnegie Mellon University

2.1 KNOWLEDGE REPRESENTATION IN ACT-R

Knowledge representation is one of the thorniest issues in cognitive science. If we are to have a theory in which mental objects undergo transformations, we need to have some notation to represent these objects. The difficulty is determining what it is about a representation that amounts to a substantive theoretical claim, and what is just notation. This is a particularly grievous problem with production systems, because they take the form of a complete programming language. As a consequence, one can write systems that solve complex problems, going all the way from initial input to final output, leaving nothing that is unspecifed or performed by an unanalyzed system. Programming languages, however, bring with them considerable theoretically irrelevant trappings, and it is easy to come to believe that these syntactic trappings amount to significant psychological claims.1

It is important to clarify which claims in the ACT-R representation are significant, and which are just notation. There are three essential theoretical commitments one makes in the ACT-R knowledge representation. One is that there are two long-term repositories of knowledge: a declarative memory and a procedural memory. The second is that the chunk is the basic unit of knowledge in declarative memory. The third is that the production is the basic unit of knowledge in procedural memory. These issues are covered in the three sections that follow.

1The problems of properly evaluating the claims of simulation models, which are acute in the case of production systems, are by no means unique to them (Prijda, 1967; Kieras, 1985; McCloskey, 1981; Neches, 1982; Ohlsson, 1988; Schneider, 1988; S. R. Young, 1985).
2.2 THE DECLARATIVE–PROCEDURAL DISTINCTION

The most fundamental distinction in ACT is the distinction between declarative and procedural knowledge. This distinction goes back to the original ACT system (Anderson, 1976), and has remained throughout all the modifications. At the time of its proposal, this distinction was viewed as something of a radical suggestion,\(^2\) one which many researchers viewed as having just been discredited in artificial intelligence (e.g., Winograd, 1975). Since 1976, however, a good deal of empirical evidence has accumulated for such a distinction in the human mind, as I review here.

Intuitively, declarative knowledge is factual knowledge that people can report or describe, whereas procedural knowledge is knowledge people can only manifest in their performance. A good example of declarative knowledge would be our knowledge that Washington, DC is the capital of the United States; a good example of procedural knowledge would be our ability to speak English.

The same abstract knowledge can have both procedural and declarative embodiments. Thus, declaratively we might have memorized the layout of the typewriter keyboard, and procedurally we may know the keyboard as part of our typing skill. Many people have lost declarative knowledge of the keyboard while remaining excellent typists. The only way they can tell where a key is on the keyboard is to imagine themselves typing the letter and seeing where their finger goes. This is an example in which the declarative knowledge seems to have atrophied. The two sources of knowledge are not mutually exclusive, however. People can and do maintain both declarative and procedural representations of the same knowledge.\(^3\)

As noted earlier, one thing that distinguishes the ACT line of theories from the neo-classical theories and Soar is that ACT has long-term representations of both declarative and procedural knowledge, whereas these other systems have only a procedural long-term memory. The primary motivation for proposing a separate declarative memory in 1976 was to explain many of the phenomena of human memory (for a discussion see Anderson, 1983a, pp. 13–17). In particular, it is awkward, at best, to model with production rules much of the verbal learning research that is best conceived of as declarative learning. Recently, there have been some substantial efforts to model declarative learning in Soar with the mechanism of data chunking (Newell, 1991). These efforts are notable because they succeed at a difficult task, but not because of any insights they bring to the nature of declarative learning. There are numerous phenomena of verbal learning, such as associative priming, fan effects, depth of processing, configural recall, and so on (Anderson, 1985a), that data chunking does not address and that have guided the ACT theory of declarative memory. It would probably be possible to get data chunking to emulate the ACT theory, but this would amount to creating a special type of production memory that emulates declarative memory. Thus, traditionally one can insist on representing declarative and procedural memory equivalently, but at the cost of clarity. The fundamental psychological claim is not that the two systems have different notations, but rather that they entail different memory processes.

2.2.1 Defining the Procedural–Declarative Distinction

Before progress can be made in discussing the declarative–procedural distinction, it is necessary to have a precise definition of the distinction. The natural tendency of experimental psychologists is to look for an operational definition, a set of behavioral tests that will allow them to know whether they are dealing with declarative knowledge or procedural knowledge. The most common operational definition involves verbalization. Knowledge that one is able to verbally describe or declar is considered declarative, whereas knowledge that can only be inferred from an individual's behavior is considered procedural. Although such a definition has advantages for the purposes of design and analysis of particular experiments, it does not serve very well as a general theoretical definition. There are instances of declarative knowledge that cannot be verbally communicated. For example, it is sometimes difficult to describe verbally knowledge that one might want to consider declarative, such as the shape of an object.\(^4\) It also seems unreasonable to propose that nonverbal creatures do not have declarative knowledge.

The only satisfactory way to define the procedural–declarative distinction is in terms of a theoretical framework. The distinction will be defined here within a production-system framework. The distinction between declarative and procedural knowledge turns on the fundamental mode of operation of a production system. Productions function by reading information from working memory and writing information to working memory. The information in working memory is declarative knowledge, and the information in the productions is procedural knowledge. This distinction is much like that between program and data.

It follows from this relationship between procedural and declarative knowledge that declarative knowledge will tend to be describable and procedural knowl-

---

\(^2\)Perhaps calling it a reactionary suggestion would be more accurate. The idea had been around for some time, having received a notable statement by the philosopher Ryle (1949). By 1976, it had largely been discounted as a false distinction.

\(^3\)Indeed, as I show further on, we can maintain multiple, different procedural representations of the same knowledge.

\(^4\)One can be trained to describe pictorial information that one could not previously describe. It would make no sense to claim that acquisition of further verbal reporting skill causes a whole class of knowledge to convert its status to declarative.
2. KNOWLEDGE REPRESENTATION

edge will not. To describe knowledge, some knowledge-reporting productions would have to apply that would inspect the knowledge and report it. The only knowledge that productions can inspect is the knowledge in working memory. They cannot inspect the knowledge contained in other productions. Although it would be possible to create production systems with some capacity to inspect productions, I know of only one production system, Ohlsson (1973), that has ever been created with this capacity.

It is also important to observe that all production systems cast this way, including Soar, have declarative knowledge as well as procedural knowledge. The issue that distinguishes the ACT theories from the neoclassical and Soar theories concerns the long-term status of the knowledge in working memory. In all production-system theories, there are some limitations on the capacity of working memory. In the neoclassical theories and Soar, once knowledge leaves working memory it is permanently lost, whereas in ACT the knowledge remains but is inactive. It can be reactivated for later use by a spreading activation process. The limitation on working memory capacity in ACT concerns access to declarative knowledge, not the capacity of declarative knowledge. Declarative memory contains a complete record of the past, from the last few seconds to years ago. The most recent records still tend to be active, but the older records can be made active when needed.

So, the declarative–procedural distinction is built into the concept of a production system, and the only issue is the long-term status of the declarative contents of working memory. Other frameworks fundamentally blur the declarative–procedural distinction. It is not even part of most connectionist systems. Such systems are committed to a specific use of knowledge and have difficulty with deploying knowledge in novel ways. A typical connection net, coded to do addition in base 10, would not reflect on its knowledge and redeploy it to do addition in base 6.

Why should a system have both declarative and procedural knowledge? Two types of knowledge are required because one needs different knowledge structures for flexible use of knowledge than for efficient use of knowledge. Declarative representational capacity allows the system to acquire knowledge rapidly in a flexible form that is not committed to a particular use. A procedural representational capacity allows the system the ability to optimize the application of that knowledge for a specific use. It is this need for both flexibility and efficiency which motivates the declarative–procedural distinction.

2.2 THE DECLARATIVE-PROCEDURAL DISTINCTION

2.2.2 Experimental Evidence

for Two Long-Term Memories

If one wants to retrieve what is thought of as declarative knowledge in a system without a separate long-term declarative memory, it is necessary to create productions that will perform this retrieval. Those productions would be of the form:

IF Washington, DC is mentioned
THEN note that it is the capital of the U.S.A.

In contrast, this knowledge is retrieved by a spreading activation process (described in detail in chapter 3) in a system like ACT, which has a separate declarative knowledge. In such a system, this knowledge would be activated by having activation spread from Washington, DC to the fact. The two proposals behave the same at one level: First, Washington is in working memory (or active), and as a consequence, “Washington is the capital of the U.S.A.” enters working memory (or becomes active). The claim of a system with only productions in long-term memory is that the same principles that govern the firing of productions govern this declarative retrieval, whereas the proposal for two separate long-term memories allows for different principles. The empirical question thus becomes whether procedural and declarative long-term memories behave the same.

A number of findings show that declarative and procedural long-term memories have different properties. The first three in the following list derive from the more flexible access to declarative knowledge. The last three derive from different acquisition and retention histories for the two types of information:

1. Reportability. As noted, procedural knowledge is not reportable, but declarative knowledge is potentially reportable. This issue of reportability of declarative knowledge is discussed further in chapter 12, in the discussion of verbal protocols.

2. Associative Priming. As discussed in Anderson (1983a), there is an associative priming process defined in declarative memory that has no correspondence in procedural memory. Thus, when one hears the word computer, there is priming for the word programming (e.g., one can read the word programming more rapidly), but not for one's computer programming skills (e.g., one cannot program more rapidly). Declarative priming is implemented in ACT-R through the spreading-activation mechanism.

3. Retrieval Asymmetry. As I discuss later in this chapter and throughout the book, there is an asymmetry of access to procedural knowledge that does not exist for declarative knowledge (e.g., Table 2.2 will show that learning about
LISP for purposes of coding does not generalize to using LISP for purposes of code evaluation).

4. Acquisition. As illustrated in chapter 4 and elsewhere, the origins of declarative and procedural knowledge are different. Declarative knowledge comes from direct encoding of the environment, whereas procedural knowledge must be compiled from declarative knowledge through practice. Thus, skills are often acquired by analogy to declarative representations of examples (see chapter 4).7

5. Retention. The retention functions for the two types of memories are independent. The most striking case of this is when people get better at using the procedural knowledge but worse at recalling the declarative knowledge. The typing example discussed earlier illustrates this.

6. Dissociation. There have been a number of recent demonstrations of dissociations of declarative and procedural memory in amnesic and other populations. They have received so much attention lately that I discuss them at length in the next two subsections.

This is not to say there are no features common to declarative and procedural memory. There should be, because they are both presumably implemented in neurons and should, therefore, reflect common properties of neural processing. As elaborated in chapter 4, for example, it seems that the functions describing the build-up and decay of strength are the same for the two types of memory.

 Parsimony has seductive attraction in cognitive science, and theorists are often drawn to the belief that by some representational maneuver they can get two classes of phenomena out of one set of principles. This goes back, at least, to Watson's (1930) attempts to argue that memory was just verbal habit, but these efforts have so far proven themselves to be sophism. Such systems neither run nor address the empirical phenomena.

2.2.3 Dissociation of Declarative and Procedural Memory

A recent surge of research supports the separation of declarative and procedural memory. Much of it has involved looking at amnesic who exhibit an inability to learn new information as a function of neural damage. The widely studied patient, HM, is one such subject. HM had parts of his temporal lobes removed in an attempt to treat epilepsy. He has one of the most profound amnesias ever documented and, for over 30 years, has been unable to consciously recall new events, although he remembers things from before his operation reasonably well.

7The ACT assumption that procedural knowledge originates in declarative knowledge has been criticized. I review these criticisms at the end of the next subsection.

In other words, he has lost the ability to acquire new declarative information. Nevertheless, he has been able to acquire new skills, such as a rotary pursuit task, even though, when questioned on later learning trials, he claims not to remember ever doing the task before (Corkin, 1968).

A group of patients with similar amnesic symptoms are Korsakoff patients, who have suffered extensive neural damage as a result of a long history of alcohol abuse and nutritional deficit. Cohen and Squire (1980) presented Korsakoff amnesics and normals with mirror images of words. Both groups improved equally rapidly at reading the words, but the Korsakoff patients showed substantial impairment in remembering the words they had read. Thus, they acquired the procedural skill of reading the words while failing to retain the declarative knowledge of having seen the words.

There have been numerous demonstrations of amnesics' learning of skills with strong perceptualmotor components, but relatively few demonstrations of their ability to learn skills that are more purely cognitive. Cohen, Eichenbaum, Deacedo, and Corkin (1985) reported teaching amnesics to perform in the Tower of Hanoi task, but others have failed to replicate this result (e.g., Butters, Wolfe, Martone, Granholm, & Cermak, 1985). Hirst, Phelps, Johnson, and Volpe (1988) claimed success at teaching an amnesic to speak French, and Milberg, Alexander, Charness, McGuinley-Bertho, and Barrett (1988) were able to teach two amnesics an arithmetic algorithm.

Normal individuals sometimes appear to acquire a skill without having conscious access to the knowledge required to perform the skill. For instance, Broadbent, Fitzgerald, and Broadbent (1986) demonstrated that normal subjects could learn to operate a model transportation system without being able to articulate the rules. Amnesics given the same task were also able to learn it (Phelps, 1989).

In reviewing the literature on cognitive skill learning, Phelps concluded that amnesics are able to learn skills in cases where the skill learning does not require recalling past events and integrating them. That is, if the skill can be learned using only current declarative information and without requiring long-term declarative memory, amnesic subjects can learn it. She found that amnesics were able to learn Broadbent et al.'s task only if they were given concurrent access to previous trials and did not have to remember them; normal subjects did not need such assisted access to previous trials.

The research of Nissen and her associates provides further evidence for the procedural–declarative distinction. Her paradigm involves having subjects learn to press a set of buttons according to a 10-trial repeating-stimulus sequence. She separately tracks procedural learning of the sequence and declarative learning of the sequence. Her measure of procedural learning is the increase in speed with which subjects can press the button that corresponds to the next item in the sequence. Her measure of declarative learning is the ability to describe the sequence or explicitly predict it. She has shown in a number of ways that procedural and declarative learning can be disassociated:
2. KNOWLEDGE REPRESENTATION

1. Normal subjects show both declarative and procedural learning, but the two are uncorrelated (Wellington, Nissen, & Bülmer, 1989).
2. Amnesic patients show only procedural learning (Knopman & Nissen, 1987; Wellington et al., 1989).
3. Administration of scopolamine to normals impairs declarative learning but not procedural learning (Nissen, Knopman, & Schacter, 1987).

Nissen interpreted her results as showing that acquisition of procedural knowledge does not depend on declarative knowledge, in contrast to the claims of ACT. What her results really show, however, is that acquisition of procedural knowledge does not depend on long-term declarative learning. Indeed, when she distracted subjects from forming a declarative representation of the sequence by using a dual-task procedure, they failed to show any procedural learning (Wellington et al., 1987). Thus, it seems that active (but not necessarily long-term) declarative representations are essential for procedural learning. ACT-R requires that declarative structures be active to support procedural learning; it does not require that they be capable of being retrieved at a delay.9

Broadbent (1989) has similarly argued that the results from the Berry and Broadbent (1984) task contradict ACT’s claim that procedural knowledge derives from declarative knowledge. They showed that subjects can learn to manipulate a rule-based system successfully but cannot consciously state the rules. It appears that, in this situation, subjects were basing their behavior on memory for specific examples and not on rules (Marescaux, Dejean, & Karnas, 1990). Similarly, the performance in the Reber task, which has long been cited as an example of implicit learning of rules, appears to originate in declarative memory for fragments of specific examples (Dulany, Carlson, & Dewey, 1984; Serveschreiber & Anderson, 1990). Thus, it appears that sometimes subjects are using declarative knowledge other than what the experimenter had in mind.

As one cautionary note, it is too simple to view all literature on amnesia as showing procedural long-term memory preserved. It might be more accurate to view much of it as declarative long-term storage lost. It appears that many other capacities are preserved besides procedural memory, including the priming of declarative memories established before the amnesia (Schacter, 1987).

2.2.4 Interpretation of Amnesia Results

Although the data from amnesics seem to support the procedural-declarative distinction of ACT-R, it is another question to determine how they are to be interpreted within the theory. One view would be that new declarative structures are lost rapidly after being formed, but this does not seem in keeping with

the ACT-R theory because its model for normal functioning does not involve loss of declarative structures. Rather, what declarative structures lose with time is strength. Most amnesics are not totally discontinuous with normals and can recall something on delayed memory tests. This could be produced in ACT-R either by having a weaker initial encoding process or a more rapid forgetting process. In this regard, it is interesting to note that amnesics can be made to display the same forgetting functions as normals if they are overtrained (Huppert & Piercy, 1978). Thus, perhaps the best interpretation of amnesia in ACT-R is in terms of weak initial memory traces. The concept of trace strength is developed in the next two chapters.

As a final remark, it is certainly the case that there are theoretical interpretations of these amnesias data not based on the procedural–declarative distinction. One of the more notable alternatives is the transfer-appropriate-processing explanation of Roediger and Blaxton (1987). They argued that what are classified as declarative tasks are better considered conceptually driven tasks, and what are classified as procedural tasks are better considered as data-driven tasks. Their argument would be that the ability to acquire conceptual knowledge is impaired, whereas the more perceptual knowledge can be maintained. A major problem with this proposal is the apparent ability of the amnesic subjects to acquire relatively complex problem-solving skills that seem to defy the description of data-driven. Unfortunately, the empirical record on these tasks is not as clear as it is on other tasks.

2.3 THE "CHUNK" IN DECLARATIVE MEMORY

Having determined that there are two types of long-term memory, we still must decide what their characteristics are. Declarative information is represented in terms of data structures which are basically the chunks that G. A. Miller described in 1956 and which were the focus of a great deal of research in psychology until the mid-1970s (for a review, see N. F. Johnson, 1970). New declarative knowledge is added to memory (and hence long-term memory) a chunk at a time. In the language of production systems, the units in which knowledge is represented in working memory are referred to as working-memory elements or WMEs (pronounced winnies). There are three significant features associated with a chunk, or WME, within the ACT-R theory. First, it appears that only a limited number of components can be combined in a single chunk. Second, chunks have configural properties such that different components have different roles, which would not be the case if these were just elements of a mathematical set. Third, chunks are capable of a hierarchical organization, such that

---

9Note that in many situations the requisite declarative structures are kept active in the environment.

10We will retain this name although it might be clearer to try to introduce a terminological change and call them declarative memory elements, or dimes.
chunks can appear as components of other chunks. Subsequent subsections
describe each of these properties of chunks; this is followed by some discussion
of what representational notation might be appropriate for them.

It is worth saying a little about terminology here. I am using chunk to refer
to essentially the same thing as what I called a cognitive unit in the book on
ACT\textsuperscript{+} (Anderson, 1983a). My change in terminology is just a reflection of my
realization that what was essential in the concept of a cognitive unit is identical
to what the term chunk had meant in psychology. Also, note that chunk as used
here is very different from chunking as it has been used in Soar. Chunking in
Soar refers to a form of procedural learning not unlike composition in ACT\textsuperscript{+}
(which is briefly described in chapter 4). A chunk in Miller's conception is a
declarative structure.

2.3.1 The Size of Chunks—Three Elements?

A chunk is a means of organizing a set of elements into a long-term memory
unit. Thus USA can be thought of as a chunk organizing three letters into a
single unit. It is generally thought that only so many elements can be organized
within a chunk. If more elements have to be combined, we need to have sub-
chunks within chunks. Thus, people naturally break longer sequences up into
sub-elements. For instance, telephone numbers are encoded as a prefix (the
"exchange") plus a number. If subjects are asked to repeat back a set of digits,
they will be observed to insert pauses indicating they have imposed a hierarchi-
cal structure on the set.

If there is a limit on the number of elements that can be organized within
a chunk, the question naturally arises as to what that limit is. Broadbent (1975)
argued that this number is three. He gathered together a variety of data, none
of which alone is all that convincing, but in combination it begins to create a
case. One of the things he did was observe the length of runs subjects would
give before inserting a pause in a sequence of elements. He found that 54%
of all pauses came after a pair of items, 29% after three, 9% after four, and
9% after longer sequences. This gives somewhat the appearance of a barrier
after three elements.

Broadbent also cited a number of memory studies. Ryan (1969), for exam-
ple, chunked a sequence of nine digits into three groups of various sizes for
her subjects. She found recall to be best when the subsequences were three
groups of three each. Similarly, Wickelgren (1964) found memory to be most
efficient when rehearsal groups were of size three: If three were the ideal chunk
size, using smaller rehearsal groups would mean more chunks would be required
to encode the sequence than necessary. Using rehearsal groups larger than three
would mean that the rehearsal groups would have to be broken down into sub-
chunks (that would not be of optimal size three).

2.3.2 The Configural Structure of Chunks

The second feature of chunks is that their elements assume specific relational
roles. This is most clearly demonstrated with strings of elements: If subjects
study a set of letters in one order, it takes longer to recognize them when they
are presented in a different order (e.g., Angiolillo, Bent, & Rips, 1982). The
structure of elements is not always linear, however: Many chunks are spatial
and have their elements encoded with respect to particular positions in space.
Also, it is critical to have what are variously called propositional, or semantic,
or relational representations, in which elements are encoded with respect to
their semantic relationships.\textsuperscript{11}

The experiment by Santa (1977) is a nice demonstration of the difference
between linear and spatial representations and of the importance of configura-
tion to the encoding of declarative information. Figure 2.1 illustrates his materi-
als. Subjects were shown spatial arrays of either three figures or three words.
They were then shown another array of elements and asked to judge if the same
elements were in that array. When subjects viewed an array of figures, they
were fastest when the test array presented the same figures in the same spa-
tial locations. When they were presented with words, they were fastest when
tested with a list of the elements in the left-to-right order that they would have
been read in the test array. Thus, in Fig. 2.1, the first test array is identified

\textsuperscript{10}Note, however, that Servan-Schreiber (1991) argued for binary chunks.

\textsuperscript{11}Anderson (1983a) referred to these three types of configural representations as temporal
string, spatial image, and abstract proposition.
in a hierarchy, but such studies do exist. Bower, Clark, Lesgold, and Winzenz (1969) were able to show a hierarchical organization of elements (e.g., minerals) in up to four levels. Klahr, Chase, and Lovelace (1983) found evidence for a similarly complex representation of the 26 letters in the alphabet. Chase and Ericsson (1982) argued that the 80-digit span of their memory experts also depended on a deeply nested representation. In all these cases, it is a bit difficult to get behavioral indicators to discriminate all the proposed levels, but it seems plausible that such representations are being used by the subjects.

**2.3 The "Chunk" in Declarative Memory**

**2.3.4 A Notation for Chunks**

Having settled on the significant properties of chunks, we must now concern ourselves with how to represent a chunk and its particular number of elements, configural properties, and possible positions in hierarchies. For some purposes, informal English descriptions might be adequate, but they will not be when precision is required for either theoretical statement or computer specification. In our own computer simulation work, my colleagues and I have gradually developed a "vanilla," schema-like representation like that used in many other AI applications. Table 2.1 illustrates what that representation would be like for the initial state of the addition problem discussed in chapter 1.12

The chunks in this representation are probem1, column0, column1, column2, and column3. Associated with each chunk are a number of slots. The first slot is the isa slot, which indicates what type of chunk it is. This identifies the configural properties of the chunk, because the type of the chunk determines what other slots there can be.13 The first chunk in Table 2.1 is of type numberarray, and the remainder are of type column. The remaining slots in the chunk contain pointers to other working-memory elements. The first chunk in Table 2.1 has pointers to the four columns, whereas each of the column chunks has pointers to the three rows. The four columns are encoded in a list to facilitate getting the next column by the production rule NEXT-COLUMN (why this is so becomes clear in the formal encoding of the production rule in Table 2.3). The example in Table 2.1 is basically a double-linear encoding of an array (columns encoded linearly and rows encoded linearly within columns).

This schema-like system can also be used to encode spatial and propositional information. So, the study array (top box) in Fig. 2.1a could be encoded as:

---

12 The full working-memory encoding, which includes a list of addition-table facts, is available in the Addition file in the accompanying Examples folder.

13 As we see in chapter 12, it is necessary to first declare what slots can appear with each chunk type.
picture1
  isa square
  upper-left triangle1
  upper-right circle1
  lower-middle square1

triangle1
  isa triangle
  vertical top-third
  horizontal left-third

circle1
  isa circle
  vertical top-third
  horizontal right-third

square1
  isa square
  vertical bottom-third
  horizontal middle-third

<table>
<thead>
<tr>
<th>TABLE 2.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema Representation of the Problem:</td>
</tr>
<tr>
<td>264</td>
</tr>
<tr>
<td>+ 716</td>
</tr>
</tbody>
</table>

| problem1 |
| isa numberarray |
| columns (column0 column1 column2 column3) |
| column0 |
| isa column |
| topraw blank |
| bottomrow + |
| answerrow blank |
| column1 |
| isa column |
| topraw two |
| bottomrow seven |
| answerrow blank |
| column2 |
| isa column |
| topraw six |
| bottomrow one |
| answerrow blank |
| column3 |
| isa column |
| topraw four |
| bottomrow six |
| answerrow blank |

2.4 THE PRODUCTION IN PROCEDURAL MEMORY

where the terms upper-left, upper-right, and lower-middle reflect two-dimensional coordinate information to whatever degree of accuracy the system is able to maintain.

This scheme can also be used to represent propositional information. For instance, the meaning of the sentence, "The quick brown fox jumped over the lazy dogs" could be represented as:

event1
  isa jumping
  agent fox1
  object dog1

fox1
  isa fox
  speed quick
  color brown

dog1
  isa dog
  character lazy
  number plural

This representation might need substantial work in a thorough propositional system, but it serves to indicate the basic character of how such a representation would proceed.

This representation can accommodate the three key features associated with chunks: (a) Configural information is enforced by the chunk type (isa value) and associated slots; (b) chunk size limitations can be enforced by limitations on the number of associated slots; and (c) chunk hierarchies can be achieved by having the values of chunk slots be chunks themselves.

2.4 THE PRODUCTION IN PROCEDURAL MEMORY

The production rule is an encoding of knowledge that is optimized for use. Such optimization involves achieving a tradeoff between range of applicability and efficiency of application. There are four significant features associated with productions in ACT-R: their modularity, their abstract character, their goal structuring, and their condition–action asymmetry. Each can be understood as an aspect of optimizing the utility of production rules. The subsections that follow expand on what is meant by each of these features and provide evidence for them. ACT-R is not distinguished among production systems in any of these features except for the goal factoring of the production system.\(^{14}\) Even in this feature,

\(^{14}\)It is, however, distinguished from non-production system architectures by these features.
most recent production systems do have a goal structure. After considering these features, I propose a formal notation for representing productions.

### 2.4.1 Production Modularity

*Production modularity* has often been advertised as meaning that each production is a separate element that can be added and deleted, independent of any other production. This is basically true, but one should not conclude from such a statement that production rules do not interact. Changing or deleting any production in Table 1.1 produces very different behavior from the set as a whole. Solving a problem depends on the interactions among the productions involved, and the overall performance is only as good as the weakest production involved in the solution. Part of the strength of rule-based models is that they can account for many types of erroneous behavior by deletion of and changes in production rules. For instance, a common error in children's multi-column subtraction is that they have a rule that tells them to subtract the smaller number from the larger number, whether or not it is in the top row (Brown & VanLehn, 1980). This can be represented by a different production rule than the correct one.\(^\text{15}\)

One might wonder why a cognitive skill should be broken up into rule-like units. Why not just encode the whole skill as one undifferentiated procedure? The major motivations are economy of representation and range of transfer. The five production rules in Table 1.1 can combine in various sequences to handle a wide range of addition problems. If one tried to solve the problem in large units, it would be necessary to encode special case rules for each type of addition problem (number of columns and pattern of carries). Thus, production rules tend to carve up a task at its natural joints, forming one rule for each natural unit. This explains why task analysis, which studies the structure of problems in a domain, plays such a major role in production-rule modeling.

The essential claim about modularity is that production rules are the units of the skill, which means that they are the units in which the skill is acquired, and they define the grain size at which the skill is performed. Skills are acquired in production-sized units, and any transformations of a skill occur as changes in production units. This means that a skill grows by acquiring new productions and by strengthening existing productions. This assertion often leads to the expectation that we should be able to see a step-by-step improvement in overall skill performance, but this expectation is frequently frustrated by the interactions that can occur among productions. Consider skill acquisition in the domain of learning to program in LISP, a topic that has been studied intensively in our laboratory. Figure 2.2 traces students' performance over the first six LISP problems in chapter 3 of Anderson, Corbett, and Reiser (1987). Because these problems vary in size, the measures reported are time and accuracy per unit of code. Note that there does not appear to be any systematic learning trend as a function of number of problems. It is common in observing skill acquisition for surface behavior to show a frustrating lack of learning.

The reason for this is that different problems involve different production rules, and some of the later problems involve many new productions and fewer old ones, whereas others offer few new and relatively more old ones. The LISP tutor (see chapter 7 of this volume, and Anderson, Conrad, & Corbett, 1989) decomposes this learning performance into separate measures of performance on specific production rules. Thus, rather than plotting performance as a function of number of problems, performance can be plotted as a function of number of opportunities for practice of a specific rule that is involved in producing a unit of LISP code.

One unit of code in the first problem might be generated by a production rule that was already practiced twice. It would, therefore, be considered as the third production opportunity, whereas in Fig. 2.2 it would be plotted as part of the first problem. Similarly, another unit of code might be generated in the third problem by a production rule that is firing for the first time. It would be considered as the first production opportunity, although it would have been plotted in Fig. 2.2 as part of the third problem. Figure 2.3 plots performance as a function of production opportunity. What was chaos in Fig. 2.2 is here transformed into systematic learning trends. It is data of this sort, more of which is reported throughout the book, that is evi-
of smaller units (e.g., neural elements) or larger units (e.g., schemas). Production rules are the right grain size for understanding skill acquisition. We will be too busy with the details in subsequent chapters to remark much upon this fact but it is truly remarkable. More than anything else, it justifies the principal assertion of the first chapter: that cognitive skills are realized by production rules.

2.4 Production Abstraction and Goal Structuring

Production rules are often compared with the stimulus–response bonds from an earlier era in psychology. A stimulus–response bond encoded the contingency that, if a certain stimulus was present (e.g., a red light), then a certain response would be emitted (e.g., a peck on a key). A production rule has just this structure with its condition and action. Production rules, however, offer certain generalities and restrictions over stimulus–response bonds, which give them their power. Production-rule abstraction and goal structuring capture two of the essential ways in which production rules differ from stimulus–response bonds.

Abstraction refers to the generality of production rules. Production rules do not require that a specific stimulus be present; the rules will apply in any stimulus condition that satisfies the pattern specification of the condition. Thus, an important issue is just what patterns one allows to be specified in the condition side of a production. A theory of this would allow us to specify exactly how knowledge will transfer. This is the topic of chapter 9.

Basically, what one can specify in an ACT-R production is an interconnected set of knowledge chunks. What is critical is the configuration and not the exact chunks. Thus, PROCESS-COLUMN in Table 1.1 does not depend on the exact numbers but only on the fact that two digits are in the column of focus and that they have a specific sum. As I show in the formal specification of the production rules, this generality is achieved by variabilizing each chunk. Thus, the variable is the critical element in achieving production-rule generality. Oddly, it is proper treatment of variables that is causing some of the greatest difficulties for connectionist theories of mind (Smolensky, 1990).

There is, however, a restriction on the application of a production rule that does not have a counterpart in a stimulus–response bond. Production-rule conditions not only make reference to certain external situations but also specify certain goal conditions. Different production rules can fire in response to the same external situation depending on the internal goal. For instance, given different goals a system can choose to add or subtract the same array of numbers. An ability to respond differently to the same stimulus condition is critical to the adaptiveness of the system.

Evidence for these two properties of production rules can also be found in the domain of learning how to program in LISP. A good test case concerns the
coding of variable names. In writing a computer program, one has to create a variety of variable names (e.g., \texttt{counter}, \texttt{base}, \texttt{salary}, etc.) and place them at different points in the program. The skill of coding variables could not be represented by a stimulus-response rule calling for the same action whenever a specific stimulus situation occurs. Thus, the coding of a variable name is an example of an abstract, non-stimulus-response skill. On the other hand, one can code variables for different purposes in LISP. In our LISP curriculum one starts out by coding variables as global variables to hold values, then uses them for parameters of a function, and finally employs them as local variables to hold temporary results. In the expert model for LISP programming, there are three separate productions for coding variables in response to these three goals.

Figure 2.4 illustrates the course of performance on these variable-coding productions in terms of number of errors per production. (The maximum number in the LISP tutor is three.) First we plot the average performance in Lesson 1, where these are global variables. They are introduced in Lesson 2 as parameters and continue in that role throughout Lessons 3 and 4. The figure shows their performance over Lesson 2 and their average performance in Lesson 4. Finally, it plots average performance in Lesson 5, where they are first introduced as local variables. At each point where a new use is introduced, there is a big increase in error rate. Within Lesson 2 there is a constant improvement with repeated use for the same purpose. There is continued improvement over lessons as is indicated by the performance in Lesson 4. Thus, even though it

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure2_4.png}
\caption{Accuracy in coding variables at various points in the LISP curriculum.}
\end{figure}

is not the same stimulus-response pair, there is constant improvement as long as the goal stays constant. When the goal changes, performance deteriorates because a new production rule must be learned. Thus, productions, with their variable abstraction and goal-factoring, seem to capture the range of generalization of a skill.

2.4.3 Condition-Action Asymmetry

Production rules contrast with other formalisms, such as schema representations,\textsuperscript{14} in that production rules make a distinction between condition and action. A rule will work if its condition matches and will perform its action. It is not possible to have the rule reverse itself and go from action to condition. It is this commitment to direction of action that gives the production rule its efficiency. There are, however, circumstances in which such reversal is logically possible, and many people have the intuition that we should be able to reverse our knowledge. For example, many people believe we should be able to employ the same knowledge in language generation as in language comprehension. Although we have not researched this issue in the area of natural language, in the area of problem solving we have consistently supported the condition-action asymmetry and found evidence against this belief in the symmetrical use of knowledge.\textsuperscript{15}

A good case in point comes, again, from the domain of learning to program in LISP. Kessler (1988) compared learning to write LISP code versus learning to evaluate it. Some subjects were told to write a function that rotates a list one to the right, such as:

\begin{verbatim}
(defun rotater (lis) (append (last lis) (reverse (cdr (reverse lis)))))
\end{verbatim}

In the other condition, subjects might be given this code and asked to step through how the code would apply to a list (a b c d). Thus, subjects were either asked to go from the desired behavior of the function to the code for the function or vice versa. The experiment involved crossing three conditions of the

\textsuperscript{14}I refer here not to the passive data schema structures used for chunks in ACT-R, but to the active schemas found in the work of Abelson (1981), Schank and Abelson (1977), Bobrow and Winograd (1977), Rumelhart and Ortony (1976), and the more recent neural network equivalents in, for example, Rumelhart, Smolensky, McClelland, and Hinton (1986).

\textsuperscript{15}For a brief consideration of the relevant evidence in the domain of language, see Siple and Anderson (1989, pp. 139-140).

\textsuperscript{16}It is not important that the operation of this function be understood to appreciate the point in text, but for those who care: Reverse reverses a list and cdr removes the first element. Thus, (reverse (cdr (reverse lis)))) reverses a list, removes the former last element, and reverses it back again. Thus, if lis were (a b c d) this would return (a b c). Last returns a list of the last element and append appends two lists together. Thus, if lis were (a b c d), the whole code would append (d) and (a b c) to yield (d a b c).
2. KNOWLEDGE REPRESENTATION

### TABLE 2.2
Effect of Prior Experience on Ability to Code and to Evaluate Code

<table>
<thead>
<tr>
<th>Prior Experience</th>
<th>Coding Time per Problem (Errors)</th>
<th>Evaluation Time per Problem (Errors)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coding</td>
<td>163 sec. (2.0)</td>
<td>358 sec. (8.6)</td>
</tr>
<tr>
<td>Evaluation</td>
<td>304 sec. (4.3)</td>
<td>231 sec. (5.1)</td>
</tr>
<tr>
<td>Nothing</td>
<td>412 sec. (4.9)</td>
<td>376 sec. (7.2)</td>
</tr>
</tbody>
</table>

*Data from Kessler (1988).

Initial experience with two transfer tasks. Subjects were given no prior experience, prior experience in generating such code, or prior experience in evaluating such code. They were then transferred to one of two conditions for the transfer task: generating more code or evaluating more code. Table 2.2 shows their performance on the transfer task as a function of their prior experience. Subjects were much faster and more accurate if their transfer task matched their training task. When the target task was coding, subjects with prior experience evaluating code seemed to be at some advantage over those in the control condition, but it is not a statistically significant difference. When the target task was evaluation, subjects with prior experience coding showed no advantage over control subjects.

There is an asymmetry in the use of knowledge. This basic result with respect to LISP has been replicated a number of times (McKendree & Anderson, 1987; Pennington & Nicolich, 1991). As I discuss in chapter 9, the prediction of ACT-R is that there will be no transfer from one use of knowledge to another. Productions for each use arise from a common declarative representation, so practicing the knowledge in one way offers the opportunity to practice that declarative representation and so enhance the other use. Pennington and Nicolich (1991) were able to get significantly better performance in a transfer condition than in a control condition, but there are strong limitations on the amount of transfer, and strong asymmetries result from the fact that different uses of the same knowledge cannot be implemented by the same production rules.

### 2.4.4 A Notation for Productions

Having now specified the features that distinguish productions, we need to decide on the notation for specifying them. In addition to the features enumerated, a strong constraint is that they have to read from and write to working memory. Thus, the formalism that specifies the chunks in working memory forms the foundation for the formalism that specifies the production rules. We just need to add notation to indicate variables, goals, conditions, and actions.

Table 2.3 shows the formal specification of the production rules in Table 1.1.
This is essentially the actual code for the production rules and what is found on the accompanying disk. A production rule begins with a specification of a series of working memory chunks which define its condition. The condition is separated by an arrow from the action of the production. The first working-memory chunk, denoted goal, is the goal that the production refers to. Following it are a series of working-memory elements. All of these working-memory elements are variabilized. Values of various slots, except the isa slot, can be variabilized. Variables are denoted by the prefix $=,$ a frequent convention in production systems, although not the convention used in previous ACT systems. The $>$ symbol in Table 2.3 separates the chunk name from its slots.

PROCESS-COLUMN is probably the easiest production rule to follow in the table. It applies when the goal is set to write out an answer in the column, but the answer is not yet calculated (i.e., the value of the object slot is nil). The pattern:

```
=column>
    isa column
    toprow = num1
    bottomrow = num2
```

The complete code is in the Addition file in the Examples folder. This contains a few more parentheses and LISP function calls than what is shown in Table 2.3.
retrieves the digits in the column and binds them to =num1 and =num2. The pattern:

- fact
  isa addition-fact
  addend1 =num1
  addend2 =num2
  sum = sum

retrieves their sum and binds this to =sum. The action side of this production just fills in the object slot of the goal with the sum of =num1 and =num2.

The first production NEXT-COLUMN in Table 2.3 illustrates the true complexity of production rules when they are implemented in ACT-R. Basically, it retrieves the rightmost unanswered column. The pattern:

- array
  isa numberarray
  columns ($ = column2 $)
- column2
  isa column
  - bottomrow +
  answerrow blank

matches a column, =column2, with a blank answer. The construct ($ = column2 $) matches an arbitrary column in the list of columns with the $ being string variables that match anything to the left or right. The test "bottomrow +" checks that we have not come to the end of the problem as signalled by the plus sign. To determine that this is the leftmost such column, a test is needed to guarantee that the column to the right does not also have a blank answer. This is done by negating (indicated by a minus sign in the NEXT-COLUMN) the following pattern, which represents a situation where the column to the right is blank:

- array
  columns ($ = column2 = column1 $)
- column1
  isa column
  answerrow blank

The action of the production sets a subgoal to write an answer in the column and pushes it by lpush command. This subgoal is popped by the \( \text{pop} \) actions of WRITE-ANSWER-LESS-THAN-TEN and WRITE-ANSWER-GREATER-THAN-NINE. ACT-R assumes a last-in-first-out (LIFO) goal stack, which

remembers subgoals and reactivates subgoals, such that when a subgoal is satisfied (popped), the next most recent subgoal will become active. For instance, the production system in Table 2.3 focuses on the subgoal of doing a column; when that is satisfied, it returns to the main goal of solving the addition problem. The next chapter and chapter 6 elaborate on the meaning of such goal actions and the evidence for them.

The production NEXT-COLUMN reveals the reason that columns are represented in an unbounded list rather than some hierarchical chunk structure with limited size for each chunk. The pattern ($ = column2 $) retrieves a column anywhere and ($ = column2 = column1 $) finds the preceding column. These patterns are insensitive to the serial positions of the columns. This is psychologically unrealistic, as people probably are influenced by the serial structure of the columns. In a second way, this production rule is unrealistic in that it models a situation in which all of the column structure is represented in working memory. It would be more reasonable to assume some scheme for moving focus across an external addition array and just encoding a column or two at any one time. This illustrates the point made earlier that when we find ourselves violating limits on chunk size, it is a sign that there are some unreasonable representational assumptions at work. These assumptions may be convenient approximations for some purposes, but they can be disastrous if they mask the psychological processes of interest.

Such machine-readable code specifications do not appear again until chapter 12. For the purposes of this book, I use more readable English specifications like those in Table 1.1. At one research meeting, I had people rate the specifications in Tables 1.1 and 2.3 for clarity. On a 10-point scale where 1 was totally obscure and 10 was totally transparent, Table 1.1 received an average rating of 8.6 and Table 2.3 an average rating of 4.3. Because this was a group relatively sophisticated about actual production systems, the 4.3 probably overestimated the average clarity of Table 2.3. The reason one needs such formal specifications at all is that there is no guarantee that the English production set in Table 1.1 actually corresponds to something that would work. Indeed, in writing such formal production-rule models, it is not unusual to discover flaws in one's original conception of the skill. Problems of vagueness, contradiction, and incompleteness that arise in natural language statements may only become apparent when productions are converted to code.

2.5 CONCLUSIONS

A chapter on representation has to come early in a book on cognitive architecture because one needs to establish a notation for describing the elements of cognition. The actual theoretical claims of this notation are relatively few, but they are important. Even where an important concept is being advanced, such
2. KNOWLEDGE REPRESENTATION

As the procedural–declarative distinction, it is not the case that the representational accompaniment (declarative chunks vs. procedural productions) is absolutely necessary. As briefly noted, one could get a system without the distinction to mimic a system with the distinction, if at the cost of clarity. So, even here the representational assumptions are just notation to make important theoretical points transparent. The theoretical points are more in the processes that are defined on the notation. These processes are set forth for the case of ACT-R in the next two chapters. These next two chapters therefore assume, at times, a somewhat more technical character.

3.1 GENERAL ISSUES

3.1.1 Conflict Resolution

The previous chapter specified the knowledge representation in terms of simple and complex productions that, in turn, were specified in terms of simple and complex chunks. The functional connections among these productions and chunks were specified in terms of the usual procedural connectionist architecture. In this chapter, we will consider a different model for the representation of these functional connections. This model is based on the idea of a grid, or network. The grid consists of a set of nodes, each of which represents a possible conflict between two productions. The nodes are connected by directed edges, which indicate the direction of the conflict. The grid is traversed by a rule model for