10

THE CHUNKING OF GOAL HIERARCHIES: A Generalized Model of Practice

Paul S. Rosenbloom
Stanford University

Allen Newell
Carnegie-Mellon University

Abstract

This chapter describes recent advances in the specification and implementation of a model of practice. In previous work the authors showed that there is a ubiquitous regularity underlying human practice, referred to as the *power law of practice.* They also developed an abstract model of practice, called the *chunking theory of learning.* This previous work established the feasibility of the chunking theory for a single 1023-choice reaction-time task, but the implementation was specific to that one task. In the current work a modified formulation of the chunking theory is developed that allows a more general implementation. In this formulation, task algorithms are expressed in terms of hierarchical goal structures. These algorithms are simulated within a goal-based production-system architecture designed for this purpose. *Chunking* occurs during task performance in terms of the parameters and results of the goals experienced. It improves the performance of the system by gradually reducing the need to decompose goals into their subgoals. This model has been successfully applied to the task employed in the previous work and to a set of stimulus-response compatibility tasks.

10.1 INTRODUCTION

How can systems—both natural and artificial—improve their own performance? At least for natural systems (people, for example), we know that *practice* is effective. A system is engaged in practice when it repeatedly performs one task or a set of similar tasks. Recently, Newell and Rosenbloom (1981) brought together the evidence that there is a ubiquitous law—the *power law of practice*—that characterizes the improvements in human performance during practice. The law states that when human performance is measured in terms of the time needed to perform a task, it improves as a power-law function of the number of times the task has been performed (called the *trial number*). This result holds over the entire domain of human performance, including both purely perceptual tasks, such as target detection (Neisser, Novick, and Lazar, 1963), and purely cognitive tasks, such as supplying justifications for geometric proofs (Neves and Anderson, 1981).

The ubiquity of the power law of practice suggests that it may reflect something in the underlying *cognitive architecture*. The nature of the architecture is of fundamental importance for both artificial intelligence and psychology (in fact, for all of cognitive science; see Newell, 1973; Anderson, 1983a). It provides the control structure within which thought occurs, determining which computations are easy and inexpensive as well as what errors will be made and when. Two important ingredients in the recent success of expert systems come from fundamental work on the cognitive architecture; specifically, the development of *production systems* (Newell and Simon, 1972; Newell, 1973) and *goal-structured problem solving* (Ernst and Newell, 1969; Newell and Simon, 1972). This chapter discusses recent efforts to take advantage of the power law of practice by using it as a generator for a general production-system practice mechanism. This is a highly constrained task because of the paucity of plausible practice models that can produce power-law practice curves (Newell and Rosenbloom, 1981).

As a beginning, Newell and Rosenbloom (1981) developed an abstract model of practice based on the concept of *chunking*—a concept already established to be ubiquitous in human performance—and derived from it a practice equation capable of closely mimicking a power law. It was hypothesized that this model formed the basis for the performance improvements brought about by practice. In sections 10.2 and 10.3 this work on the power law and the abstract formulation of the chunking theory is briefly summarized.

Rosenbloom and Newell (1982a, 1982b) took the analysis one step further by showing how the *chunking theory of learning* could be implemented for a single psychological task within a highly parallel, activation-based production system called XAPS2. This work established more securely that the theory is a viable model of human practice by showing how it could actually be applied to a task to produce power-law practice curves. By producing a working system, it also established the theory's viability as a practice mechanism for artificial systems.

The principal weakness of the work done up to that point was the heavy task dependence of the implementation. Both the representation used for describing the task to be performed and the chunking mechanism itself had built into them knowledge about the specific task and how it should be performed. The work reported here is focused on the removal of this weakness by the development of generalized, task-independent models of performance and practice.

This generalization process has been driven by the use of a set of tasks that fall within a neighborhood around the task previously modeled. That task sits within an experimental domain widely used in psychology, the domain of *reaction-time* tasks (Woodworth and Schlosberg, 1954). Reaction-time tasks involve the presentation to a subject of stimulus display—such as an array of lights, a string of characters on a computer terminal, or a spoken word—for which a specific "correct" response is expected. The response may be spoken, manual (such as pressing a button, pushing a lever, or typing), or something quite different. From the subject's reaction time—the time it takes to make the response—and error rate, it is possible to draw conclusions about the nature of the subject's cognitive processing.

The particular task, as performed by Seibel (1963), was a 1023-choice reaction-time task. It involved ten lights (strung out horizontally) and ten buttons, with each button right below a light. On each trial of the task some of the lights came on while the rest remained off. The subject's task was to respond as rapidly as possible by pressing the buttons corresponding to the lights that were on. There were $2^{10} - 1$, or 1023, possible situations with which the subject had to deal (excluding the one in which all ten lights were off). Rosenbloom (1983) showed that a general task representation based on the concept of *goal hierarchies* (discussed in section 10.4) could be developed for the performance of this task.

In a goal hierarchy, the root node expresses a desire to do a task. At each level further down in the hierarchy, the goals at the level above are decomposed into a set of smaller goals to be achieved. Decomposition continues until the goals at some level can be achieved directly. This is a common control structure for the kinds of complex problem-solving systems found in artificial intelligence, but this is the first time they have been applied to the domain of reaction-time tasks.

Goal hierarchies also provided the basis for models of a set of related reaction-time tasks known as *stimulus-response compatibility* tasks. These tasks involve fixed sets of stimuli and responses and a mapping between them that is manipulated. The main phenomenon is that more complex and/or "counterintuitive" relationships between stimuli and responses lead to longer reaction times and more error. A model of stimulus-response compatibility based on goal hierarchies provides excellent fits to the human reaction-time data (Rosenbloom, 1983).

The generalized practice mechanism (described in section 10.5) is grounded in this goal-based representation of task performance. It resembles a form of store-versus-compute trade-off, in which composite structures (chunks) are created that relate patterns of goal parameters to patterns of goal results.

These chunking and goal-processing mechanisms are evaluated by implementing them as part of the architecture of the XAPS3 production system. The XAPS3 architecture is an evolutionary development from the XAPS2 architecture. Only those changes required by the needs of chunking and goal processing have been made. This architecture is described in section 10.6. From this implementation simulated practice results have been generated and analyzed for the Seibel and compatibility experiments (section 10.7).[1]

Following the analysis of the model, this work is placed in perspective by relating the chunking theory to previous work on learning mechanisms (section 10.8). The theory stakes out an intermediary position among four previously disparate mechanisms, bringing out an unexpected commonality among them.

Before concluding and summarizing (section 10.10), some final comments are presented on ways in which the scope of the chunking theory can be expanded to cover more than just the speeding up of existing algorithms (section 10.9). Specifically, the authors describe a way in which chunking might be led to perform other varieties of learning, such as generalization, discrimination, and method acquisition.

10.2 THE POWER LAW OF PRACTICE

Performance improves with practice. More precisely, the time needed to perform a task decreases as a power-law function of the number of times the task has been performed. This basic law, the power law of practice, has been known since Snoddy (1926). This law was originally recognized in the domain of motor skills, but it has recently become clear that it holds over a much wider range of human tasks, possibly extending to the full range of human performance. Newell and Rosenbloom (1981) brought together the evidence for this law from tasks involving perceptual-motor skills (Snoddy, 1926; Crossman, 1959), perception (Kolers, 1975; Neisser, Novick, and Lazar, 1963), motor behavior (Card, English, and Burr, 1978), elementary decisions (Seibel, 1963), memory (Anderson, 1980), routine cognitive skill (Moran, 1980), and problem solving (Neves and Anderson, 1981; Newell and Rosenbloom, 1981).

Practice curves are generated by plotting task performance against trial number. This cannot be done without assuming some specific *measure* of performance. There are many possibilities for such a measure, including such things as quantity produced per unit time and number of errors per trial. The power law of

[1]A more comprehensive presentation and discussion of these results can be found in Rosenbloom (1983).

practice is defined in terms of the *time* to perform the task on a trial. It states that the time to perform the task (T) is a power-law function of the trial number (N):

$$T = BN^{-\alpha} \tag{1}$$

As shown by the following log transform of Equation 1, power-law functions plot as straight lines on log-log paper:

$$\log(T) = \log(B) + (-\alpha)\log(N) \tag{2}$$

Figure 10-1 shows the practice curve from one subject in Kolers' study (1975) of reading inverted texts—each line of text on the page was turned upside down—as plotted on log-log paper. The solid line represents the power-law fit to this data. Its linearity is clear ($r^2 = 0.932$).

Many practice curves are linear (in log-log coordinates) over much of their range but show a flattening at their two ends. These deviations can be removed by using a four-parameter *generalized* power-law function. One of the two new parameters (A) takes into account that the asymptote of learning can be greater than zero. In general, there is a nonzero minimum bound on performance time, determined by basic physiological limitations and/or device limitations—if, for example, the subject must operate a machine. The other added parameter (E) is required because power laws are not translation invariant. Practice occurring before the official beginning of the experiment—even if it consists only of transfer of training from everyday experience—will alter the shape of the curve, unless the effect is explicitly allowed

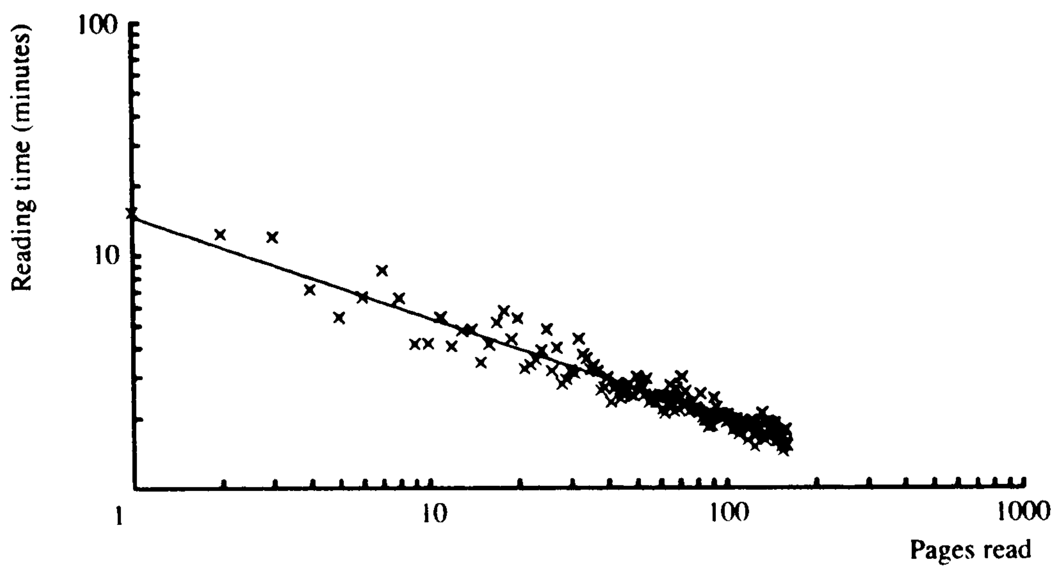


Figure 10-1: Learning to read inverted text (log-log coordinates). Plotted from the original data for Subject HA (Kolers, 1975).

for by the inclusion of this parameter. Augmenting the power-law function by these two parameters yields the following generalized function:

$$T = A + B(N + E)^{-\alpha} \tag{3}$$

A generalized power law plots as a straight line on log-log paper once the effects of the asymptote (A) are removed from the time (T), and the effective number of trials prior to the experiment (E) are added to those performed during the experiment (N):

$$\log(T - A) = \log(B) + (-\alpha)\log(N + E) \tag{4}$$

Figure 10-2 shows a practice curve from the Seibel task (Seibel, 1963), as fit by a generalized power-law function (each data point represents the mean reaction time over a block of 1023 trials). This curve, which shows flattening at both ends when plotted as a simple power law, is now linear over the whole range of trials. As stated earlier, similar fits are found across all dimensions of human performance. Though these fits are impressive, it must be stressed that the power law of practice is only an *empirical* law. The true underlying law must resemble a power law, but it may have a different analytical form.

10.3 THE CHUNKING THEORY OF LEARNING

The chunking theory of learning proposes that practice improves performance via the acquisition of knowledge about patterns in the task environment. Implicit in this theory is a model of task performance based on this pattern knowledge. These

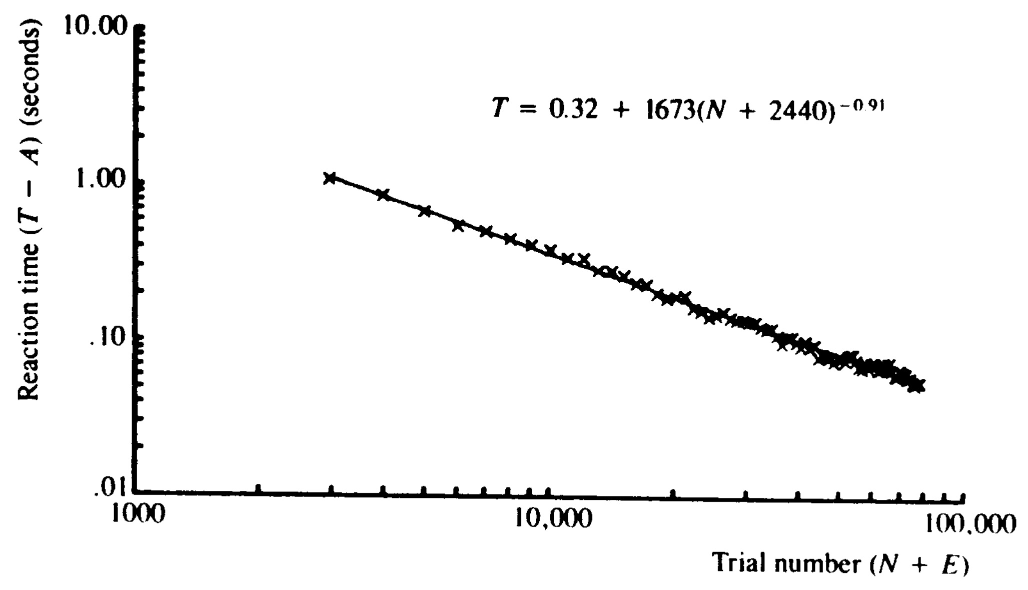


Figure 10-2: Optimal general power-law fit to the Seibel data (log-log coordinates).

patterns are called *chunks* (Miller, 1956). The theory thus starts from the *chunking hypothesis:*

The chunking hypothesis: A human acquires and organizes knowledge of the environment by forming and storing expressions, called *chunks*, that are structured collections of the chunks existing at the time of learning.

The existence of chunks implies that memory is hierarchically structured as a lattice (tangled hierarchy, acyclic directed graph, and so on) rooted in a set of preexisting *primitives*. A given chunk can be accessed in a top-down fashion, by *decoding* a chunk of which it is a part, or in a bottom-up fashion, by *encoding* from the parts of the chunk. Encoding is a recognition or parsing process.

The existence of chunks does not need to be justified solely on the basis of the practice curves. Chunks stand on their own as a thoroughly documented component of human performance (Miller, 1956; DeGroot, 1965; Bower and Winzenz, 1969; Johnson, 1972; Chase and Simon, 1973; Chase and Ericsson, 1981). The traditional view of chunks is that they are data structures representing a combination of several items. For example, in one set of classic experiments, Bower and colleagues (Bower, 1972; Bower and Springston, 1970; Bower and Winzenz, 1969) showed that recall of strings of numbers or letters is strongly influenced by the segmentation of the string. If the segmentation corresponds to a previously learned grouping of the items (for example, FBI-PHD-TWA-IBM), performance is better than if no such relation obtains (FB-IPH-DTW-AIB-M). These results were interpreted as evidence for segmentation-guided chunking of familiar strings. By replacing a string of several letters with a single chunk, the subject's memory load is reduced, allowing more letters to be remembered. At recall time the chunks are decoded to yield the original items to be recalled.

The chunking theory of learning proposes two modifications to this classical view. The first change is the assumption that there is not a single symbol (chunk) to which the items are encoded and from which they can later be decoded. As a simple example, the process of reading the string IBM out loud is more than just the encoding of the three letters into a chunk, followed by the subsequent decoding to the three letters. What needs to be decoded is not the visual representation of IBM, but the articulatory representation—allowing the subject to *say* "IBM."

Based on this consideration, the chunking theory assumes that there are two symbols for each chunk—a *stimulus* symbol and a *response* symbol. The process of using a chunk consists of encoding the stimulus items to the stimulus symbol (a many-one mapping), mapping the stimulus symbol to the response symbol (a one-one mapping), and decoding the response symbol to the response items (a one-many mapping) (see figure 10-3). Encoding and decoding are fast parallel hierarchical processes, and the mapping serves as a (serial) point of control at which the choice of response can be made. Acquisition of a chunk speeds up performance by reducing the number of mappings to be performed. In the example in figure 10-3, before the chunk

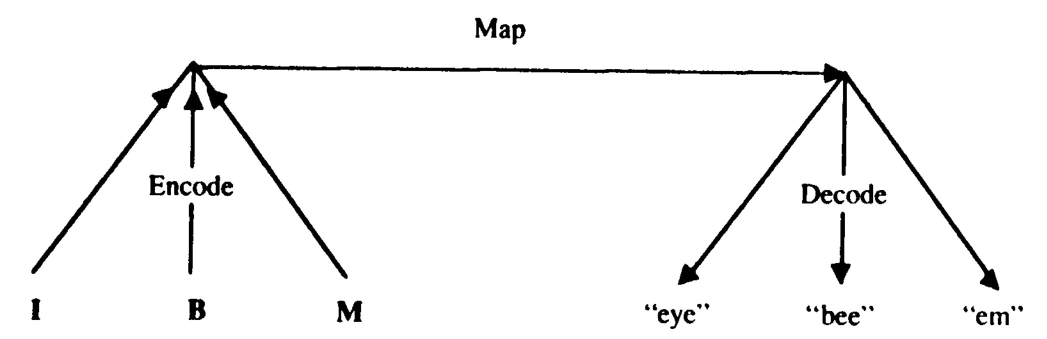


Figure 10-3: A three-part chunk for the articulation of the visual string "IBM."

is acquired, three mappings (one for each letter) are required. After acquisition of the chunk, one mapping suffices.

The second difference between this proposal and the classical view of chunking is the idea that the chunk consists of the three processes (encoding, mapping, and decoding), not just the symbols. This is really just a shift of emphasis; chunks are viewed as the processes rather than just as the results of the processes.

10.4 GOAL-STRUCTURED PERFORMANCE MODELS

In its current formulation, chunking explains how performance on a task can be sped up with practice. It does not explain how the organism first learns to do the task (but see the discussion in section 10.9). Consequently, each task simulation must be initialized with a performance model for the task. What is needed—and is provided by the notion of a *goal hierarchy*—is a general, task-independent representation for these performance models. Goal hierarchies are frequently found in artificial intelligence systems, but they have not previously been employed in the modeling of the kinds of reaction-time tasks dealt with here.

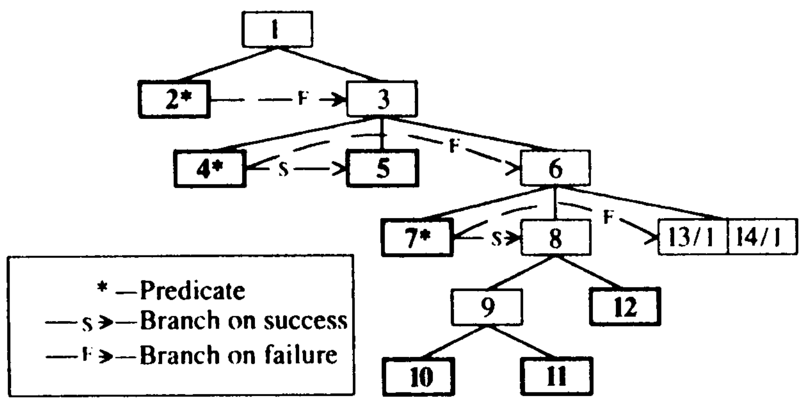
Goal hierarchies are built out of *goals;* each goal is a data structure representing a desired state of affairs. A goal is not a procedure for bringing about that state; it is only a description of the state. In order for the goal state to be brought about, there must be a *method* associated with the goal. The method could be a rigid algorithm, or it could be one of the more flexible *weak methods* (Newell, 1969), such as means-ends analysis (Ernst and Newell, 1969) or heuristic search (Nilsson, 1971). In the discussion that follows, the properly distinct notions of goal and method will be conflated together into a single active concept for the sake of convenience. These conflated "goals" are active processes that take a set of *parameters* and return a set of *results*.

When a goal can be decomposed into a set of simpler goals (Nilsson, 1971) and those goals can be decomposed even further, a goal hierarchy results. In its simplest form, as an AND hierarchy, a goal is successful if all of its subgoals are successful. The structure to be described here more closely resembles an AND/OR hierarchy, in which some goals succeed only if all of their subgoals succeed, and others succeed if

any one of their subgoals succeed. A *terminal* goal is reached when the goal can be fulfilled directly, without the need for further decomposition.

We use a *depth-first* strategy for processing a goal hierarchy, in which the most recently generated (the deepest) goal is always selected as the next goal to process. With a depth-first paradigm there is always exactly one goal being actively worked on at any point in time. We will refer to this goal as the *active* or *current* goal. When a subgoal becomes the active goal, the parent goal of that subgoal is *suspended* until control is returned to it by completion of the subgoal, at which point it again becomes the current goal. On completion, the subgoal will have either *succeeded* or *failed*. The *control stack* specifies the location in the hierarchy at which control currently resides. It does this by maintaining the path from the root goal of the hierarchy to the current goal. This path consists of the active goal and all of its suspended ancestors.

Figure 10-4 shows two different representations of a goal hierarchy for the Seibel (1963) 1023-choice reaction-time task. At the top of the figure, the hierarchy is shown as a tree structure. At the bottom of the figure, the goals are shown in their depth-first processing order. In both representations the boldface goals are the



1. Do-Lights-If-Any (Min-X, Max-X)
  2. **No-Light-On?** (Min-X, Max-X)
  IF-FAILED *No-Light-On?* THEN
    3. Do-Lights (Min-X, Max-X)
      4. **No-Light-Off?** (Min-X, Max-X)
      IF-SUCCEEDED *No-Light-Off?* THEN
        5. **Do-Press-All-Buttons** (Min-X, Max-X)
      IF-FAILED *No-Light-Off?* THEN
        6. Do-Off-And-On (Min-X, Max-X)
          7. **One-Light-On?** (Min-X, Max-X)
          IF-SUCCEEDED *One-Light-On?* THEN
            8. Press-Button-Under-On-Light (Min-X, Max-X)
              9. Get-On-Light-X (Min-X, Max-X)
                10. **Get-On-Light-Stimulus** (Min-X, Max-X)
                11. **Get-Stimulus-X**
              12. **Press-Button-At-X**
          IF-FAILED *One-Light-On?* THEN
            13. Do-Lights-If-Any (Min-X, [Min-X + Max-X]/2)
            14. Do-Lights-If-Any ([Min-X + Max-X]/2, Max-X)

Figure 10-4: Goal hierarchy for the Seibel (1963) task.

terminals—those goals that can be fulfilled directly. The labeled arrows (either S or F) represent *branch* points in the hierarchy. With simple depth-first processing, the children of a node are processed in a strict left-to-right fashion. In this work, this style of control has been augmented by allowing the left-to-right processing to be conditioned on the success or failure of the previous child (actually, of any previous goal for which this information is still available). For example, goal 2 (**No-Light-On?**) tests whether there are any lights on within a specific region of the display of lights. Only if this goal fails should goal 3 (Do-Lights) be attempted. If goal 2 succeeds, then there are no lights on that need to be processed and the parent goal (goal 1: Do-Lights-If-Any) can be terminated successfully. Those goals that do not return a result are called *predicates* (denoted by an asterisk in figure 10-4) and are used to test the truth of various conditions. It is only the success or failure of predicates that matters (as the basis for a branch). If the predicate succeeds, then the condition is true. If it fails, the reason could be either that the condition is false or that something went wrong during the attempt.

The goal structure in figure 10-4 is based on a recursive divide-and-conquer algorithm in which the stimulus display is broken up into smaller and smaller horizontal segments until *manageable* pieces are generated. There are three types of horizontal segments that have been defined as manageable. The first type of manageable segment is one in which no lights are on. Such segments require no explicit processing, so the goal just returns with success. The opposite of the first type of segment—one in which no lights are off—is also manageable. For such a segment, the system generates a single response specifying that a *press* action must occur in the entire region defined by the segment (using the **Do-Press-All-Buttons** goal). Specifying a single button press is actually a special case of this, in which the region is just large enough to contain the one button. Allowing multi-on-light segments to be manageable implies that sequences of adjacent on-lights can be pressed simultaneously even before chunking has begun. Such behavior is seen very early in the trial sequence for some subjects. The remaining manageable segments are those that contain exactly one light on. These segments are processed (using the Press-Button-Under-On-Light goal) by finding the location of that light and generating a button press at that location. If a generated segment does not meet any of these three criteria, it is unmanageable and is split into two smaller segments.

The recursive aspect of the algorithm implies that many different instances of each goal will be simultaneously represented in the system, although at most one can actually be active. It is necessary to keep track of which goal instance is relevant to which segment of the stimulus display, so the segment (in terms of its minimum and maximum X values) is an *explicit* parameter to the goals. In addition to the explicit parameters, a goal can have *implicitly* defined parameters. Any object existing before the activation of the goal (i.e., as part of the goal's *initial state*) that is examined during the processing of the goal—such as a stimulus light—is an implicit parameter of the goal. Implicit parameters are dynamically determined by the actual processing of the goal.

A second implication of the recursion in this hierarchy is that the system does not start off with a constant built-in goal hierarchy. Instead, it has a generator of goal hierarchies. That chunking works on such a structure is important for any claims about the potential generality of the mechanism (for more on this, see section 10.9).

The recursion occurs at goals 13 and 14 in figure 10-4. They are repetitions of the topmost goal in the hierarchy (Do-Lights-If-Any), but the scope of each is limited to one-half of the display currently being processed. The numeric computation to obtain the middle of the segment (involving an addition and a division), while on the surface too powerful a computation to appear where it does, is only intended as an approximation to a process that divides the stimulus display into two (or three) roughly equal parts.

In addition to the goal hierarchy, the model assumes a *working memory* for the short-term storage of information relevant to the processing that is going on. For each goal, the working memory is logically partitioned into two components—the *initial state* and the *local state*. The initial state consists of the data existing at the time the goal is first activated. The remainder of the working memory—consisting of those pieces of data created during the processing of the goal—makes up its local state. Only the local state of a goal can be modified during the processing of that goal; the initial state can be examined, but it cannot be modified. The modularity resulting from this scoping rule increases the likelihood that an arbitrary set of goals can be pursued without their interfering with each other. This modularity is also important in insuring correct performance of the chunking mechanism.

Each datum in the working memory is relevant in a particular temporal *processing context*. As long as the datum's creator goal is in the control stack, the system is working either on that goal or on one of its descendants. The datum may be relevant at any point in this processing. However, once the goal disappears from the control stack, it is no longer being pursued. The datum will most likely no longer be relevant. The processing context of a datum is that period of processing during which its creator goal is in the control stack. Once a piece of information becomes *out of context*—its creator is no longer part of the control stack—it usually can be safely deleted from the working memory. Data that cannot be deleted—because they are needed outside of the context in which they were created—are called the *results* of the goal. Their continued presence in the working memory is insured by changing their context to be the context of the parent goal.

10.5 CHUNKING ON GOAL HIERARCHIES

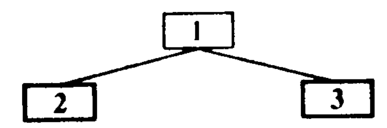
Given a task-independent organization for performance models, it is possible to return to the original objective of describing a task-independent formulation of the chunking theory. In the chunking mechanism described by Rosenbloom and Newell (1982a, 1982b), chunks related patterns of stimuli (lights) to patterns of responses (button presses). The goal-oriented formulation is obtained by altering this definition slightly so that chunks relate patterns of goal parameters to patterns of goal results. Each chunk improves the performance of the system by eliminating the need to process fully a specific instance (a combination of parameter values) of a particular goal. It replaces the normal processing (decomposition into subgoals for nonterminal goals and direct execution of an action for terminal goals) with a direct connection between the relevant parameter values and results. A goal can (and almost certainly will) have more than one possible chunk; each combination of parameter values requires its own chunk.

As with the abstract characterization of the chunking theory, each chunk consists of three components: encoding, decoding, and connection (or mapping). The goal's parameter values form the basis for the encoding component. Given the presence of those values in the working memory, the encoding component generates a new object representing their combination. Encoding is a parallel, goal-independent, data-driven process. Every encoding component executes as soon as appropriate, irrespective of whatever else is happening in the system. The results of encoding components can themselves become parameter values of other goals, leading to a hierarchical encoding process.

The results of the goal form the basis for the decoding component. Given the presence of an encoded result-object in the working memory, the decoding component generates the actual results returned by the goal. Decoding occurs when the results are needed. As with encoding, decoding is a parallel, goal-independent process. The set of decoding components forms a hierarchical structure in which complex results are decoded to simpler ones, which are then decoded even further.

The connection component of the chunk generates the encoded result from the encoded parameter. Connections provide a locus of control by occurring serially, under the control of the goals. Thus a connection can be made only when the system is working on the goal for which the chunk was formed (and after the encoding component has executed). This insures that, even though encoding and decoding are uncontrolled, only appropriate results are generated.

A chunk can be created for a goal when the following two conditions are met: (1) the goal has just completed successfully, and (2) all of the goal's subgoals were themselves processed by chunks. The first condition insures both that chunks are created for appropriate goals and that the chunk is created at a time when the information required for the chunk is available. The second condition causes chunks to be created bottom up in the goal hierarchy. It is this bottom-up aspect of chunking that leads to hierarchical encoding and decoding networks. However, notice that bottom-up chunking does not imply that all low-level chunks are learned before any high-level chunks are learned, or even that all of the chunks must be learned for a subgoal before any can be learned for its parent goal. The second condition on chunk creation merely states that chunks must exist for the goal's subgoals *in the current situation*. Whether other chunks exist or do not exist for the subgoals is irrelevant.



1. Compute-Average-Of-Two-Numbers
2. **Compute-Sum-Of-Two-Numbers**
3. **Divide-Sum-By-2**

Figure 10-5: A simple three-goal hierarchy for the averaging of two numbers.

Given enough practice with enough task variations, all of the situations for all of the goals in the hierarchy will be chunked, and asymptotic behavior will be reached for the task. The amount of time this takes depends on the number of goals, the number of situations for each goal, how frequently the different situations arise, and whether chunks are created whenever they can be.

The three-goal hierarchy shown in figure 10-5 provides a simple example of how chunking works. This structure computes the average of two numbers. The top-level goal (Computer-Average-Of-Two-Numbers) takes as parameters the two numbers to be averaged and returns to a single result, which is their mean. The first subgoal **(Compute-Sum-Of-Two-Numbers)** performs the first half of the computation. It takes the two numbers as parameters and returns their sum as its result. The second subgoal finishes the computation by taking the sum as a parameter and returning half of it as its result.

Suppose that the first task is to average the numbers 3 and 7. Control would pass from goal 1 to goal 2. When goal 2 finishes and returns its result of 10, a chunk of three components is created (bottom left of figure 10-6). An encoding component is created that encodes the two parameters (3 and 7) into a new symbol (E1). It executes as soon as it is created, because the parameters are in the working memory. A decoding component is created that decodes from a second new symbol (D1) to the result (10). A connection component (the horizontal line with the goal name above it and goal number below it) is created that generates the result symbol (D1) when it detects both the presence of the encoded parameter (E1) and that goal 2 is the active goal. The connection does not execute immediately because goal 2 is already complete when the chunk is created.

Following the termination of goal 2, goal 1 is reactivated but then is suspended in favor of goal 3 (**Divide-Sum-By-2**). When this goal terminates successfully (returning the number 5), a chunk is created for it (bottom right of figure 10-6). The encoding component encodes the number 10 into the symbol E2; the decoding component decodes from the symbol D2 to the number 5; and the connection component connects E2 to D2 (in the presence of an active goal 3). In contrast to the chunk for goal 1, this chunk can be used in more than one task situation. It can be used whenever goal 1 generates a sum of 10, whether it does it by adding 3 and 7, 5 and 5, or any other pair of numbers. This is a form of transfer of training.

Following the termination of goal 3, goal 1 is reactivated and terminates successfully (returning the number 5). No chunk is created for goal 1 because its subgoals were not processed by chunks. At this point, the task is complete.

Winston, P. H., "Learning Structural Descriptions from Examples," in *The Psychology of Computer Vision*, P. H. Winston (Ed.), McGraw-Hill, New York, 1975.

Woodworth, R. S., and Schlosberg, H., *Experimental Psychology*, rev. ed., Holt, Rinehart and Winston, New York, 1954.

11

KNOWLEDGE COMPILATION: The General Learning Mechanism

John R. Anderson
Carnegie-Mellon University

Abstract

The ACT learning mechanisms of knowledge compilation, consisting of composition and proceduralization, are discussed. Composition operates by collapsing multiple productions into a single production that has the effect of the set. Proceduralization operates by building into productions information that previously had to be retrieved from long-term memory. It is shown how these two mechanisms can simulate the initial stages of skill acquisition in the domain of learning how to program. It is also shown that these mechanisms can reproduce the effects that have been attributed to inductive learning mechanisms involving generalization and discrimination. Generalizations and discriminations emerge as consequences of compiling the processes of analogy formation and error correction.

11.1 INTRODUCTION

One of the oldest intellectual issues is whether all forms of learning can be accounted for by associative learning—that is, learning by associating co-occurring elements (see Anderson and Bower, 1973, for a historical review). One consequence of our new technical age has been to refine this question practically out of existence. However, the issue has received a new embodiment in the world of machine learning in which there is a set of proposals about learning by induction and discovery. These learning mechanisms assume implicitly that it is not possible for adequate learning to be achieved by means of association by contiguity. Approximately half the papers at the 1983 Machine Learning Conference had this character.