PSYCHOLOGICAL REVIEW

VOL. 79, NO. 5 SEPTEMBER 1972

COMPLEXITY AND THE REPRESENTATION OF PATTERNED SEQUENCES OF SYMBOLS[1]

HERBERT A. SIMON[2]

Carnegie-Mellon University

All of the principal alternative theories that explain behavior in tasks involving patterned sequences are variants on a basic theme. They agree in proposing that subjects perform these tasks by inducing pattern descriptions from the sequences, and that these descriptions incorporate the relations of "same" and "next" between symbols, iteration of subpatterns, and hierarchic phrase structure. As a result, measures of the relative complexity of different patterns show high agreement among the theories, and it becomes difficult to choose among alternative encoding schemes as explanations of the human performance.

In the pages of the *Psychological Review* alone, at least four apparently distinct theories have been proposed during the past decade to explain how human subjects process patterned sequences of symbols (Leeuwenberg, 1969; Restle, 1970; Simon & Kotovsky, 1963; Vitz & Todd, 1969). Additional theoretical proposals have appeared in other journals and books (e.g., Feldman, 1959, 1961; Feldman, Tonge, & Kanter, 1963; Glanzer & Clark, 1962; Gregg, 1967; Klahr & Wallace, 1970; Laughery & Gregg, 1962; Pivar & Finkelstein, 1964; Simon & Sumner, 1968; Williams, 1969). These theories differ radically from earlier explanations (in the traditions of S–R theory or stochastic learning theory) by postulating specific encodings of the sequences and by predicting from the hypothesized encodings a number of characteristics of subjects' behavior (e.g., loci of their errors, the relative difficulty of different problems, judged complexity of patterns, and so on).

In this paper it will be shown that these various theoretical proposals, so different in their surface appearance, derive from a common central core, and that it is this commonality that accounts for their success in explaining behavior. Our analysis requires us to pay careful attention to the notations in which the patterns of sequences are described (by subjects and by theorists); hence, the analysis may cast some light on the general psychological significance and implications of notational devices.

Representational (or "information-processing") explanations of serial pattern processing generally postulate some kind of language or notation[3] for expressing or de-

[1] This work was supported in whole or in part by Public Health Service Research Grant MH-07722 from the National Institute of Mental Health. I am grateful to William G. Chase, Lee Gregg, David Klahr, Kenneth Kotovsky, Allen Newell, and Paul C. Vitz for comments on an earlier draft.

[2] Requests for reprints should be sent to Herbert A. Simon, Department of Psychology, Carnegie-Mellon University, Schenley Park, Pittsburgh, Pennsylvania 15213.

[3] I shall not distinguish systematically between the terms "language" and "notation," but will generally use the former to refer to the broader features of a representation, and the latter to refer to the more particular features of a representation.

scribing the patterns, and one or more processes for inducing pattern descriptions from sequences, for storing descriptions, for reproducing sequences, and for extrapolating them. Not all of the theories incorporate all of these elements.

The theories have been applied to various kinds of sequential material, including binary sequences (Feldman, 1959, 1961; Glanzer & Clark, 1962; Restle, 1967), sequences of letters and/or digits (Restle, 1970; Simon & Kotovsky, 1963; Vitz & Todd, 1969), linear geometric figures (Leeuwenberg, 1969), intelligence test items (Williams, 1969), and musical scores (Restle, 1970; Simon & Sumner, 1968).

The tasks that have been used to test the various representational theories differ not only in the kinds of patterned sequences they use, but also in what they ask the subject to do. In some cases, he must remember and reproduce a finite (or cyclically recurring) sequence (e.g., Gregg, 1967; Laughery & Gregg, 1962; Restle, 1967, 1970; Vitz & Todd, 1969); in others, he must extrapolate a partial sequence—that is, predict its continuation (e.g., Feldman, 1959; Simon & Kotovsky, 1963; Williams, 1969); in still others, he must rank different sequences by complexity (e.g., Leeuwenberg, 1969; Vitz & Todd, 1969).

Several of the basic elements that are incorporated in the representational languages can be found in work that predates these theories. Runs of identical symbols, for example, and simple and double alternations were discussed by earlier theorists (see e.g., Estes & Burke, 1955; Goodnow & Pettigrew, 1955). I limit this comparison, however, to contemporary information-processing analyses of the phenomena.

INFORMATION PROCESSING AND INFORMATION THEORY

Information theory provides one direct method for measuring the complexity of sequences, by equating complexity with amount of information. Among the approaches under discussion here, the theory of Vitz and Todd (1969) is based on this information-theoretic measure of complexity.

The amount of information in a patterned sequence, in information-theoretic terms, is equal to the number of symbols required to describe the sequence when the maximally efficient code is used. If the coding alphabet consists of the binary digits, 0 and 1, then the information content of a sequence is equal to the number of such digits, or bits, in its code when it is coded efficiently. What code will be most efficient for representing a set of sequences depends both on the collection of sequences that belong to the set and on the relative frequency with which they occur. An encoding can be made more efficient for a given set of sequences by assigning short codes to sequences that occur frequently.

Suppose, for example, that five sequences have been encoded by the 12-symbol binary strings: 111100001111, 111111100000, 110000110000, 111111111111, and 010101-010101, respectively. Suppose that the first and fourth of these sequences occur somewhat more frequently than the second and third, and that the fifth is exceedingly rare. Then we can encode them more efficiently by making the transformations: 1111 → 111, 0000 → 110, 111 → 101, 000 → 100, 11 → 011, 00 → 010, 1 → 001, and 0 → 000. With this scheme the first four sequences are encoded: 111110111, 111101110000, 011110011110, and 11111-1111, respectively. The second and third sequences have codes of length 12, as before, but the first and fourth now have codes of only length 9. If complexity is measured by code length, then in the original encoding, all the sequences were of the same complexity; while in the new encoding, the first and fourth are simpler than the second and third. But while these particular four sequences all have codes of the same or shorter length in the new encoding as compared with the original one, this is dramatically untrue of the fifth sequence. The pattern 010101010101, also 12 symbols in length, would be recorded as 000001000001000001000001000001000001—a string of 36 symbols. The price paid for shortening the codes of some sequences is to lengthen the codes of others. The total number of encodings that can be repre-

sented by binary strings of length n is exactly 2^n, independent of the code used. The recoding will nevertheless increase average code efficiency if the fifth sequence is sufficiently rare.

Thus, information theory does not provide us with an unambiguous index of sequence complexity, but only measures complexity relative to some particular code. The complexity rankings of the sequences in a set can be changed at will by altering the encoding scheme on which the index of complexity is based. If an index of complexity is to have significance for psychology, then the encoding scheme itself must have some kind of psychological basis.

Encoding Alphabet

Also, it is not obvious why, in measuring code lengths, the binary alphabet, consisting only of the symbols 0 and 1, should be employed, rather than richer alphabets —the 10-decimal digits, for example, or the digits supplemented by the letters of the Roman alphabet. If we measure complexity by code length in *symbols* instead of code length in *bits*, then the complexity of sequences can be reduced by using a larger coding alphabet, each symbol of which can convey several bits of information. The alphabet of digits allows more than three bits to be encoded per symbol ($10 > 2^3$), while the Roman alphabet can encode between four and five bits per symbol ($2^4 < 26 < 2^5$).

Psychological theory provides good reasons why code length should be measured in terms of symbols rather than bits. If we replace the term "symbol" by "chunk," then we see that this observation is at the root of Miller's (1956) chunking hypothesis. The chunking hypothesis asserts that human short-term memory is limited in the number of symbols (or chunks) it can hold, not in the number of bits of information it can hold. Similarly, the EPAM (elementary perceiver and memorizer) theory of verbal learning asserts (Simon & Feigenbaum, 1964) that fixation of information in human long-term memory requires a constant time per symbol (or chunk), and not a constant time per bit.

Of course, when measuring complexity by code length, we cannot admit arbitrarily large coding alphabets, for these would allow us to reduce complexity to any desired level. In the limit, we could assign one letter of the alphabet to each sequence in the population to be encoded, with the result that each sequence would have a complexity of exactly one symbol. The solution to this dilemma, derivable from the EPAM theory, is to admit anything as encodable in a single symbol that has become familiar through prior learning. Anything recognizable by a subject as a "chunk," as the result of previous training or experience, is assumed to be codable into a symbol. The available coding alphabet then consists of the set of such symbols.

That the complexity of sequences must be measured relative to the psychological characteristics of the subject can be demonstrated in a variety of ways. Perhaps the most trivial is to observe that literate persons in a culture that uses the Roman alphabet will judge the sequence A B C D E to be simpler than the sequence A C E D B. Illiterate persons, and monolingual Chinese and Russians may judge the two sequences to be equally complex. Clearly, then, the presence or absence of a particular alphabet stored in long-term memory can have a major influence on judgments of complexity.

Code Length and Pattern Complexity

The theory to be proposed here takes code length as the basic measure of sequence complexity. It is postulated that most subjects in our culture will use approximately the same coding alphabets and encoding procedures. The code length of a sequence, then, means the number of symbols in the encoding when the alphabets and coding procedures common to the culture are used. We may call this encoding the *common pattern* of the sequence and we will measure the complexity of the sequence by the length, in symbols, of this pattern.

From this point of view, any particular pattern language implies a psychological

(or sociological) assertion about the alphabets and coding procedures that a subject will use when instructed to process certain sequences. While subjects' pattern language is not directly observable, it has observable consequences. If two or more pattern languages predict different complexity orderings of the sequences in some set, then empirical observations of the relative difficulty that the subject experiences with members of the set allow, at least in principle, a choice to be made of the language that corresponds most closely to the actual encoding scheme he is using. This is the underlying logic of virtually all experiments using patterned sequences as stimuli.

Now, in fact, there is almost always a high correlation among different measures of pattern complexity when several such measures are applied to the same set of patterns. Since the presence of correlation is not logically necessary, it must result from the fact that the coding schemes actually invented by investigators share some basic common features, which also lie at the base of the coding schemes used by subjects.

The next section discusses the common elements that are incorporated in the coding schemes of Simon and Kotovsky (1963), Glanzer and Clark (1962), Simon and Sumner (1968), Leeuwenberg (1969), Vitz and Todd (1969), and Restle (1970), as well as the specific differences among these schemes. These six papers represent fairly well the range of alternative coding schemes that have appeared in the literature.

PATTERN LANGUAGES

In serial tasks, both terminating and nonterminating sequences have been used. To simplify the treatment, regard all sequences as nonterminating. Thus, a sequence of finite length (e.g., 12321) will be assumed to repeat indefinitely (123211232-112$\cdots$). As is customary in programming notation, the asterisk, *, will be used as superscript to denote indefinite repetition: (12321)*.

Conventional and Descriptive Pattern Names

A sequence of symbols can be designated by a conventional name or by a descriptive name. (We call the latter the *pattern* of the sequence.) Thus "Fibonacci Numbers" is the conventional name of the sequence: 1 2 3 5 8 13 21 34 $\cdots$. This sequence can be named descriptively as "the sequence each of whose members is the sum of the two preceding members." A descriptive name for a sequence, its pattern, provides the information required by an appropriate encoder to extrapolate the sequence indefinitely. A conventional name does not provide this information; hence if only the conventional name is communicated to a decoder, the latter must already have stored the description (or its informational equivalent) if it is to be able to extrapolate the sequence.

If a pattern consisting of a finite string of symbols is to describe an infinite sequence, then the pattern must define one or more relations between each element of the sequence and preceding elements, or between each element and its order number in the sequence. An example of a pattern of the first kind is provided by the familiar formula for the Fibonacci Numbers: $F(i) = F(i-1) + F(i-2)$. An example of a pattern of the second kind is provided by a formula for the sequence of squares, 1, 4, 9, 16 $\cdots$: $S(i) = i^2$.

A pattern, then, is a finite string of symbols that states the rule governing the indefinite continuation of a nonterminating sequence. The symbols of the sequence are numbered in order, beginning with 0. The pattern must express relations between the ith symbol as the dependent variable, and one or more preceding symbols and/or constant symbols as independent variables. Thus, the language must be capable of expressing the relations themselves, the names of the symbols that enter as dependent variables, and the names of the symbols that enter as independent variables.

Consider, by way of example, the simple sequence:

ABCD $\cdots$. [1]

In the pattern to describe a sequence, the letters of the alphabet employed in the sequence can represent themselves. The letter n will represent the relation of *next* on an alphabet. The ith symbol in the sequence will be represented by Si; and the relations of sum and difference of pairs of integers, by $+$ and $-$, respectively. Using these notational devices, a pattern for Sequence 1 is

$$S0 = A; Si = n(S(i-1)); i(1:*) \quad [2]$$

where the last relation indicates that i is to range from 1 to infinity. Counting all punctuation marks as symbols, this pattern has length 24, which number we may therefore take as a measure of the complexity of Sequence 1 in this encoding.

Let us consider, in turn, each of the components of such an encoding scheme: the relations it admits, the way in which it names symbols, and the way in which it notates iteration.

Relations

What relations are available for defining the pattern of a sequence depends on the properties of the set of symbols, S, from which the sequence is constructed. For any set of symbols, S, we can define the identity relation (s, for "same") on the product set $S \times S$. Using this relation, we can describe simple patterns. Thus the pattern:

$$AQAQAQ \cdots. \quad [3]$$

can be defined by:

$$S0=A; S1=Q; Si=s(S(i-2)); i(2:*) \quad [4]$$

(This pattern can also be defined simply by: $S(2i)=A; S(2i+1) = Q; i(0:*)$.)

If the symbol set, S, is ordered, that is, constitutes an alphabet, then we can also define on $S \times S$ the successor relation (n, for "next"), and its inverse (p, for "immediate predecessor"). Moreover, these two relations, together with s, are the only relations that are common to all ordered sets.[4] With two exceptions, discussed below, they are also the only relations that enter into the serial pattern tasks that have been used in psychological experiments and tests.

The relations s and n are central to the pattern language developed by Simon and Kotovsky (1963) and extended by Simon and Sumner (1968) and by Williams (1969). Both play an equally central role in Leeuwenberg's (1969) pattern language [5] and in Restle's (1970). Restle calls the s relation r (for "repeat") and the n relation t (for "transpose").

Glanzer and Clark (1962) and Vitz and Todd (1969) limit themselves to binary and ternary sets of symbols (the sets 0,1; a,b; and a,b,c), hence make no use of the n relation, but only of the s relation. Therefore, their pattern languages are both simpler and more limited in range of application than the languages used by the other authors.

We come now to two additional kinds of relations that are used by Simon and Sumner (1968; and subsequently by Restle, 1970) to describe certain patterns—relations that are relevant only to the alphabet of integers. With the integers, we may use, first of all, the relations of *sum* ($+$) and *difference* ($-$). These can be defined in terms of the successor and predecessor relations, as in the Peano axiomatization of arithmetic (e.g., $(i+2)=n(i+1)=n^2(i)$; $(i-1) = p(i)$), but it is often more convenient, notationally, to retain the arithmetic relations as well as the successor and predecessor relations.

Finally, with the alphabet of integers, we may also define the *complement* relation (denoted ck, where k is an integer). To see what this means, consider the sequence:

$$123321 \quad [5]$$

[4] More precisely, all other relations that are common to all ordered sets can be defined in terms of these, for these are the only relations required for the definition of an ordering.

[5] The resemblance of Leeuwenberg's language to the others is not easy, at first, to detect underneath his notation. An integral sign plays the role of n in his language, and parentheses the role of s. Thus, the pattern $\int(1)$ describes the sequence 1234 $\cdots$ in his notation. That is to say, (1) denotes (1)*, and the integral sign designates successive summations: 1, 1+1, 2+1, etc. In the notation of Simon and Kotovsky (1963), this same sequence would be encoded as $[\mu \leftarrow$ integers$]$ $(n(\mu))^*$; in Restle's (1970) notation, it would be encoded as $N^*(1)$.

Here $S5 = S0$, $S4 = S1$, $S3 = S2$, or, in general, $Si = S(5 - i)$; $i(0:5)$. Defining $ck(i)$ as the k's complement of i, we can write instead: $Si = S(c5(i))$.

Restle (1970) introduces a relation which he calls m (for "mirror") in order to permit complementation. He does not, however, use the relation consistently, and hence his various examples do not all fit the same definition of it.[6]

In the example above, the complementation relation is applied to the postscripts designating the locations of symbols. If the sequence uses the alphabet of integers, the relation can also be applied to the symbols themselves. That is to say, Sequence 5 can also be described by $Si = c4(S(i - 3))$; $i(3:5)$; $Sj = (j + 1)$; $j(0:2)$.

In summary, all of the pattern languages under consideration describe sequences with the help of the relations s, n, p, $+$, $-$, and ck, or some subset of these. The latter three relations are defined only on integers and are in any case definable in terms of n and p. The first three relations are defined on pairs of symbols drawn from any alphabet (ordered set); hence, they are common structural elements for virtually all kinds of linear patterns.

Designation of Symbols

While all of the pattern languages use essentially the same small set of relations, they differ widely in the notation they employ to name or designate the symbols that enter into the relations. There appear to be three different ways to designate symbols: (*a*) naming them explicitly by their positions in the sequence (e.g., $S4$ is the symbol that follows $S3$ and precedes $S5$), (*b*) displaying relations in the pattern in the order in which the corresponding symbols appear in the original sequence, so that the pattern becomes a "template" for the sequence,[7] and (*c*) notating successive runs of identical symbols by numbers corresponding to the run lengths.[8] In the examples thus far, we have used the first method, naming each symbol explicitly by postscripting "S" with its order number in the sequence (numbering from zero). This is a very flexible notation, but it is not compact. Consider the sequence:

$$\text{ABMCDM} \cdots. \qquad [6]$$

In this notation the sequence would be described by the relations:

$$\begin{aligned} &S0 = \text{A}; S3i = nS(3i - 2); \\ &S(3i - 2) = n(S3(i - 1)); \\ &S(3i - 1) = \text{M}; i(1:*) \end{aligned} \qquad [7]$$

The postscripts do not reflect very clearly the periodic or hierarchic structure of Sequence 6, in which each triad of symbols repeats the relations of the previous triad. The periodicity becomes more visible if each symbol is designated by a double postscript, the first component denoting the period, and the second, the location (0, 1, or 2) within the period. With this scheme, the second symbol in the sixth period would be designated $(S(5,1))$. The pattern can now be rewritten as:

$$\begin{aligned} &S0 = A; S(i + 1, 0) = nS(i,1); \\ &S(i,1) = nS(i,0); \\ &S(i,2) = \text{M}; i(0:*) \end{aligned} \qquad [8]$$

This notation, employed by Simon and Sumner (1968), still does not fully reveal the structure of Sequence 6, which consists of a simple alphabetic sequence (the first two symbols of each period) interwoven with a repetition of the letter M. The notation can be further simplified, using the second method of designating symbols, by writing down the relations in a definite order, and by introducing names for *working memories*.

[6] The example in his Figure 2, as well as the first two examples in his Table 1, is inconsistent with his definition of the relation m.

[7] More formally, this scheme retains some of the information about the order of symbols in the sequence by an order-preserving many-one mapping of symbols of the sequence onto subsequences of the pattern. The number of symbols mapped onto successive sub-sequences of the pattern defines the *period* of the sequence.

[8] As we shall see, this scheme is a specialization and abstraction of the preceding one that eliminates redundant symbols from the pattern, retaining only the superscripts that denote numbers of iterations.

A working memory, call it μ, contains the name of an alphabet and a pointer to a particular location of the alphabet. Moving the pointer locates the *next* location on the alphabet. Using this idea we can notate the pattern of Sequence 6 by:

$$[\mu \leftarrow \alpha](n(\mu)n(\mu)\mathrm{M})^* \qquad [9]$$

which can be read: "Set the pointer in μ to the name of the Roman alphabet, α. Move the pointer one place down the alphabet (i.e., to $n(\mu)=\mathrm{A}$), then one more place (i.e., to $n(\mu) = \mathrm{B}$); then produce the letter M; repeat these three processes indefinitely." (The square brackets in Pattern 9 denote an operation that does not produce a symbol.)

This new notation abandons entirely the use of postscripts to designate symbol locations. Instead, the rules for the three positions in the basic triad are written down, one following the other, in order.

As a result of these two changes in notation—introducing working memories and dispensing with the explicit names for symbols at designated locations—the description is greatly abbreviated. In place of the 52 symbols in Pattern 7, or the 51 in Pattern 8, Pattern 9 contains only 17 symbols. The notation of Pattern 9 is essentially that introduced in Simon and Kotovsky (1963).

Restle (1970) later introduced a different device to permit the suppression of explicit names for symbols. He employed operations upon *sequences* of symbols, the effect of such an operation being to change all of the symbols in the sequence in exactly the same way. To see the relation between his notation and that of Simon and Kotovsky, we denote by S_i a sequence of symbols, $S1$, $S2$, $\cdots$. Then we can define, for any operator, o,

$$o\{S_i\} = \{o(Si)\} \qquad [10]$$

so that, for example,

$$n\{\mathrm{A,B,C}\} = \{n(\mathrm{A}),\ n(\mathrm{B}),\ n(\mathrm{C})\} = \{\mathrm{B,C,D}\}.$$

Restle also allows concatenation of operators:

$$(o1,\ o2)\{\mathrm{S}_i\} = o1\{S_i\}o2\{S_i\} \qquad [11]$$

that is, the effect of applying the concatenated operator, $(o1,o2)$, to a sequence $\{S_i\}$ is to produce the sequence $o1\{S_i\}$ followed by the sequence $o2\{S_i\}$.

Restle introduces one other useful notational device. Let a lowercase letter, o, designate any operator; then the corresponding capital letter, O, designates the concatenation of s with o. That is,

$$O = s,o \qquad [12]$$

so that $O\{S_i\} = s\{S_i\}o\{S_i\} = \{S_i\}\{o(S_i)\}$. While Restle's notational innovations are applicable only to very special classes of patterned sequences, they permit sequences belonging to these classes to be described perspicuously and compactly. Consider, for example,

$$\mathrm{DEFGEFGHFGHI}\ \cdots. \qquad [13]$$

In Restle's notation, this can be described by:

$$N^*(N^3(\mathrm{D})) \qquad [14]$$

where

$$N^i\{S_i\} = \{S_i\}N^{i-1}\{n(S_i)\};$$
$$N\{S_i\} = \{S_i\}\{n(S_i)\},$$

whence:

$$N^*(N^3(\mathrm{D}))\rightarrow N^*(\mathrm{D}N^2(\mathrm{E}))\rightarrow N^*(\mathrm{DE}N(\mathrm{F}))$$
$$\rightarrow N^*(\mathrm{DEF}\ n(\mathrm{F}))\rightarrow N^*(\mathrm{DEFG})$$
$$\rightarrow[\mathrm{DEFG}\ N^*(\mathrm{EFGH})]$$
$$\rightarrow[\mathrm{DEFGEFGH}\ N^*(\mathrm{FGHI})]\rightarrow \text{etc.}$$

In the Simon and Kotovsky (1963) notation, the description of Sequence 13 might take the form:

$$[\mu_2\leftarrow \mathrm{D}]([\mu_1\leftarrow\mu_2]\mu_1[n(\mu_1)]^3[n(\mu_2)])^* \qquad [15]$$

Pattern 14 requires only nine symbols, while Pattern 15 requires 27 to describe Sequence 13.

On the other hand, many sequences cannot be described at all in Restle's special notation—Sequence 6 being a simple example. The reason is easy to see. In Sequence 6, unlike Sequence 13, the symbols of the ith period cannot be obtained by applying the same operator to the corresponding symbols of the previous period.

The notation used by Leeuwenberg (1969) resembles most closely that of Simon and Kotovsky (1963). In fact, when the 15 patterns of the Thurstone Letter Series Completion Test are encoded in both notations, the pattern length in the Simon and Kotovsky notation has a .8 correlation with the pattern length in Leeuwenberg's notation. Leeuwenberg used a rather elaborate system of punctuation to obviate the need for naming working memories explicitly. Essentially, his idea in encoding a pattern like Sequence 6 is to describe separately each of the intertwined subpatterns, then to indicate which of the symbols in each period belongs to each of the subpatterns. Sequence 6, for example, is constructed from two subpatterns, one providing the first two symbols in each period; the other, the third symbol. (Details of Leeuwenberg's encoding, which involves the notation of "runs" of the subpatterns [see below] can be found in his paper.)

In the schemes of Glanzer and Clark (1962) and Vitz and Todd (1969), matters become simpler because of the restriction to binary (or ternary) alphabets and to the relation s. In the binary case, a periodic pattern can be described in terms of the successive run lengths of 1s and 0s, respectively. Thus, the sequence

$$00100011111\cdots \qquad [16]$$

can be encoded as (02135)*. (By convention the sequence starts with a run of 1s, in this case, of length zero.) The longer the runs, on average, the more efficient is this encoding. The least parsimonious description—that for a simple alternation—is simply the sequence itself.

With a ternary alphabet, each run can be described by juxtaposition of the symbol and run length. Thus,

$$AAABBBCCC\cdots \qquad [17]$$

can be described as (A3B3C3)*.

In terms of our previous notations, we could describe Sequence 16 by:

$$((s(0))^2\ (s(1))^1\ (s(0))^3\ (s(1))^5)^* \qquad [18]$$

Thus the run-length notation can be derived from the more general one by omitting all of the redundant, predictable symbols in the explicit notation and retaining only the superscripts.

Still more complicated codes may facilitate the description of special classes of binary patterns. If patterns consist, for example, of runs *and* simple alternations on a binary alphabet, then it may be efficient to recode into a ternary alphabet—$1 \rightarrow \alpha$, $0 \rightarrow \beta$, $01 \rightarrow \gamma$—and then use the ternary code suggested above for Sequence 17 (cf. Feldman, 1961).

Where one symbol occurs much more frequently than the other in a binary sequence, a more efficient coding may be achieved by simply mentioning those positions in the sequence that are occupied by the rarer symbol, for example $S1 = S12 = S19 = 1$. By convention, any symbol not explicitly mentioned is the common one (in the example, 0). This method of abbreviation is a special case of a more general Principle of Exceptions. Normal values are defined for certain symbols, and the symbol is not explicitly mentioned when it takes on its normal value. Thus, in the Simon and Kotovsky (1963) notation, we can take s as the normal value of the operator on a working memory, and simply write μ in place of $s(\mu)$ wherever the latter occurs.

Basically, all of the representations we are considering describe sequences by defining the relations that determine symbols in terms of preceding symbols. The length of code for any given sequence (hence the complexity of the pattern) depends basically on the number of relations that must be defined. However, some coding economies can be realized for special classes of sequences by using the devices just mentioned; for example, by omitting mention of common elements, or by exhibiting explicitly only nonnormal values of symbols.

These devices are used by human subjects to achieve succinct descriptions of sequences. Glanzer and Clark (1962), whose method involved asking subjects to report patterns verbally, found that the subjects generally described binary patterns either

in terms of runs (i.e., in a language like that used above to describe Sequence 16, or by naming the location of the "exceptional" symbols (e.g., 1s in positions 2 and 6"—a simplification, for binary sequences, of the Simon & Sumner, 1968, language).

Vitz and Todd (1969), as previously noted, constructed an information-theoretic measure of pattern complexity. The encoding system underlying their measure was based on the detection in sequences of runs of individual symbols and runs of subpatterns (see Vitz & Todd, 1969, pp. 435–436). As a result, there is a very high correlation between their information measure and a simple count of runs. Table 1 shows, for four sets of data presented by Vitz and Todd, the correlation coefficients (*a*) between Vitz and Todd's information measure (H) and sequence complexity (C) as judged by subjects; (*b*) between H and length of code (L), based on runs; and (*c*) between C and L.

We see from Table 1 that H accounts for 80% to 95% of the variance in judged C of the sequences. Except in one case, where the correlation is .79, the correlation of H with L is around .9. Hence variation in code length accounts for about 80% of the variance in amount of information contained in these sequences. If H were the "true" estimator of C, and L an approximate surrogate for H, otherwise unrelated to C, then the correlation between L and C would be $r'_{LC} = r_{HC} \times r_{HL}$. As Table 1 shows, the actual correlation, r_{LC}, is always larger than r'_{LC}. Hence code length makes a (small) independent contribution to the prediction of complexity, over and above the part explanable by the relation of L to H.

In summary, the pattern languages we are examining use a variety of notations to designate the symbols between which specified relations hold. There is a tradeoff, in general, between the generality of a notation—the range of pattern types it can describe—and its efficiency in describing particular types of patterns. Of the notations reviewed here, that of Simon and Sumner (1968) is the most explicit and general, but also the most "verbose" in designating locations. The Simon and Kotovsky (1963) notation is considerably more concise, giving up only a little generality. The notation of Restle (1970) is quite specialized but parsimonious where applicable. The coding schemes considered by Glanzer and Clark (1962) and Vitz and Todd (1969) are even more specialized and concise. They are obtained by omitting all the information that is redundant for the special sequences to which they are applicable.

TABLE 1

CORRELATION OF CODE LENGTH, INFORMATION CONTENT, AND JUDGED COMPLEXITY OF FOUR SETS OF PATTERNED SEQUENCES

Variable		Set of sequences			
		1	2	3	4
1. H_{VT} × complexity	r_{HC}	.98	.95	.94	.91
2. H_{VT} × code length	r_{HL}	.92	.79	.91	.88
3. Code length and complexity	r_{LC}	.91	.79	.89	.90
4.	r'_{LC}	.90	.75	.85	.80

Note.—H_{VT} = Vitz-Todd information measure; code length = number of runs of identical symbols; complexity = judged complexity (Sets 1–3), or number of reproduction errors (Set 4). Data for the four sequence sets from Vitz and Todd (1969), Tables 6, 7, 8, and 9, respectively. r'_{LC} = predicted correlation between code length and complexity. $r'_{LC} = r_{HC} \times r_{HL}$.

Iteration and Hierarchy

Most of the coding schemes use numerical superscripts or subscripts to indicate repetition of pattern elements—operators, symbols, or subpatterns. Thus n^2 is a convenient notation for "next of next," so that $n^2(A) = n(n(A)) = C$. Similarly, $[\mu \leftarrow A](n(\mu))^3$ provides a shorthand for $[\mu \leftarrow A](n(\mu)n(\mu)n(\mu)) = BCD$. We have seen how a run-length code can be derived by abstraction from such a notation for iteration:

$$11100111100 \rightarrow (1)^3(0)^2(1)^4(0)^2 \rightarrow 3242.$$

The structure of patterns is not limited to simple periodicity, but may involve a hierarchy of periods. Restle (1970) has paid particular attention to hierarchic patterns, which were earlier used to a

TABLE 2

RESULTS OF OPERATION

Operations	Push-down list	Symbols produced	Restle's operators
$[\mu \leftarrow 1]$	1		1
A push	11		
B push	111		
$\mu\, n(\mu)$	211	12	N
Pop	11		
C $(\ )^2$	(Repeat from B to C)	12	S
$[n(\mu)]$	21		
D $(\ )^2$	(Repeat from B to D)	23 23	N
Pop	1		
E $(\ )^2$	(Repeat from A to E)	12 12 23 23	S
$[C7(\mu); n \leftarrow p]$	6		
F $(\ )^2$	(Repeat from A to F)	(65 65 54 54)^2	C7

limited extent by Kotovsky and Simon,[9] and extensively by Simon and Sumner (1968) in their description of music.

Consider Restle's (1970) example:

12122323121223236565545465655454 [19]

The hierarchic phrase structure of this sequence can be made visible by punctuation:

(12 12) (23 23), (12 12) (23 23);
(65 65) (54 54), (65 65) (54 54), [20]

With our adaptation of Restle's notation, we can describe the sequence succinctly, thus:

C7(S(N(S(N(1))))), [21]

and from this pattern we can regenerate the sequence again by successive steps:

C7(S(N(S(N(1)))))→C7(S(N(S(12))))
→C7(S(N(12 12)))→C7(S(12 12 23 23))
→C7(12 12 23 23 12 12 23 23)
→(12 12 23 23 12 12 23 23
65 65 54 54 65 65 54 54) [22]

There is a serious objection to taking Pattern 21 as a hypothesis of how a subject might encode Sequence 20. As Sequence 22 shows, regenerating the sequence from the pattern calls for a short-term memory capacity of at least 16 symbols of the sequence as input.

[9] K. Kotovsky and H. A. Simon. Empirical tests of a theory of human acquisition of concepts for sequential patterns. In preparation.

An alternative encoding, based on the Simon and Kotovsky (1963) notation, avoids this difficulty. To explain it, it is necessary to introduce the concept of *push-down list.* A push-down list is simply a sequence that can be altered only by adding a symbol to, or taking a symbol from, its front end. Let us call *push* the operation of duplicating the first symbol of a push-down list, and *pop* the operation of removing the first symbol. Then push(ABC) = AABC, while pop(ABC) = BC.[10] We can use a working memory, μ, holding a push-down list, and these operations to encode Sequence 19 as:

$[\mu \leftarrow 1]((\text{push}\ ((\text{push}\ \mu\, n(\mu)\ \text{pop})^2[n(\mu)])^2\ \text{pop})^2[C7(\mu); n \leftarrow p])^2$ [23]

where $[n \leftarrow p]$ replaces Operation n by Operation p everywhere throughout the pattern.

Let us follow step by step (see Table 2) the generation of the sequence from Pattern 23, indicating also the correspondence between these steps and Restle's operators in Pattern 21.

Now in this process, unlike Restle's, only the symbols in the push-down list need be held in short-term memory (a maximum of three symbols), not all of the symbols of the sequence already produced (a maximum of 16 symbols). From this standpoint, the representation employing the push-down list provides a more plausible model of the psychological process than the recursive representation. The same argument applied to Patterns 14 and 15 for generating Sequence 13 shows that the former pattern (Restle, 1970) calls for a maximum of four symbols in short-term memory, the latter (Simon & Kotovsky, 1963), a maximum of two. With the use of a push-down list, Pattern 15 could be rewritten:

$[\mu \leftarrow \text{D}]\ (\text{push}\ \mu\ (n(\mu))^3\ \text{pop}\ [n(\mu)]^*$ [25]

[10] Push-down lists were invented along with computer list-processing languages in the mid-1950s. They have been used widely in implementing parsing programs (so-called "push-down automata") for computing languages and natural languages. Hence, the notion that a push-down list may be part of the organization of short-term memory already has some currency and support in psycholinguistics (see Yngve, 1960).

In Restle's (1970) recursive scheme, the short-term memory requirements depend on the length of the period at the highest level of the pattern hierarchy. This number can be expected to increase (approximately) geometrically with number of levels. In the scheme using a push-down list, the short-term memory requirements increase only linearly with number of levels in the hierarchy.

Restle has produced some data that demonstrate quite convincingly (e.g., Restle, 1970, Figure 3) the psychological reality of the hierarchic phrase structure of patterns. However, his data are equally compatible with the encodings represented by Patterns 21 and 23, for both postulate the same phrase structure in the pattern.

In summary, various of the encoding schemes make provisions for iteration of subpatterns and for hierarchic phrase structure of patterns. Empirical evidence can be (and has been) produced to demonstrate that subjects, in fact, encode some patterns hierarchically. The different encoding schemes place quite different demands on short-term memory, however, in generating sequences from their patterns.

ADDITIONAL PSYCHOLOGICAL IMPLICATIONS

The analysis up to this point has been concerned mainly with describing different languages that have been proposed for encoding patterned sequences. We have introduced behavioral evidence only occasionally to elucidate some particular point of comparison or contrast. The remainder of this paper is concerned with additional implications of coding languages for psychological theory.

If a particular sequence can be encoded, and represented internally, in several different ways, then it is important to explain what determines which encoding a human subject will use. If different subjects use different encodings, it is important to explain the causes for the difference, and its consequences in terms of task performance.

The tasks that subjects have performed in the laboratory with patterned sequences include: (*a*) ranking sequences according to complexity, (*b*) inducing patterns from sequences, (*c*) describing sequences verbally, (*d*) learning and remembering sequences, and (*e*) extrapolating sequences.

Task *a* only produces observable behavior when combined with *c*, *d*, or *e*. Any one of the latter can be used, in turn, to produce measures of sequence "difficulty" —that is, difficulty can be measured by the time required to perform any one of these tasks with a given sequence, or by number of errors. (Not only the number of errors, but also the precise nature of the errors can be determined.)

A priori measures of sequence complexity can also be constructed and these can be correlated with sequence difficulty as measured by subjects' behavior. As we have seen, sequence complexity may be measured by code length or by information content; the two kinds of measures are mutually translatable, the one into the other; and both kinds of measures are defined relative to a specific encoding scheme.

Correlation of A Priori Complexity Measures

When two or more a priori complexity measures are computed for the same set of sequences, the correlation among these will indicate the similarities of the encodings or, conversely, the sensitivity of the measures to details of encoding.

We have already seen that Leeuwenberg's (1969) measure correlates .8 with Simon and Kotovsky's (1963) when both are applied to a set of 15 sequences from the Thurstone Letter Series Test, and that this high correlation is a consequence of the basic similarities in the coding concepts underlying both schemes.

Similarly, a simple computation verifies that Vitz and Todd's (1969) information measure correlates at about the .9 level with code length when binary and ternary sequences are encoded in terms of simple run lengths.

Since we emphasized at the outset that code length is relative to the coding language, it is not obvious why these high correlations occur. They occur because the various coding schemes are not unrelated but are all based on the same set of rela-

tions (n, p, s, c), and on notation for iteration. When we compare Patterns 14 and 15 for Sequence 13, for example, we see that, however more compact Pattern 14 is, each symbol in that pattern corresponds to a symbol, or set of symbols, in Pattern 15. Thus, we have the correspondences:

$$N^* \leftrightarrow ([\mu_1 \leftarrow \mu_2] \cdots [n(\mu_2)])^*$$
$$N^3 \leftrightarrow \mu_1(n(\mu_1))^3$$
$$D \leftrightarrow [\mu_2 \leftarrow D].$$

As long as we do not change the basic relations used, we can expect that *relative* complexity of sequences will be rather insensitive to details of the coding, providing that the sequences under consideration are not too heterogeneous. Hence, our ability to predict measures of behavior may not depend on our knowing in detail the exact encoding scheme that subjects use. If this is so, it will be difficult to choose alternative encoding hypotheses without sophisticated analysis of the behavioral data.

Correlation of A Priori with Judged Complexity

A major use of complexity measures (see e.g., Vitz & Todd, 1969) has been to predict complexity as judged by subjects from an a priori measure of complexity. For the reasons just stated, the achievement of high correlations cannot be taken as a validation of the specific a priori measure used. Rather, the high correlations should be interpreted to imply: (*a*) that subjects confronted with sequences do in fact encode them with the aid of the relations we have discussed, and (*b*) that the length of the encoded pattern is the basis for the subjects' judging the complexity of the sequence.

This general hypothesis was put forth independently by Glanzer and Clark (1962) and by Simon and Kotovsky (1963). Glanzer and Clark went further, however, and, in their "verbal loop hypothesis," asserted that the encoding was verbal (i.e., consisted of a sequence of English words and phrases). They said, specifically:

> [The subject] carrying out a perceptual recall task puts the information through a verbal loop. (1) He translates the visual information into a series of words. (2) He holds the verbalization and makes his final response on the basis of that. . . . Complexity, under this hypothesis, becomes identified with the length of the S's [subject's] verbalization [p. 295].

There is nothing in the data, however, to demonstrate that the internal representation is verbal. The verbal descriptions that Glanzer and Clark elicited from their subjects could equally well have been a fairly direct recoding into verbal form of the internally represented symbolic (but nonverbal) structures. It seems preferable, therefore, to adopt the more neutral stance of Simon and Kotovsky (1963) regarding the nature of the internal representation:

> Subjects attain a serial pattern concept by generating and fixating a pattern description of that concept. . . . [Subjects] have stored in memory a program capable of interpreting and executing descriptions of serial patterns [p. 538].

Complexity and Difficulty

A second use of complexity measures, whether a priori or judged complexity, is to predict the relative difficulty subjects will experience in performing specific tasks on different sequences.

It is not obvious that the ranking of sequences by difficulty will be the same for all tasks—the same for describing sequences verbally as for learning periodic sequences as for extrapolating nonperiodic sequences. To describe a sequence verbally, the subject must (*a*) discover the pattern and (*b*) hold the pattern in memory (short term or long term) while he produces it (Glanzer & Clark, 1962). To learn a periodic sequence he must either (*a*) fixate the sequence by rote or (*b*) discover the pattern, hold it in memory, and use it to generate the sequence (Gregg, 1967). To extrapolate a nonperiodic sequence he must (*a*) discover the pattern, (*b*) remember it, and (*c*) use it to generate the sequence (Simon & Kotovsky, 1963).

An EPAM-like learning theory would suggest that three or four different sources of task difficulty are present in different combinations in these tasks: (*a*) difficulty

of *discovering* a pattern, (*b*) difficulty of *fixating* either a sequence or a pattern in *long-term memory*, and (*c*) difficulty of holding in *short-term working memory* the place keepers or pointers required to produce the sequence from the pattern.

However, disentangling these components of difficulty empirically is not at all easy. In general, the more complex the pattern (as measured by code length), the more relations it is necessary to induce from the sequence in order to find the pattern, the more symbols it is necessary to fixate in order to store the pattern in long-term memory, and the more working memories are necessary in order to produce the sequence from the pattern. Because of this confounding of components of difficulty, all investigators have found code length to be a reasonably good predictor of difficulty, in whichever way difficulty is measured. Thus, Glanzer and Clark (1962) report a correlation of −.826 between mean verbalization length (code length) and accuracy of recall; Vitz and Todd (1969), using the same data, find a correlation of −.82 between information content (which is interpreted here as an alternative estimate of code length) and accuracy of recall. Leeuwenberg's (1969) data show correlations of .65 and .66 between two measures of difficulty (errors and time, respectively, for a group of 12 subjects) on the Thurstone Letter Series Task and code length, using his encoding scheme; there are similar correlations, of .74 and .61, between these same two measures of difficulty and code length with the Simon and Kotovsky (1963) coding scheme.

Direct Evidence of Encoding

Three kinds of "direct" evidence have been sought to determine the actual encodings used by subjects. Glanzer and Clark (1962) simply instructed the subject to describe each sequence, with the results already mentioned. Kotovsky (1970) and Kotovsky and Simon (see Footnote 9) obtained verbal protocols and other evidences of subjects' sequential behaviors while discovering the patterns in a set of sequences. Williams (1969) obtained a combination of verbal protocols and eye-movement data. The patterns inferred from this direct evidence generally fit well the Kotovsky and Simon coding language. While numerous individual differences were found, these almost all fell within the general range of concepts allowed by the formal language—that is, it was usually possible to express the variant encodings in that language.

For example, in the specific case of Sequence 13, some eight of Kotovsky and Simon's 14 subjects used a pattern description corresponding closely to Pattern 15 or 23, and only one a description resembling Pattern 14. On the other hand, with the slightly simpler pattern, PON ONM NML $\cdots$, four subjects used an encoding resembling Restle's (1970). Restle's description only requires three working memories to handle the latter pattern, but four to handle Pattern 14, and this difference in the demands upon working memory may account, in whole or part, for the preferences among encodings.

Gregg (1967) has also provided direct evidence on how subjects describe a pattern they are trying to discover and, more important yet, how their mode of description affects the ease or difficulty of learning the pattern.

CONCLUSION

A survey of the principal alternative formulations of a theory to explain human performance in tasks involving patterned sequences shows all of these formulations to be mild variants on a basic theme. The formulations agree in proposing the following: (*a*) that subjects perform these tasks by inducing pattern descriptions from the sequences; and (*b*) that these pattern descriptions incorporate the relations of *same* and *next* (on familiar alphabets) between symbols, iteration of subpatterns, and hierarchic phrase structure.

As a result of this concensus on fundamentals, measures of the relative complexity of different patterns can be expected to be nearly the same, independently of which specific encoding languages are used in computing them. A little direct

evidence is now available to help us understand what circumstances subjects will use one or another of these encodings (or others).

It is quixotic to seek crucial experiments to determine *the* coding language, because (*a*) different subjects almost surely use different representations for the same sequences, and (*b*) the representations used may well be modified by task instructions, previous experience, and the character of the set of sequences used in the task. Empirical evidence on these points has already been mentioned. The same conclusion is suggested by the fact that the correlations between subjects in ratings of complexity are far from perfect. (In their paper, Vitz & Todd, 1969, mention mean or median among-subject correlations of .67 and .84 for two sets of data.) Instead of seeking to discover "the" pattern code or "the" complexity measures, we need to determine the conditions (sociological and psychological) under which subjects will adopt one or another coding scheme in handling patterned sequences.

REFERENCES

ESTES, W. K., & BURKE, C. J. Application of a statistical model to simple discrimination learning in human subjects. *Journal of Experimental Psychology,* 1955, **50**, 81–88.

FELDMAN, J. An analysis of predictive behavior in a two-choice situation. Unpublished doctoral dissertation, Carnegie-Mellon University, 1959.

FELDMAN, J. Simulation of behavior in the binary choice experiment. *Proceedings of the Western Joint Computer Conference,* 1961, **19**, 133–144.

FELDMAN, J., TONGE, F. M., JR., & KANTER, H. Empirical explorations of a hypothesis-testing model of binary choice behavior. In A. C. Hoggatt & F. E. Balderston (Eds.), *Symposium on simulation models.* Cincinnati, Ohio: South-Western Publishing, 1963.

GLANZER, M. S., & CLARK, H. H. Accuracy of perceptual recall: An analysis of organization. *Journal of Verbal Learning and Verbal Behavior,* 1962, **1**, 289–299.

GOODNOW, J. J., & PETTIGREW, R. F. Effect of prior patterns of experience upon strategies and learning sets. *Journal of Experimental Psychology,* 1955, **49**, 381–398.

GREGG, L. W. Internal representation of sequential concepts. In B. Kleinmuntz (Ed.), *Concepts and the structure of memory.* New York: Wiley, 1967.

KLAHR, D., & WALLACE, J. G. The development of serial completion strategies; an information processing analysis. *British Journal of Psychology,* 1970, **61**, 243–257.

KOTOVSKY, K. An empirical test of the Simon and Kotovsky "concept former" model of human letter series sequence extrapolation behavior. Unpublished master's thesis, Carnegie-Mellon University, 1970.

LAUGHERY, K. R., & GREGG, L. W. Simulation of human problem-solving behavior. *Psychometrika,* 1962, **27**, 265–282.

LEEUWENBERG, E. L. L. Quantitative specification of information in sequential patterns. *Psychological Review,* 1969, **76**, 216–220.

PIVAR, M., & FINKELSTEIN, M. Automation, using LISP, of inductive inference on sequences. In E. C. Berkeley & D. G. Bobrow (Eds.), *The programming language LISP.* Cambridge, Mass.: Information International, 1964.

RESTLE, F. Grammatical analysis of the prediction of binary events. *Journal of Verbal Learning and Verbal Behavior,* 1967, **6**, 17–25.

RESTLE, F. Theory of serial pattern learning: Structural trees. *Psychological Review,* 1970, **77**, 481–495.

SIMON, H. A., & FEIGENBAUM, E. A. Elementary perceiver and memorizer: Review of experiments. In A. C. Hoggalt & F. E. Balderston (Eds.), *Symposium on simulation models.* Cincinnati, Ohio: South-Western Publishing, 1963.

SIMON, H. A., & KOTOVSKY, K. Human acquisition of concepts for sequential patterns. *Psychological Review,* 1963, **70**, 534–546.

SIMON, H. A., & SUMNER, R. K. Pattern in music. In B. Kleinmuntz (Ed.), *Formal representation of human judgment.* New York: Wiley, 1968.

VITZ, P. C., & TODD, R. C. A coded element model of the perceptual processing of sequential stimuli. *Psychological Review,* 1969, **76**, 433–449.

WILLIAMS, D. S. Computer program organization induced from problem examples. In H. A. Simon & L. Siklóssy (Eds.), *Representation and meaning.* Englewood Cliffs, N. J.: Prentice-Hall, 1972.

WILLIAMS, D. S. Computer program organization induced from problem examples. Unpublished doctoral dissertation, Carnegie-Mellon University, 1969.

YNGVE, V. A model and a hypothesis for language structure. *Proceedings of the American Philosophical Society,* 1960, **104**, 444–446.

(Received April 24, 1972)